МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чернігівський національний технологічний університет

ІМІТАЦІЙНЕ МОДЕЛЮВАННЯ НА JAVA

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторного практикуму та самостійної роботи
з дисципліни

«Моделювання систем»

для студентів спеціальності
128 – "Комп'ютерна інженерія"

ЗАТВЕРДЖЕНО
на засіданні кафедри
інформаційних і комп'ютерних систем
протокол № 1 від 29.08.18

Чернігів ЧНТУ 2018

Імітаційне моделювання на Java. Методичні вказівки до лабораторного практикуму та самостійної роботи з дисципліни «Моделювання систем» для студентів спеціальності 128 – „Комп'ютерна інженерія”. /Укл.: Бивойно П.Г. - Чернігів: ЧНТУ, 2018. - 137 с.

Укладач: Бивойно Павло Георгійович, канд. техн. наук, доцент,

Відповідальний за випуск: С.В. Зайцев, доктор. техн. наук, зав. кафедри інформаційних та комп'ютерних систем Чернігівського національного технологічного університету

Рецензент: В.А. Бичко, канд. ф-м. наук, доцент кафедри інформаційних і комп'ютерних систем Чернігівського державного технологічного університету.

ЗМІСТ

Вступ

Методичні вказівки призначені для використання при виконанні лабораторних робіт з дисципліни «Моделювання» і для самостійної роботи з вивчення курсу.

Перша лабораторна робота стосується питань теорії імовірностей і має на меті спонукати студентів оновити знання, що пов'язані з поняттям імовірності.

У другій роботі розглядаються питання створення, тестування та використання генераторів випадкових чисел.

Третя і четверта роботи присвячені вирішенню проблем створення імітаційної моделі і програмної організації взаємодії паралельно працюючих об'єктів.

У п'ятій роботі розглядаються питання експериментального та теоретичного дослідження простої СМО і проводиться порівняння результатів імітаційного моделювання з теоретичними результатами, що отримані за допомогою ланцюгів Маркова.

Шоста робота присвячена питанням планування однофакторних експериментів з моделями та дисперсійному і регресійному аналізу отриманих результатів.

У сьомій роботі розглядаються питання дослідження перехідних процесів у СМО та плануванню експериментів пов'язаних з пошуком екстремумів.

Особливість методичних вказівок полягає також у тому, що в них докладно розглядається реалізація програмного комплексу, що використовується при виконанні лабораторних робіт. Це дає можливість студентам познайомитися з реальними програмними рішеннями на мові Java. Запропоновані програмні рішення базуються на об'єктно-орієнтованому підході, тому робота студентів з методичними вказівками буде сприяти закріпленню знань і навичок, отриманих при вивченні курсу «Об'єктно-орієнтованого програмування».

Методичні вказівки можуть бути корисні при вивченні дисциплін «Теорія ймовірностей і математична статистика», «Алгоритми та структури даних».

Кожна робота завершується тестом.

Робота може бути зарахована, якщо студент виконав роботу та завдання для самостійної роботи і отримав по тесту не менше 30 балів. Звіт, завдання для самостійної роботи, та етап РГР разом оцінюються до 30 балів. Для оцінки вищої за 80 балів потрібно пройти співбесіду з викладачем.

1 ЛАБОРАТОРНА РОБОТА 1. КОРОТКІ ВІДОМОСТІ ПРО ТЕОРІЮ ІМОВІРНОСТЕЙ. ЗНАЙОМСТВО З КОМПОНЕНТАМИ, ЩО ВИКОРИСТОВУЮТЬСЯ ПРИ МОДЕЛЮВАННІ

Мета роботи:

– Знайомство з базовими поняттями теорії ймовірностей.

– Знайомство з компонентами, використовуваними при моделюванні.

1.1 Короткі теоретичні відомості

Випадкові величини можуть бути дискретними і безперервними.

У якості характеристик випадкових величин використовують або функцію щільності розподілу імовірностей, або інтегральну функцію розподілу імовірностей.

Зверніть увагу, випадкова величина характеризується не конкретним значенням, а функцією, що визначає імовірність появи якогось значення.

1.1.1 *Функція щільності розподілу ймовірностей для випадкової дискретної величини*

Випадкова **дискретна** величина приймає значення з деякого обмеженого набору. Наприклад, оцінка у системі ESTS (A, B, C, D, E, F), або у національній системі (5, 4, 3, 2).

Значення, які може приймати випадкова дискретна величина, з'являються з деякою частотою. Характеристикою частоти появи деякого значення випадкової дискретної величини є ймовірність появи цього значення. Імовірність - це межа відношення кількості випробувань, в яких відбулося деяка подія до загальної кількості випробувань. Звідси випливає, що сума ймовірностей для всіх можливих значень з набору, дорівнює 1. Наприклад, для результатів іспиту ймовірності появи оцінок можуть бути такими, як показано в таблиці 1.1.

Таблиця 1.1 – Імовірності оцінок на іспиті.

Оцінка	Ймовірність
2	0.1
3	0.4
4	0.3
5	0.2

Прийнято говорити, що така таблиця задає розподіл ймовірностей для випадкової дискретної величини.

Розподіл ймовірностей може бути представлено і графічно, так як це зроблено на рисунку 1.1.

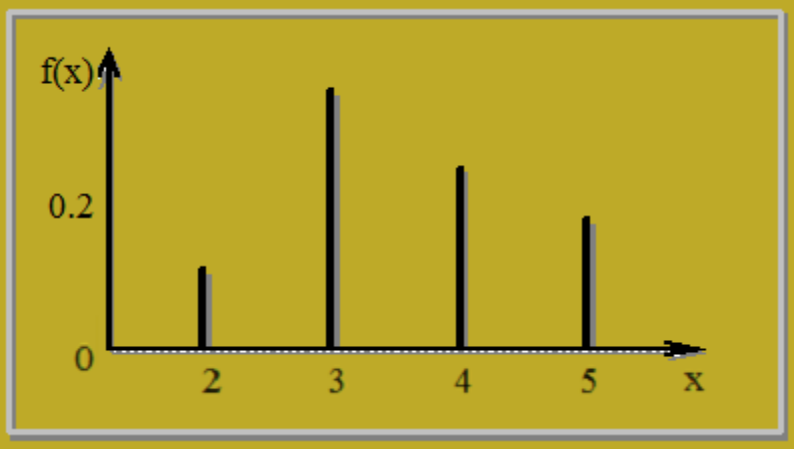

Рисунок 1.1 – Функція щільності розподілу ймовірностей для оцінок на іспиті

1.1.2 *Функція щільності розподілу ймовірностей для випадкової безперервної величини*

Випадкові безперервні величини можуть приймати будь-яке значення з деякого інтервалу, хоча межі цього інтервалу часто невідомі. У цьому випадку можна вважати, що значення випадкової неперервної величини теоретично може бути будь-яким. Наприклад, час перебування студента в аудиторії під час іспиту є випадковою безперервною величиною. Але оскільки випадкова величина може приймати будь-яке значення, то таких значень може бути нескінченно багато, а тому ймовірність кожного з них дорівнює нулю. Не нульовою може бути тільки ймовірність того, що випадкова величина потрапить у деякий інтервал. Наприклад, ймовірність того, що іспит для студента триватиме рівно 0.25 години, дорівнює нулю. Але ймовірність того, що студент буде на іспиті від 0.2 до 0.3 години, може бути вже й більшою, ніж нуль

З цієї причини, коли оцінюють імовірності появи випадкових безперервних величин, то визначають ймовірності попадання цієї випадкової величини в деякий інтервал. Очевидно, що ця ймовірність залежить як від властивостей самої випадкової величини, так і від ширини інтервалу. Ймовірність того, що числова випадкова величина потрапить в інтервал нескінченної ширини, дорівнює одиниці.

Якщо взяти інтервали однакової ширини, тоді ймовірності попадання випадкової величини в різні інтервали можна порівнювати, а в якості міри порівняння брати висоту прямокутника, основою якого буде інтервал, а висота дорівнюватиме імовірності появи випадкової величини у цьому інтервалі. Одержану таким чином фігуру називають гістограмою. Гістограма зводить випадкову безперервну величину до деякої умовної дискретної величини, значення якої інколи приймають рівним значенню середини інтервалу.

Як приклад, на рисунку 1.2 наведено гістограму для часу перебування студента на іспиті.

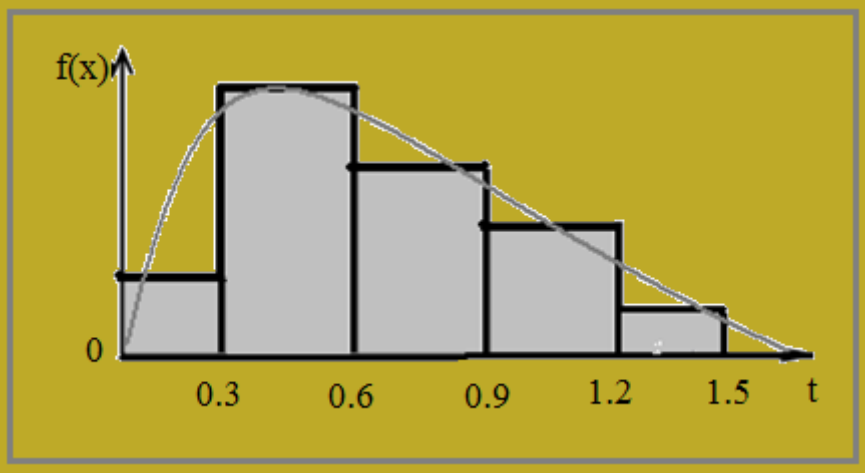

Рисунок 1.2 - Функція щільності розподілу ймовірностей для часу перебування на іспиті

Якщо ширину прямокутників гістограми зменшувати, а кількість їх пропорційно збільшувати, то лінія, що огинає вершини прямокутників буде поступово згладжуватися і, гранично, стане гладкою лінією, а для того, щоб висота одержуваної фігури не перетворилася на нуль, витримують площу фігури під кривою рівною одиниці. У цьому випадку ймовірність попадання випадкової величини в деякий інтервал буде дорівнює площі фігури утвореної межами інтервалу і огинаючої.

Отриману таким чином огинаючу лінію називають функцією щільності ймовірностей для випадкової безперервної величини.

Оскільки ймовірність випадкової величини не може бути негативною, то функція щільності ймовірностей не може мати негативних значень.

1.1.3 *Інтегральна функція розподілу ймовірностей для випадкової величини*

Поряд з функцією розподілу щільності ймовірностей часто використовується інтегральна функція розподілу ймовірностей, яка утворюється шляхом інтегрування функції щільності ймовірностей по значенню випадкової величини. Оскільки імовірності не можуть бути негативними, а сума ймовірностей дорівнює одиниці, то інтегральна функція - це функція яка не може зменшуватися, у якої мінімальне значення дорівнює 0, а максимальне значення дорівнює 1.

На рисунках 1.3 та 1.4 наведено інтегральні функції розподілу для наведених вище функцій щільності ймовірностей.

Значення інтегральної функції для деякого даного значення випадкової величини дорівнює ймовірності того, що випадкова величина прийме значення, що не перевищує дане значення.

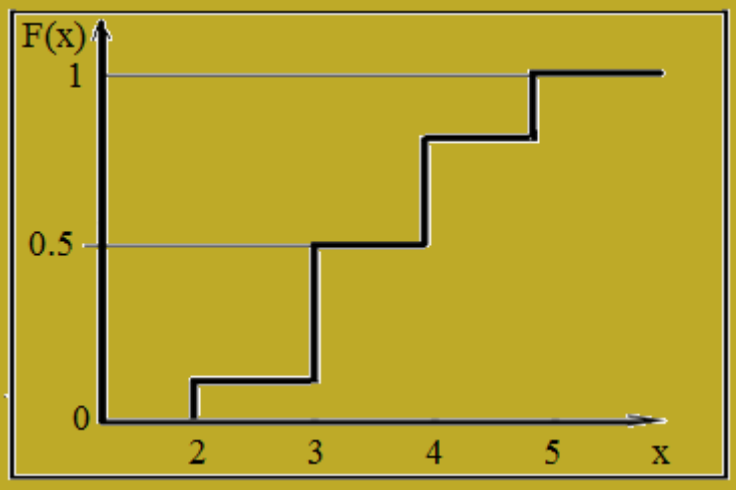

Рисунок 1.3 – Інтегральна функція розподілу ймовірностей для оцінок на іспиті

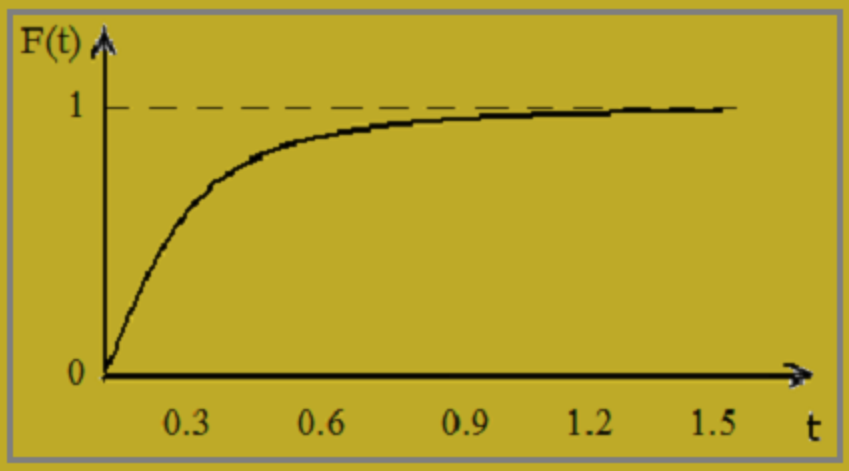

Рисунок 1.4 – Інтегральна функція розподілу ймовірностей для часу, витраченого на іспит

1.1.4 *Середнє значення та дисперсія випадкової величини*

Функції щільності ймовірностей або інтегральна функція розподілу ймовірностей дають найбільш повну характеристику випадкової величини. Але поряд з цими функціями часто використовують числові характеристики випадкової величини – середнє значення і дисперсію.

Для безперервної випадкової величини середнє значення дорівнює математичному очікуванню цієї випадкової величини, і визначається з функції щільності ймовірностей f (x) за формулою 1.1.

$$m = \int_{-\infty}^{+\infty} f(x) \cdot x \cdot dx \qquad (1.1)$$

Для дискретної випадкової величини середнє значення визначається за формулою 1.2

$$m = \sum_{i=1}^{n} x_i \cdot p_i \qquad (1.2)$$

де x – це значення випадкової величини, а p – імовірність цього значення.

За цією ж формулою можна визначити середнє значення і по гістограмі. У цьому випадку x – це значення випадкової величини у середині інтервалу, а p – імовірність того, що випадкова величина потрапить у цей інтервал.

Якщо ж ми маємо просто вибірку випадкових чисел, то середнє значення знаходиться як середнє арифметичне (1.3)

$$m = (\sum_{i=1}^{n} x_i) / n \qquad (1.3)$$

Дисперсію D для безперервної випадкової величини можна знайти за формулою 1.4.

$$D = \int_{-\infty}^{\infty} f(x) \cdot (x - m)^2 dx \qquad (1.4)$$

Для дискретної випадкової величини дисперсію можна знайти за формулою 1.5. Змінні у цій формулі ті ж самі, що у формулі 1.2.

$$D = \sum_{i=1}^{n} (x_i - m)^2 \cdot p_i \qquad (5)$$

Формулу 1.5 можна використовувати і для визначення дисперсії по гістограмі. У цьому випадку x – це середина інтервалу.

Якщо ж ми маємо просто вибірку випадкових чисел, то дисперсію можна знайти за формулою 1.6.

$$D = \sum_{i=1}^{n} (x_i - m)^2 / n \qquad (1.6)$$

Поряд з дисперсією випадкової величини часто використовують середнє квадратичне відхилення або стандартне відхилення (1.7).

$$\sigma = \sqrt{D} \qquad (1.7)$$

1.1.5 *Закони розподілу імовірностей випадкових величин*

1.1.5.1 **Рівномірний розподіл**

Рівномірний закон розподілу має дві характерні риси.

Перша полягає в тому, що випадкові числа можуть приймати значення тільки з деякого інтервалу, що визначається лівою та правою границями інтервалу.

Друга особливість полягає в тому, що щільність ймовірності для всього інтервалу однакова.

На рисунку 1.5 зображено функцію щільності ймовірностей та інтегральну функцію для випадкової величини, рівномірно розподіленої в діапазоні від ***a*** до ***b***.

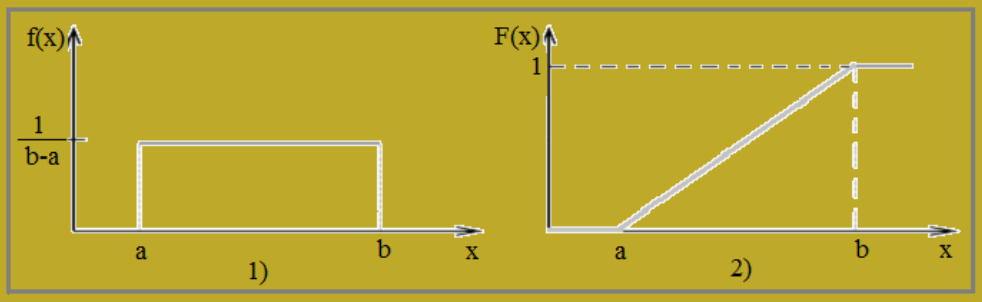

Рисунок 1.5 – Функція щільності ймовірності (1) та інтегральна функція (2) для рівномірного розподілу

Значення функції щільності ймовірності визначається виходячи з того факту, що площа фігури, що обмежується цією функцією повинна дорівнювати одиниці. Отже, значення щільності ймовірності на інтервалі від ***a*** до ***b*** буде ***1/(b-a)***.

Рівномірно розподілені випадкові числа використовуються для отримання випадкових чисел розподілу, що підпорядковуються іншим законам. При цьому найчастіше розглядаються числа, рівномірно розподілені на інтервалі від 0 до 1. Математичне очікування такої випадкової величини дорівнює 0.5, а дисперсія 1/12 або 0.083(3).

1.1.5.2 **Нормальний закон розподілення**

Нормальне розподілення є одним з найбільш поширених розподілень для випадкових величини. Такі випадкові величини зазвичай є результатом накопичення певної кількості випадкових впливів, що несуттєво впливають на загальний результат, наприклад, антропометричні дані людини.

Графічне зображення функції щільності імовірності та інтегральної функції для цього закону представлено на рисунку 1.6.

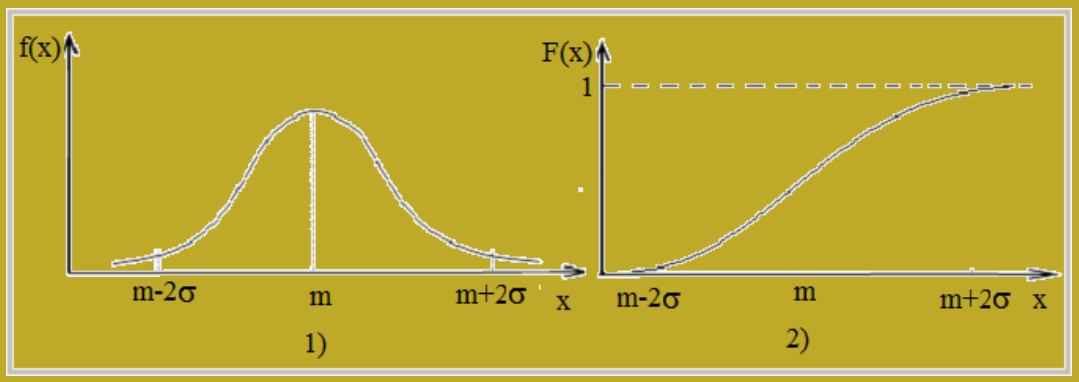

Рисунок 1.6 – Функція щільності ймовірностей (1) та інтегральна функція (2) для нормального розподілення

На рисунку 1.6 параметр розподілу m визначає середнє значення

випадкової величини, а параметр σ визначає середнє квадратичне відхилення.

1.1.5.3 **Експоненціальний закон розподілення**

Графіки функцій щільності ймовірностей та інтегральної функції для експоненціального закону розподілення представлений на рисунку 1.7.

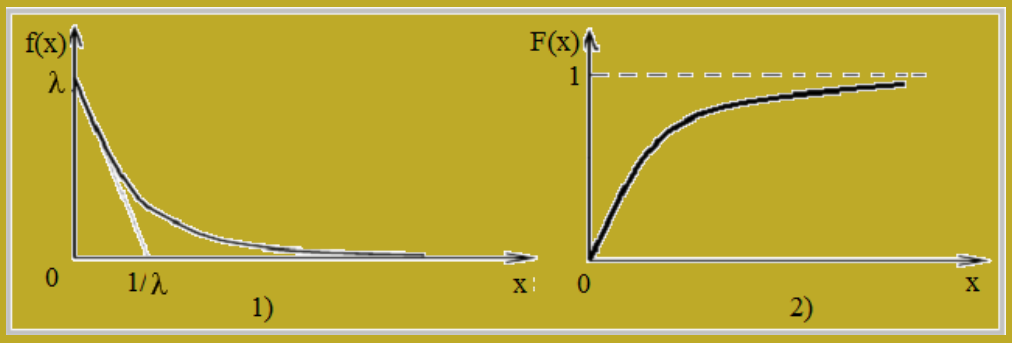

Рисунок 1.7 – Функція щільності ймовірностей (1) та інтегральна функція (2) для експоненціального розподілення

Цей закон розподілу часто використовується для визначення інтервалу між випадковими подіями, поява яких залежать від великої кількості незалежних факторів. Єдиний параметр цього розподілення λ, який зветься інтенсивністю, дорівнює оберненому середньому значенню випадкової величини.

1.1.5.4 **Закон розподілення Ерланга**

Розподілення Ерланга має два параметри: інтенсивність **λ** - це величина обернена середньому значенню випадкової величини, і коефіцієнт **k**, який називається коефіцієнтом Ерланга. Це розподілення займає проміжне місце між нормальним і експоненційним розподіленнями. При великих значеннях коефіцієнту **k** розподілення наближається до нормального розподілення, а при **k=1** співпадає з експоненціальним.

Графічне зображення функції розподілення щільності ймовірностей для закону Ерланга при різних значеннях k представлено на рисунку 1.8

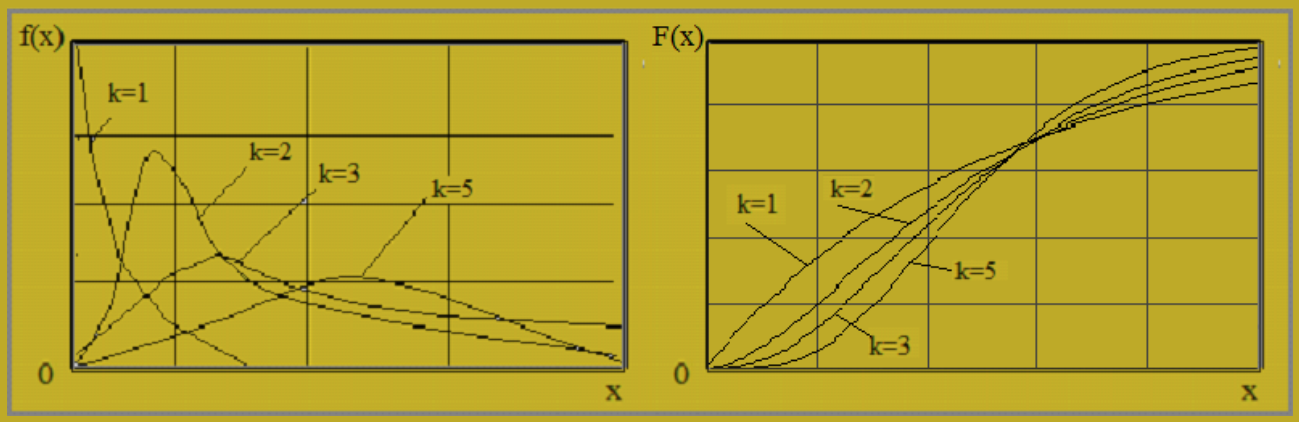

Рисунок 1.8 – Вигляд функції щільності ймовірностей та інтегральної функції для розподілення Ерланга

1.1.5.5 **Трикутний закон розподілення**

Цей закон розподілення може стати у нагоді, коли випадкові величини не можуть виходити за межі заданого діапазону, але у межах цього закону вони розподілені нерівномірно. Щільність вірогідності є найбільшою у деякій точці інтервалу і лінійно спадає до нуля на границях інтервалу.

Графічне зображення функції щільності імовірності для цього закону представлено на рисунку 1.9.

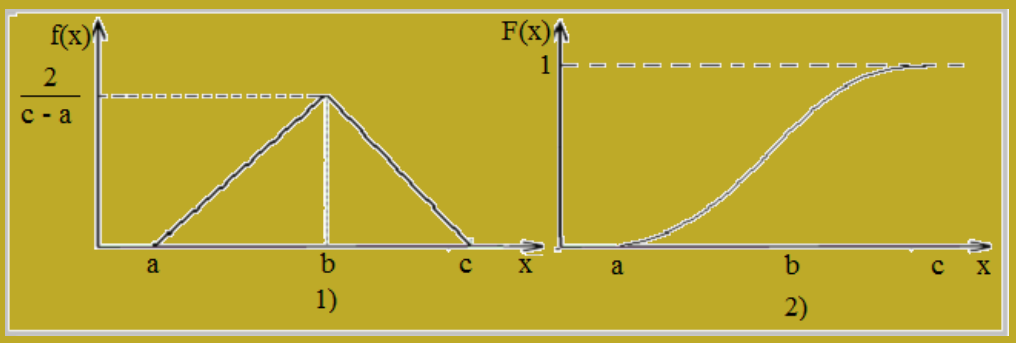

Рисунок 1.9 – Функція щільності ймовірностей (1) та інтегральна функція (2) для трикутного розподілення

1.1.5.6 **Довільні закони розподілення**

У випадках, коли статистичні дані не вписуються у межі якогось із відомих законів розподілу, функцію щільності імовірностей представляють у вигляді гістограми відносних частот. Інтегральна функція у цьому випадку буде виглядати, як сукупність відрізків прямих, рисунок 1.10, і може бути задана за допомогою двох масивів - x і F, що задають координати стиковки відрізків.

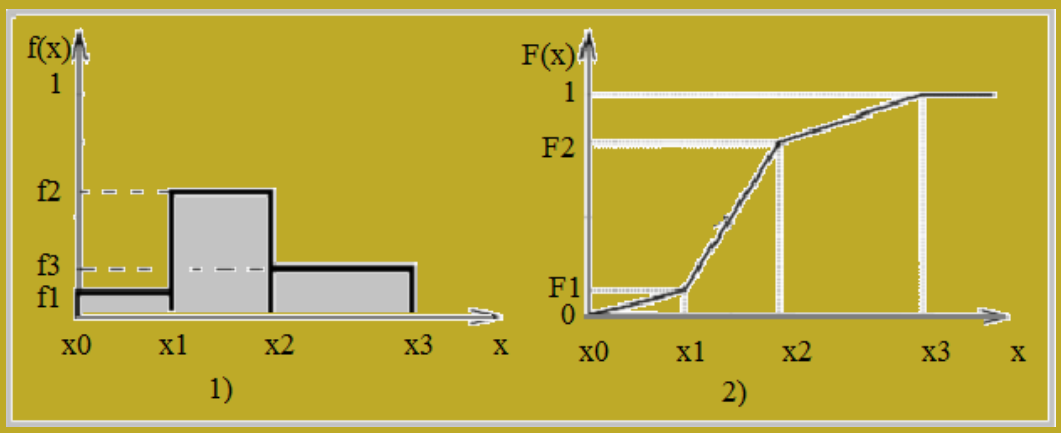

Рисунок 1.10 – Довільний розподіл.
1) гістограма відносних частот. 2) кускова лінійна апроксимація інтегральної функції розподілення

1.1.5.7 **Дискретне розподілення**

Дискретне розподілення звичайно задається масивом можливих значень випадкової величини і масивом ймовірностей цих значень. Наприклад, оцінки

на іспиті можуть приймати значення 2, 3, 4, 5, а імовірності їх появи 0.1, 0.4, 0.3, 0.2.

Графічне зображення такого розподілення представлено на рисунку 1.11.

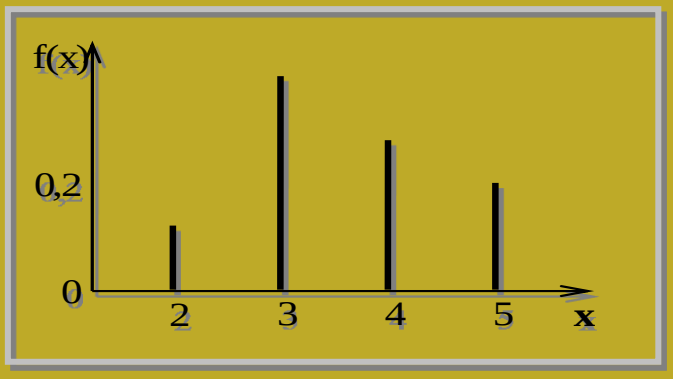

Рисунок 1.11 – Функція щільності розподілення ймовірностей для дискретного закону розподілення

1.2 Компоненти, що використовуються для моделювання

Лабораторні роботи та розрахунково-графічна робота з дисципліни виконується на основі фреймворку Simulation, створеного на кафедрі ІКС. Фреймворк Simulation містить пакети з класами, що дозволяють створювати моделі систем масового обслуговування та досліджувати їх. Тут ми розглянемо тільки деякі з компонентів фреймворку.

1.2.1 *Клас ChooseRandom*

Цей клас визначає візуальний компонент, що забезпечує вибір і налаштування потрібного генератора випадкових чисел. Елементом, які постійно знаходяться на формі є панель з кнопкою і полем для виведення інформації про вибраний законі розподілу. Вигляд цієї панелі представлений на рисунку 1.12.

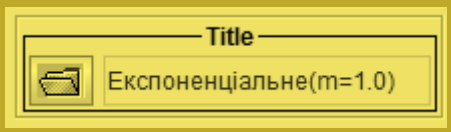

Рисунок 1.12 – Вигляд компонента ChooseRandom

При натисканні на кнопку відкривається комбобокс з переліком можливих розподілів, рисунку 1.13.

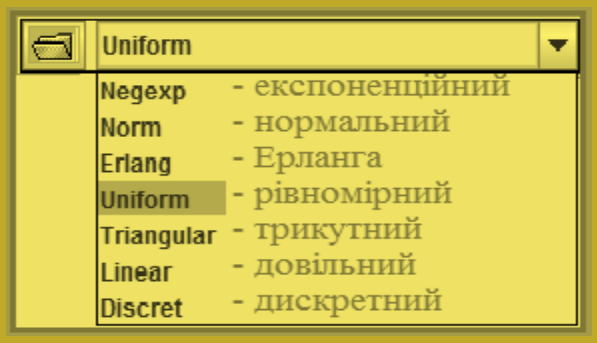

Рисунок 1.13 – Перелік можливих розподілів компоненту ChooseRandom

Після вибору необхідного розподілу з'являється діалогове вікно для налаштувань параметрів вибраного розподілу, рисунок 1.14.

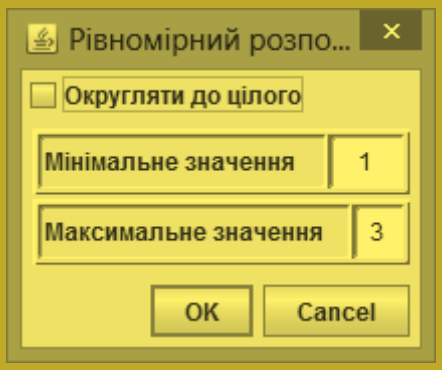

Рисунок 1.14 – Діалогове вікно для налаштування рівномірного розподілу компоненту ChooseRandom

Даний компонент реалізує інтерфейс Randomable. Основні методи цього інтерфейсу наступні:

– next() - повертає наступне випадкове число;

– probability(double) - повертає значення інтегральної функції розподілу.

1.2.2 *Клас ChooseData*

Об'єкти цього класу доцільно використовувати для введення числових параметрів компонент моделі. Цей клас успадковує клас JTextField. Поряд із усіма можливостями стандартного текстового компоненту клас ChooseData надає додаткові послуги. Вигляд компоненту представлено на рисунку 1.15.

Рисунок 1.15 – Вигляд компоненту класу ChooseData

Метод setTitle(String) дозволяє позначити призначення компоненту.

Методи getInt(), getFloat() та getDouble() дозволяють отримати числа відповідних типів.

Методи setInt(int), setFloat() та setDouble(double) дозволяють відображати числа відповідних типів.

Методи getIntArray() та getDoubleArray() дозволяють отримати масиви відповідних типів.

1.2.3 *Клас ChooseDataH*

Об’єкти цього класу також можна використовувати для введення числових параметрів компонент моделі. Вигляд компоненту представлено на рисунку 1.16.

Рисунок 1.16 – Вигляд компоненту класу ChooseDataH

За своїм функціоналом ці об’єкти майже такі, як і об’єкти попереднього класу. Так само з компоненту можна отримати дані типів int, double та масиви даних цих типів. Але цей клас успадковує клас JSplitPanel. Праву частину розщепленої панелі займає компонент JTextField і він доступний через метод getJTextField(). Початкове положення роздільника між лівою та правою частиною можна налаштовувати властивістю resizeWeight. За замовчуванням його значення дорівнює 0.75.

1.2.4 *Засоби для збирання та обробки статистичної інформації*

Класи Histo і DiscretHisto дозволяють накопичувати інформацію про значення випадкових безперервних і дискретних величин, і подавати інформацію про ці величини у вигляді гістограм. Гістограма може бути представлена у вигляді стовпчастої діаграми або у вигляді таблиці.

Для зручності деякі методи гістограм винесено у інтерфейс IHisto:

– init(). Цей метод без параметрів можна використовувати для ініціалізації об'єктів обох класів. У цьому випадку межі гістограми формуються автоматично.

– add(double). Цей метод додає у гістограму число, що передається до методу в якості параметру.

– addFrequencyForValue(double, double). В якості першого параметра задається вага переданого випадкового числа, а другим передається саме випадкове число. У простих випадках вага може дорівнювати 1. Саме так реалізовано метод add(double).

– showRelFrec(Diagram). Використовується для виведення результатів у вигляді стовпчастої діаграми. До методу як параметр передається посилання на об'єкт класу Diagram, де буде відображатися діаграма. Існує й інша модифікація методу, з більшим числом параметрів та більшими можливостями.

– getAverage(). Повертає середнє значення для накопичених даних.

– toString(). Використовується для виведення результатів обробки

накопичених даних у вигляді тексту з таблицею відносних частот.

Нижче наведені ще деякі методи, що не увійшли до єдиного інтерфейсу.

Для ініціалізації, додатково можна використовувати такі методи:

– initFromTo(int, int). Цей метод використовують для об'єктів класу DiscretHisto. Два цілих числа, що передаються в якості параметрів, визначають межі гістограми.

– initFromTo(double, double, int). Цей метод використовують для об'єктів класу Histo. Два дійсних числа, що першими передаються в якості параметрів, визначають межі гістограми, а ціле число визначає кількість інтервалів.

1.2.5 *Класи для графічного відображення результатів моделювання*

Для графічного відображення результатів моделювання використовуються класи widgets.Diagram і widgets.Painter.

Клас Diagram реалізує візуальний компонент для відображення графіків і діаграм. Вид компоненту класу Diagram показано на рисунку 1.17.

Компонент дозволяє настроювати мінімальні і максимальні значення по осях координат, заголовок діаграми. Можна настроювати координатну сітку. Розміри компоненту можна змінювати.

Нижче наведено деякі методи з класу Diagram.

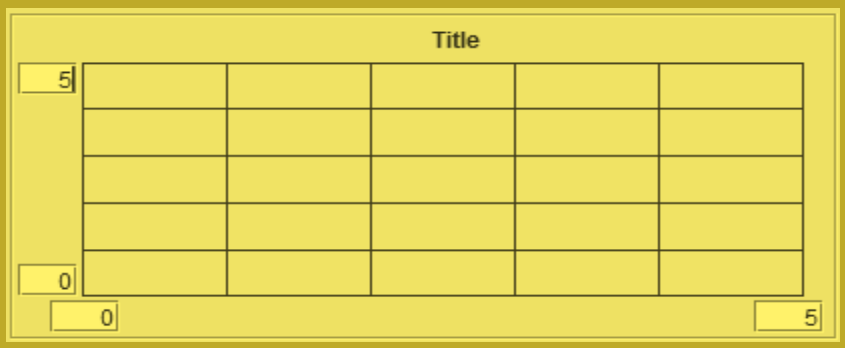

Рисунок 1.17 – Вигляд компоненту класу Diagram

Методи setGridByX(int), setGridByY(int) використовуються для налаштування масштабної сітки на діаграмі.

Методи setHorizontalMinText(String), setHorizontalMaxText(String), setVerticalMinText(String), setVerticalMaxText(String) використовуються для налаштування діапазону діаграми по горизонтальній та вертикальній висях.

Метод setTitle(String) використовуються для налаштування заголовку діаграми.

Метод clear() очищає діаграму.

Метод setPainterColor(Color) налаштовує колір зображень.

Зображення геометричних фігур на діаграмі виконуються об'єктом класу Painter, що входить до складу об'єктів класу Diagram. З його допомогою можна зображувати лінії, прямокутники, овали. Колір ліній налаштовується. Особливість об'єкту Painter полягає в тому, що після зображення лінії, він пам'ятає своє становище на діаграмі, що спрощує зображення графіків

Об'єкти класу Painter можуть існувати і незалежно від діаграми, а мати тільки посилання на неї. Завдяки цьому, кілька об'єктів класу Painter можуть використовувати одну й ту ж діаграму, що дозволяє зображати на одній діаграмі кілька графіків.

Метод placeToXY(float, float) використовуються для переміщення пера у точку із заданими відносно діаграми реальними координатами.

Метод drawToXY(float, float) використовуються для проведення лінії у задану точку на діаграмі.

Метод drawDependency(…) можна використовувати для виведення графіків. Як параметри в останній метод передається два масиви, які задають координати точок графіка, колір графіка і параметр логічного типу, який вказує, чи слід налаштовувати діаграму під дані графіка.

Метод drawBarsDiagram(…) цього класу можна використовувати для виведення стовпчастих діаграм. Як параметри в метод передається масив границь інтервалів по горизонтальній висі, масив значень по вертикалі, ширина стовпчика в долях ширини інтервалу, зсув стовпчика від лівої межі проміжку і параметр логічного типу, який вказує, чи потрібна зміна налаштування діаграми під дані.

Метод drawNeedleDiagram(…) цього класу можна використовувати для відображення голчастої діаграми. Перелік параметрів цього методу майже такий самий як і у попереднього.

Перелічені вище методи можна викликати і через об’єкти класу Diagram.

На діаграмах також можна відображати гістограми. Але ці методи викликаються для об’єктів-гістограм, а посилання на діаграму у ці методи передається як параметр.

1.3 Завдання на виконання роботи

1.3.1 *Підготовка до роботи*

– Активізуйте файл SimulationAllLab2018.jar. Версії цих файлів змінюються, тому назва файлу може бути і іншою. Можна також завантажити до Eclipse файл проекту із архіву SimulationAllLab2018.zip і активізувати стартовий файл застосування mainUI, що знаходиться у папці main проекту. У цьому проекті до кожної лабораторної роботи є папка, Назва якої починається словами «toLab..».

1.3.2 *Робота із програмою «Знайомство з випадковими величинами»*

– Налаштуйте компонент ChooseRandom на рівномірний закон розподілу з довільними значеннями параметрів.

– Отримайте гістограму для вибірки випадкових чисел та інтегральну функцію для заданих параметрів розподілу.

– Змініть значення параметрів розподілу та знов зафіксуйте гістограму та інтегральну функцію. **Не забувайте налаштовувати параметри відображення відповідно до параметрів розподілу. На графіку мають бути**

усі значення інтегральної функції, від 0 до 1.

– Виконайте такі самі дії для нормального, експоненціального, трикутного законів та закону Ерланга.

– Налаштуйте компонент ChooseRandom на дискретний закон розподілу, що визначає розподіл оцінок на іспиті, та отримайте гістограму для вибірки дискретних випадкових чисел і інтегральну функцію.

– Налаштуйте компонент ChooseRandom на довільний закон розподілу, функція щільності імовірностей якого має вигляд двох горбів різної висоти. Не забувайте, що загальна площа горбів має дорівнювати одиниці. А у якості налаштувань генератора передаються значення інтегральної функції, перше з яких дорівнює нулю, а останнє одиниці.

1.4 Тестування

Запустіть програму тестування і потренуйтесь у відповідях на запитання.

1.5 Завдання для виконання етапу РГР

Отримати у викладача варіант РГР. Ознайомитися із описом системи, що підлягає моделювання. Визначити перелік параметрів моделі, що підлягає налаштуванню. Створити візуальний клас (див. зразок розрахунково-графічної роботи), і у цьому класі створити панель налаштування параметрів моделі.

1.1 Завдання для самостійної роботи

Використовуючи компоненти фреймворку Simulation створіть застосування для тестування методу random() класу Math. Застосування має надавати можливість створювати та переглядати гістограму для вибірки чисел, заданого розміру як у вигляді стовпчастої діаграми, так і у вигляді тексту. За допомогою функції test, що є у контекстному меню гістограми можна перевіряти гіпотезу про гаданий розподіл.

1.1.1 *Рекомендації до виконання завдання*

До проекту треба підключити файл Simulation2018.jar.

Для доступу до компонентів фреймворку використовуйте пункт ChooseBean меню панелі компонентів.

Для введення обсягу вибірки використовуйте об’єкт класу ChooseData або ChooseDataH.

Для накопичення даних використовуйте метод add(double) компонент класу Histo, але його слід розташувати за межами форми. Перед накопиченням даних компонент слід ініціалізувати.

Для виведення текстової інформації використовуйте стандартний компонент JTextArea і метод toString() об’єкту класу Histo.

Для візуального відображення діаграми використовуйте компонент класу Diagram і метод showRelFrequency(Diagram) класу Histo.

Прототип застосування можна переглянути за допомогою пункту меню «Що треба створити самостійно» до лабораторної роботи.

1.2 Звіт про виконання роботі

У звіті мають бути наведені результати виконання роботи у вигляді:

- копій екранів для пункту «Знайомство з випадковими величинами»
- код проекту для завдання на самостійну роботу
- зображення панелі налаштувань моделі, що має бути створена у межах РГР.

2 ЛАБОРАТОРНА РОБОТА №2.
ГЕНЕРАТОРИ ВИПАДКОВИХ ВЕЛИЧИН

Мета роботи:

– Знайомство зі способами одержання рівномірно розподілених випадкових чисел (РРВЧ).

– Знайомство з методами тестування генераторів РРВЧ.

– Дослідження впливу налаштувань генератора РРВЧ на характеристики псевдо випадкової послідовності що генерується.

2.1 Короткі теоретичні відомості

Випадкові величини, які ми маємо моделювати, можуть бути різними. Наприклад, час чекання перед світлофором – це число, а колір сигналу світлофора – це зелений, жовтий або червоний колір. Але у моделі будь-які величини зазвичай приводяться до числових значень. Тобто значення кольорів у наведеному прикладі будуть моделюватися числами, наприклад, 1 - зелений, 2 - жовтий, 3 - червоний.

Таким чином, під час моделювання виникає потреба генерувати випадкові числа. Для цього використовуються так звані генератори випадкових чисел. Зазвичай, це об’єкти, які у відповідь на виклик таких методів як next(), nextDouble(), nextInt(), та подібних до них, повертають число відповідного типу та потрібного розподілу.

2.1.1 *Генератори рівномірно розподілених чисел*

Рівномірно розподілені випадкові числа (РРВЧ) - це випадкові числа, які підпорядковуються рівномірному закону розподілу.

2.1.1.1 Тестування генераторів РРВЧ

Ефективність статистичного моделювання багато в чому залежить від якості випадкових чисел, ось чому генератори випадкових чисел повинні пройти тестування. Найважливіші характеристики послідовностей випадкових чисел, які формують генератори, це:

– період,

– рівномірність,

– стохастичність,

– незалежність.

Перевірка на період полягає у визначенні довжини послідовності, де числа послідовності не повторюються.

Перевірка на рівномірність визначає, чи відповідають числа отриманої послідовності рівномірному розподілу.

Перевірка на стохастичність визначає, наскільки якісно перемішано числа послідовності у межах періоду.

Перевірка на незалежність має підтвердити відсутність взаємозв’язків (кореляції) між числами послідовності.

Окрім перелічених класичних тестів, які будуть розглянуті у лабораторній роботі, існує багато інших способів тестування таких генераторів [1,2]. На сьогодні одним із найбільш суворих наборів тестів для оцінки якості набору випадкових чисел вважаються тести «diehard», розроблені Джорджем Марсалі. Недаремно назва цього набору з 12 тестів перекладається фразеологізмом «міцний горішок

Визначення періоду випадкових чисел

Ця перевірка полягає у визначенні довжини послідовності, де числа не повторюються. У зв'язку з тим, що при деяких значеннях констант генератора початкові числа послідовності можуть приймати унікальні значення, які не з'являються пізніше, то при визначенні періоду зазвичай пропускають початкову ділянку, а потім фіксують значення якого-небудь числа (а інколи і декількох) й підраховують кількість чисел у послідовності доти, доки чергове число (або числа) не будуть співпадати із зафіксованим числом.

Перевірка на рівномірність

При перевірці на рівномірність перевіряється гіпотеза про рівномірний розподіл псевдо випадкових чисел. Оцінити рівномірність розподілу можна за допомогою гістограми. Для цього діапазон можливих значень випадкових чисел, наприклад (0,1) розбивається на N рівних частин. Число інтервалів за звичай вибирають пропорційно логарифму від обсягу вибірки. Після цього підраховують скільки разів випадкові числа потрапляли до кожного з інтервалів. Очевидно, що якщо випадкова величина є рівномірно розподіленою, то при великій кількості членів послідовності, значення частот у кожному інтервалі повинні наближатися до величини V/N, де V - число членів послідовності (обсяг вибірки), а N - число інтервалів. Але у зв’язку з тим, що обсяг вибірки не буває нескінченним, завжди існують якісь відхилення дійсних (емпіричних) частот від теоретичних частот.

Розглянемо, наприклад, випадкові числа, що наведені на рисунку 2.1.

10 09 73 25 33	76 52 01 35 86	34 67 35 48 76	80 95 90 91 17
37 54 20 48 05	64 89 47 42 96	24 80 52 40 37	20 63 61 04 02
08 42 26 89 53	19 64 50 93 03	23 20 90 25 60	15 95 33 47 64
99 01 90 25 29	09 37 67 07 15	38 31 13 11 65	88 67 67 43 97
12 80 79 99 70	80 15 73 61 47	64 03 23 66 53	98 95 11 68 77

Рисунок 2.1 – Фрагмент таблиці випадкових чисел

З кожної пари цифр можна сформувати дробову частину числа меншого одиниці. Таким чином отримаємо 100 чисел у діапазоні від 0 до 1.

Розіб'ємо весь діапазон на 5 рівних частин і підрахуємо, скільки чисел потрапляє в кожний інтервал. Результати представимо у вигляді таблиці 2.1.

Таблиця 2.1 – Абсолютні емпіричні частоти для інтервалів

Границі інтервалів	0 .. 0.199	0.2 .. 0.399	0.4 .. 0.599	0.6 .. 0.799	0.8 .. 1
Абсолютні частоти	22	19	15	20	24

Теоретична абсолютна частота при рівномірному розподілі однакова для всіх інтервалів (звичайно, якщо інтервали однакової ширини) і визначається як частка від ділення обсягу вибірки на кількість інтервалів. У нашому випадку обсяг вибірки дорівнює 100, а інтервалів 5. То виходить, що теоретична частота дорівнює 20.

Аналізуючи таблицю 2.1 бачимо, що майже в кожному інтервалі спостерігаються відхилення абсолютних частот від теоретичного значення. Виникає питання, чи є ці відхилення випадковими, чи дійсно числа розподілені не рівномірно?

Для відповіді на це питання можна скористатися статистичними критеріями перевірки гіпотез про відповідність емпіричних розподілів теоретичним розподілам.

Критерій відповідності Пірсона ("Хі-квадрат").

У тих випадках, коли обсяг вибірки не менше 100, для перевірки гіпотези про відповідність емпіричного та теоретичного законів розподілу можна скористатися критерієм відповідності Пірсона ("хі-квадрат").

У цьому критерії розбіжність теоретичних і емпіричних абсолютних частот, отриманих для деякої вибірки, оцінюється величиною, що визначається за формулою 2.1.

$$\chi^2 = \sum_{i=1}^{n} \frac{(w_i^{Э} - w_i^{T})^2}{w_i^{T}} \qquad (2.1)$$

де $w^{Э}$, w^{T} - емпіричні і теоретичні абсолютні частоти для інтервалів. Інтервали, для яких значення абсолютної частоти менше 5, поєднують із сусіднім інтервалом

Отримане за формулою 2.1 значення "хі-квадрат" порівнюють із критичним значенням.

"Критичне" значення "Хі-квадрат" визначається із статистичних таблиць відповідно до числа степенів вільності й прийнятим рівнем значимості "альфа".

Число степенів вільності визначається за формулою S-*R*, де *S* - число інтервалів, а *R* - число параметрів розподілу, визначених по вибірці.

Рівень значимості (ймовірність того що, відкидаючи висунуту гіпотезу, ми зробимо помилку) звичайно приймають рівним 0.05.

Якщо отримане значення "Хі-квадрат" менше "критичного", гіпотеза приймається.

Нижче, у таблиці 2.2 наводяться деякі критичні значення "Хі-квадрат" для рівня значимості "альфа"=0.05.

Таблиця 2.2 – Критичні точки розподілу "Хі-квадрат"

Число ступенів вільності	Критичне значення
3	7.8
5	11.1
7	14.1
10	18.3

Критичні значення "Хі-квадрат" для інших значень числа степенів вільності й рівнів значимості можна знайти в довідковій літературі [3].

Для прикладу, що розглядався у підпункті 2.1.4.3, значення «хі-квадрат», знайдене по формулі 2.4, буде наступним.

$$\chi^2 = \frac{(22-20)^2}{20} + \frac{(19-20)^2}{20} + \frac{(15-20)^2}{20} + \frac{(20-20)^2}{20} + \frac{(24-20)^2}{20} = 2.3$$

Число степенів вільності в нашім випадку дорівнює числу інтервалів, тобто п'яти. Для цього числа знаходимо критичне значення з таблиці 2.2, воно дорівнює 11.1.

Отримане (спостережуване) значення «хі-квадрат», що дорівнює 2.3, менше критичного, котре дорівнює 11.1, отже гіпотезу про рівномірний розподіл випадкових чисел можна прийняти.

Перевірка стохастичності

Стохастичність (випадковість) отримуваної послідовності рівномірно розподілених чисел можна оцінювати різними методами.

Один з них називається методом серій. У цьому методі випадкова послідовність чисел *X* розбивається на елементи двох видів (наприклад + і -) відповідно до умови 2.2.

$$E = \begin{cases} +, \text{ якщо } X < r \\ -, \text{ якщо } X \geq r \end{cases} \qquad (2.2)$$

де *r* довільно задається з інтервалу існування змінної *X*.

Після такої заміни послідовність отримує, наприклад,

такий вид: + + + - - + + - - - + - + - + +....

Ділянки цієї послідовності, що складаються з елементів «+», називають серіями. Елемент «-» виступає у ролі межі серії. Якщо два елементи «-» розташовані поруч, то довжина серії між ними дорівнює 0.

У наведеному прикладі маємо серії такої довжини: 3,0,2,0,0,1,1,2...

Таким чином, ми отримуємо нову послідовність випадкових цілих чисел.

Визначимо теоретичні значення імовірності появи кожного з цих чисел, що визначають довжину серії.

Для випадкових чисел, рівномірно розподілених на інтервалі 0..1, ймовірність події «*+*» дорівнює *r*, а ймовірність події *«-»* дорівнює 1-*r*.

Таким чином, ймовірність серії довжиною 0 дорівнює ймовірності появи події «-» , тобто $P_0 = 1-r$.

Ймовірність серії довжиною 1 дорівнює ймовірності появи послідовності

подій «+ -», тобто $P_1=r\ (1-r)$.

Ймовірність серії довжиною 2 дорівнює ймовірності появи послідовності подій «+ + -», тобто $P_2=r^2\ (1-r)$.

Ймовірність появи серії довжиною n буде дорівнювати

$$P_n = r^n \cdot (1-r) \tag{2.3}$$

Для перевірки стохастичності генератора необхідно сформувати послідовність випадкових чисел, задатися деяким r, виділити в цій послідовності серії різної довжини, підрахувати їхню кількість і знайти відносні частоти їхньої появи, а потім порівняти їх з теоретичними значеннями.

Звичайно частоти, що знайдені експериментально, не співпадають з теоретичними ймовірностями внаслідок обмеженості обсягу вибірки. Тому необхідно перевірити, чи випадкові отримані відхилення чи ні.

Для цього можна теж скористатися критерієм Пірсона.

При перевірці варто мати на увазі, що довжина серії - величина дискретна, тому при обробці даних можна не використовувати поняття інтервалу, а визначати накопичені значення відносних частот для кожної з отриманих довжин серій

Перевірка незалежності

Перевірка незалежності проводиться шляхом обчислення кореляційного моменту (коефіцієнту кореляції) між випадковими величинами даної послідовності, і деякої іншої. Інша послідовність може бути отримана з даної послідовності шляхом її зміщення. Зміщення послідовності можна зробити, якщо відкинути в ній декілька перших елементів. Наприклад, якщо вихідна послідовність складається із чисел 3 8 5 0 2 5 7 1 3 8 1 ..., то, відкинувши перші 3 елементи, отримаємо послідовність із чисел: 0 2 5 7 1 3 8 1..., яку можна порівнювати з вихідною.

Можна перевіряти на незалежність і послідовності, отримувані за допомогою різних генераторів.

Величина кореляційного моменту k визначається за формулою 2.4,

$$k = \frac{\overline{x \cdot y} - \overline{x} \cdot \overline{y}}{\sigma_x \cdot \sigma_y} \tag{2.4}$$

де $\overline{xy}$ - середнє значення добутку відповідних елементів двох послідовностей;

$\overline{x} \cdot \overline{y}$ - добуток середніх значень елементів двох послідовностей;

σ_x, σ_y - середньоквадратичні відхилення випадкових величин.

Кореляційний момент двох незалежних випадкових величин при досить великій довжині вибірки буде прямувати до нуля. Однак внаслідок того, що обсяг вибірки обмежений, значення кореляційного моменту зазвичай відрізняється від нуля. Для того щоб визначити, чи випадкове це відхилення, знаходять значення випадкової допоміжної величини T за формулою 2.5.

$$T = \sqrt{\frac{k^2}{1-k^2}(n-2)} \qquad (2.5)$$

де n- обсяг вибірки,

k - кореляційний момент.

Відомо, що випадкова величина Т повинна задовольняти розподілу Стьюдента з n-2 ступенями вільності. У статистичних таблицях [3] можна знайти критичні значення для величини Т. Фрагмент такої таблиці наведений нижче, таблиця 2.3. Якщо отримане значення критерію Т не перевищує критичного значення, то можна вважати, що випадкові величини незалежні.

Таблиця 2.3 – Критичні точки розподілу Стьюдента для рівня значимості 0.05

Число ступенів волі	Критичне значення
5	2.01
10	1.81
20	1.73
40	1.68

Варто мати на увазі, що при використанні декількох генераторів, залежність між ними може виникнути не через погані налаштування, а внаслідок того, що вони будуть ініціалізовані у межах однієї мілісекунди. У цьому випадку вони будуть формувати однакові послідовності випадкових чисел, і коефіцієнт кореляції вийде рівним одиниці. Щоб уникнути цього треба перед ініціалізацією кожного генератора робити невелику затримку.

2.1.1.2 **Способи отримання РРВЧ**

Існують наступні способи одержання РРВЧ – фізичний, табличний і алгоритмічний.

Фізичний метод заснований на перетворенні у число випадкових вихідних сигналів, що виникають під час роботи якогось пристрою, найчастіше електронного або електромеханічного. Отримані в такий спосіб числа дійсно є випадковими, однак такий спосіб досить складний і в наш час використовується не часто.

Табличний метод заснований на використанні спеціальних таблиць, з яких вибирається послідовність РРВЧ.

Ми вже бачили (рисунок 2.2) фрагмент таблиці випадкових чисел.

Числа із цієї таблиці можна вибирати у будь-якому порядку. Наприклад, якщо потрібні цілі числа не більші 10, то можна брати цифри підряд по горизонталі 1, 0, 0, 9, 7, 3, ..., або вертикалі 1, 3, 0, 9, 1, ...

Якщо потрібні дробові числа, що не перевищують 1, з точністю до чотирьох знаків, то отримаємо 0.1009, 0.7325, 0.3376, 0.5201, ...

Таблиці зручні при ручній роботі з випадковими числами, однак при роботі з комп'ютером, цей спосіб незручний.

Алгоритмічні методи засновані на застосуванні різних обчислювальних процедур. Ці процедури звичайно використовують рекурентні формули, що дозволяють отримувати послідовності випадкових чисел, у яких кожне наступне число визначається попереднім, або декількома попередніми. Початкове число послідовності, іноді називається «зерном». Його потрібно або довільно задати, або воно автоматично визначається по лічильнику мілісекунд системного таймера. Отримані числа називаються псевдовипадковими тому, що з одного боку послідовності таких чисел задовольняють вимогам тестів, а з іншого боку залишаються повністю відтворюваними, тому що при тому самому «зерні» формується та сама послідовність випадкових чисел.

Метод серединних квадратів

Цей метод, розроблений фон-Нейманом, використовувався на перших комп'ютерах, у середині минулого століття. Суть його полягала в наступному. Ціле число підносили до квадрата і якщо була потреба, доповнювали ліворуч нулями до восьми знаків. Як наступне число брали число, представлене середніми чотирма цифрами отриманого. Наприклад, початкове число 1234. Після піднесення його у квадрат одержуємо 01522756. Середні чотири цифри дають число 5227. Підносячи його у квадрат, одержуємо 27321529. Отже, наступним випадковим числом буде 3215, і так далі.

Цей метод виявився недостатньо ефективним і за якістю отримуваних чисел і за швидкістю.

Адитивно-мультиплікативний конгруентний метод

Адитивно-мультиплікативний конгруентний метод (АМКМ) є досить поширеним методом отримання рівномірно розподілених випадкових чисел, хоча і не проходить тести «diehard». Саме цей метод використовується у класі java.util.Random. Тому розглянемо його тут докладніше.

Цей метод заснований на застосуванні рекурентного співвідношення 2.3 для формування псевдо випадкової послідовності цілих чисел.

$$U_{i+1} = (A + B \cdot U_i) \bmod M, \qquad (2.3)$$

де U_i, U_{i+1} – попереднє і наступне значення послідовності;

M - модуль;

A - адитивна константа;

B – мультиплікативна константа;

Кожне випадкове число, що отримано за цією формулою, являє собою цілочислову остачу від ділення виразу ($A + B*U$) на модуль M.

Як приклад, отримаємо послідовність випадкових чисел для модуля 10, прийнявши A=9 і B=7.

Нехай зерно дорівнює 0, тобто $U_0=0$;

Тоді $U_1 = (9+7*0) \bmod 10 = 9$;

Зверніть увагу, якщо попереднє число дорівнює 0, то наступне дорівнює адитивній константі.

$U_2 = (9+7*9) \bmod 10 = 2$;

$U_3 = (9+7*2) \bmod 10 = 3;$
$U_4 = (9+7*3) \bmod 10 = 0.$

Як бачимо, числа почали повторюватися. Раніше чи пізніше, але це завжди відбувається. Послідовність неповторюваних цілих чисел не може бути довша, ніж значення модуля.

Якість одержуваної послідовності випадкових чисел залежить від вибору констант *A* і *B*, та модуля.

Вибір модуля впливає на довжину неповторюваної послідовності, що називається періодом. Чим більший модуль, тим більшим може бути період.

Крім того, від вибору модуля залежить швидкість роботи генератора. Якщо вибрати модуль рівний степеню двійки, то операція mod може бути виконана дуже швидко за рахунок використання побітової операції логічного «і». Якщо модуль відповідає розрядності шини даних комп'ютера, то тоді результат обчислення цілочислового виразу (*A+B*U*) буде автоматично обрізатися, і операцію mod взагалі не потрібно буде застосовувати. Саме такі значення модуля найчастіше використовуються, коли продуктивність комп'ютера недостатньо висока.

Якість отримуваної послідовності багато в чому залежить від вибору мультиплікативної константи. Вона повинна бути достатньо велика, інакше на деяких ділянках послідовності числа будуть підпорядковуватися лінійному закону й, отже, не будуть випадковими. Однак значення близькі до модуля теж не можна вважати гарними. Мультиплікативна константа не повинна мати спільних дільників з модулем, щоб уникнути різкого скорочення довжини періоду.

Питання вибору значення мультиплікативної константи добре вивчене для значень модуля, що дорівнюють степеню два. Наприклад, в [2] для вибору значення мультиплікативної константи пропонується формула $B=8p\pm3$, а в [4] $B=2^p+1$, де ***p*** - довільне ціле додатне число.

Адитивна константа - це непарне ціле число більше 0. Вона оберігає генератор від зриву, коли чергове U отримує значення 0.

Метод "Mother-of-All"

Метод запропоновано відомим науковцем з проблем генераторів випадкових чисел Джорджем Марсалією.

Цей метод розширює попередній за рахунок використання попередніх членів послідовності, і відповідної кількості адитивних констант. За рахунок цього метод забезпечує дуже великий період і проходить тести «diehard», але він має невелику швидкість, бо використовує багато операцій множення. Алгоритм може бути застосований в прикладних науках, там де не потрібна велика швидкість.

Вихор Мерсена

Метод "Mersenne twister" розроблено в 1997 році японськими науковцями Мацумото і Нішімура. Переваги методу - величезний період (2 ^ 19937-1) і швидка генерація випадкових чисел, що потрібно для задач криптографії. Але

метод не забезпечує достатню випадковість послідовності чисел. Тому галузь застосування алгоритму дещо обмежена. Незважаючи на це вихор Мерсена реалізований у стандартних бібліотеках для PHP, Python і Рубі.

Генератори типу "Xorshift"

Генератори цього типу запропоновані Джорджем Марсалією і використовують ідею методу "Mother-of-All", але замість операцій множення використовуються тільки побітові операції xor та побітові зсуви. Це один з найбільш вживаних на даний час алгоритмів. Послідовність, що генерується, достатньо випадкова, має період 2^128 − 1 та проходить тести Diehard.

Нижче наведено реалізацію даного алгоритму на C:

```
unsigned long xor128() {
        static unsigned long
        x=123456789, y=362436069,
        z=521288629, w=88675123;
        unsigned long t;
        t=(x^(x<<11));
        x=y; y=z; z=w;
        w=(w^(w>>19))^(t^(t>>8));
        return w;
}
```

2.1.1.3 Приведення отримуваних чисел до інтервалу 0..1

Всі розглянуті вище алгоритмічні методи базуються на роботі із цілими числами. Для того щоб звести ці числа до інтервалу 0..1 слід розділити отримувані числа на величину модуля. Якщо модуль досить великий, то дискретністю отримуваних чисел можна знехтувати й вважати, що ми маємо справу з безперервною випадковою величиною.

2.1.1.4 Реалізація генератора РРВЧ в Java

В Java для одержання РРВЧ використовується описаний вище адитивно-мультиплікативний метод, який реалізований у класі java.util.Random. Цей клас, крім реалізації методів, що забезпечують отримання рівномірно розподілених чисел, дозволяє отримувати випадкові числа, що підпорядковуються розподілу Гауса (нормальний розподіл). Тому в деяких методах виконуються операції, що стосуються обох розподілів.

Для створення генератора можна скористуватися пустим конструктором. У цьому випадку в якості «зерна» використовується поточне значення мілісекунд системного таймеру. Але якщо таким чином створювати декілька генераторів, то існує велика імовірність того, що вони будуть генерувати однакові послідовності, навіть коли моменти створення відділені один від одного великою кількістю коду. Адже швидкодія сучасних комп’ютерів дуже велика і одна мілісекунда – це дуже великий проміжок часу.

Можна скористатися і конструктором з параметром цілого типу. Значення цього параметру буде однозначно перетворено у зерно.

Для отримання цілих позитивних чисел за заданим модулем використовується метод next(int). Він також є базовим для отримання рівномірно розподілених чисел різних типів.

Метод nextLong повертає випадкове число типу long.

Метод nextInt() повертає випадкове число типу int.

Метод nextFloat() повертає випадкові числа типу float.

Метод nextDouble() повертає випадкові числа типу double.

Метод nextBoolean() повертає значення true або false.

2.1.2 *Генерація нерівномірних розподілів*

2.1.2.1 **Метод оберненої функції**

Метод оберненої функції є універсальним способом формування випадкових чисел з будь-яким законом розподілення. В цьому методі використовується властивість інтегральної функції розподілення відображати випадкові числа, які підпорядковуються її закону розподілення, у випадкові числа, які рівномірно розподілені в діапазоні від 0 до 1. Зворотне судження також є дійсним. Графічну інтерпретацію цього методу розглянемо на прикладі рівномірного розподілення. На рисунку 2.2 зображений графік інтегральної функції розподілення ймовірностей для випадкової величини x, рівномірно розподіленої в діапазоні від **a** до **b.**

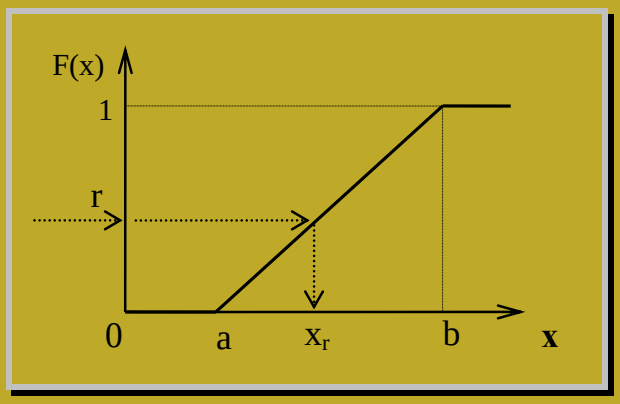

Рисунок 2.2 – Графічна інтерпретація методу оберненої функції для рівномірного розподілення

Якщо отримати деяке випадкове число **r,** рівномірно розподілене в діапазоні від 0 до 1, і відкласти його на осі ординат графіка інтегральної функції розподілення, то відповідні йому числа на осі абсцис будуть підпорядковуватись закону, який задається цією інтегральною функцією. Тобто обернена функція перетворює **r** в $\mathbf{x_r}$.

Для рівномірно розподілених чисел зв'язок між **r** та $\mathbf{x_r}$ може бути заданий аналітично, у вигляді формули 2.4, яку можна одержати, якщо розглянути подібні трикутники на рисунку 3.1.

$$x_r = a + r * (b - a) \qquad (2.4)$$

де r – рівномірно розподілене від 0 до 1 випадкове число.

Саме ця формула використовується для генерації рівномірно розподілених в заданому діапазоні випадкових чисел у класі Uniform, який входить до складу пакету rnd.

2.1.2.2 **Довільний закон розподілення.**

При використанні методу оберненої функції для отримання випадкових чисел з довільним законом розподілення, інтегральна функція для цього закону розподілення представляється відрізками прямих ліній, рисунок 2.3. В результаті такої заміни функція приблизно може бути задана за допомогою двох масивів - X і F, що задають координати стиковки відрізків.

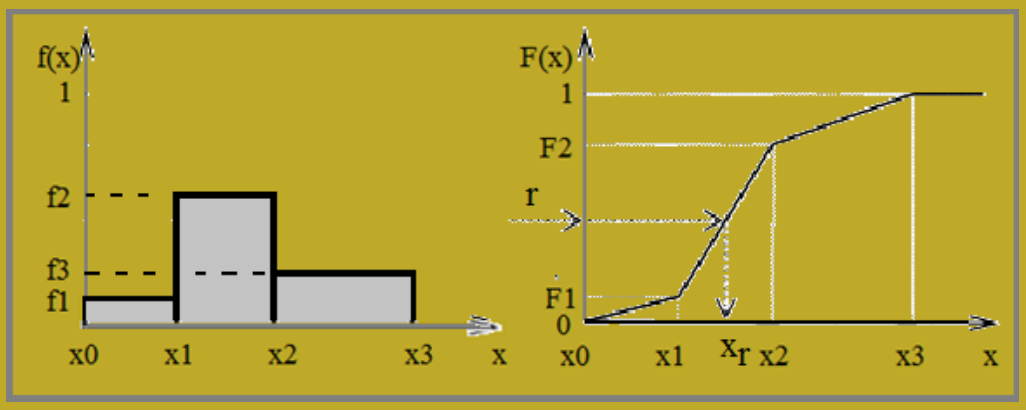

Рисунок 2.3 - Графічна інтерпретація методу оберненої функції для довільного розподілення

Алгоритм формування випадкового числа такий:

– генерується рівномірно розподілене від 0 до 1 випадкове число r;

– визначаємо границі інтервалу на осі ординат, в який потрапляє це випадкове число, F1 та F2;

– визначаємо границі відповідного інтервалу на осі абсцис x1, x2;

– визначаємо кутовий коефіцієнт оберненої прямої k=(x2-x1)/(F2-F1);

– обчислюємо випадкове число $x_r = x1 + k*(r - F1)$.

Для генерації таких чисел в пакеті rnd знаходиться клас Linear.

2.1.2.3 **Дискретне розподілення**

Для генерації дискретних, випадкових чисел також можна використовувати метод оберненої функції. Для цього необхідно сформувати масив накоплених значень ймовірностей, який задає значення інтегральної функції розподілення. Наприклад, якщо оцінки на екзамені можуть приймати значення 2, 3, 4, 5, а імовірності їх появи 0.1, 0.4, 0.3, 0.2, то масив накоплених імовірностей буде таким: 0.1, 0.5, 0.8, 1.0.

Графік такої інтегральної функції розподілення має ступінчастий характер, рисунок 2.4. Внаслідок цього випадкові величини, отримані методом оберненої функції, можуть приймати лише дискретні значення.

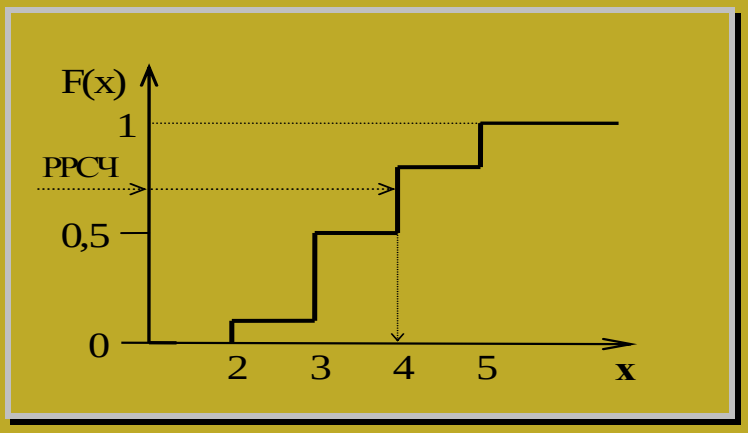

Рисунок 2.1 – Використання методу оберненої функції для дискретного закону розподілення

Описаний метод генерації випадкових дискретних чисел реалізований у класі Discret пакету rnd.

2.1.2.4 **Експоненціальний закон розподілення.**

Метод оберненої функції можна використовувати не лише в графічній інтерпретації, але й аналітично. Однак для цього необхідно отримати аналітичний вираз оберненої функції. Інколи це можливо, наприклад, для експоненціального закону розподілення.

Інтегральна функція розподілення для цього закону описується формулою 1.12.

$$F(x)=1-e^{-\lambda x} \tag{1.12}$$

Для такої функції можна знайти обернену функцію, тобто формулу для обчислення x.

$$\begin{aligned} &1-F(x)=e^{-\lambda x};\\ &-\lambda x=\ln(1-F(x));\\ &x=-\frac{1}{\lambda}\ln(1-F(x)). \end{aligned} \tag{2.5}$$

Використовуючи метод оберненої функції, в якості F(x) ми беремо випадкові числа рівномірно розподілені в діапазоні від 0 до 1, тому величина 1-F(x) також буде випадковою й рівномірно розподіленою в діапазоні від 0 до 1. Отже, для обчислення випадкових чисел, які підпорядковуються експоненціальному закону розподілення можна використовувати формулу 2.7.

$$x=-(1/\lambda)\ln r, \tag{2.6}$$

де r – РРВЧ в діапазоні [0;1].

Для генерації таких чисел в пакеті rnd міститься клас Negexp.

2.1.2.5 **Закон розподілення Ерланга**

Інтегральна функція для розподілення Ерланга має два параметри: інтенсивність **λ** - це величина обернена середньому значенню випадкової величини, і коефіцієнт **k**, який називається степенем післядії, або коефіцієнтом Ерланга. Ця функція описується досить складним виразом 1.13

$$F(x) = 1 - e^{-\lambda kx} \sum_{i=0}^{k-1} \frac{(\lambda kx)^i}{i!} \qquad (1.13)$$

З такої формули отримати вираз для обчислення x важко.

Тому для формування чисел, які підпорядковуються закону Ерланга, використовують властивість експоненціального розподілення випадкових чисел, яка полягає в тому, що сума **k** випадкових чисел, що розподілені експоненційно з інтенсивністю λ є випадковим числом Ерланга з коефіцієнтом **k** та інтенсивністю λ/k. Отже, для того щоб отримати число Ерланга з інтенсивністю **λэ** і коефіцієнтом **k,** необхідно накопичити суму **k** експоненційно розподілених випадкових чисел, що мають інтенсивність kλэ.

$$x_э = \sum_{i=1}^{k} (-\frac{1}{k\lambda_э} \ln r_i) = -\frac{1}{k\lambda_э} \sum_{i=1}^{k} \ln r_i = -\frac{1}{k\lambda_э} \ln \prod_{i=1}^{k} r_i \qquad (2.7)$$

Для генерації таких чисел в пакеті rnd знаходиться клас Erlang.

2.1.2.6 **Нормальний закон розподілення**

Нормальне розподілення є одним з найбільш поширених розподілень для випадкової величини. Але аналітичного виразу для інтегральної функції не існує бо інтеграл від функції щільності ймовірностей не береться. Тому метод оберненої функції тут не використовується.

Але для формування нормально розподілених випадкових чисел можна використовувати центральну граничну теорему теорії ймовірностей (теорему Ляпунова), яку можна сформулювати наступним чином. *Якщо випадкова величина X є сумою великої кількості взаємно незалежних випадкових величин, вплив кожної з яких на всю суму незначний, то X має розподілення, близьке до нормального розподілення.*

Для отримання взаємно незалежних випадкових величин, які мало впливають на суму, можна використовувати генератор РРВЧ і тоді формула для отримання нормально розподілених чисел X матиме вигляд:

$$X = \sum_{i=1}^{N} r_i \qquad (2.8)$$

де r_i – рівномірно розподілене випадкове число в діапазоні від 0 до 1.

Математичне очікування **m** цього числа дорівнює 0.5N, а дисперсія N/12.

Для того, щоб нормувати отримане число (тобто зробити його математичне очікування рівним нулю і дисперсію рівну 1) потрібно від числа відняти його математичне очікування і поділити на середньоквадратичне відхилення, вираз 2.9.

$$Z = ((\sum_{i=1}^{N} r_i) - 0.5N) / \sqrt{N/12} \qquad (2.9)$$

З практики відомо, що числа, які отримані по цій формулі, мають розподілення близьке до нормального вже при N > 6. Тому N можна взяти рівним 12, тоді формула для отримання нормованих нормально розподілених випадкових чисел спрощується.

$$Z = ((\sum_{i=1}^{12} r_i) - 6) \qquad (2.10)$$

Для отримання нормально розподіленого випадкового числа із заданим математичним очікуванням **m** і середньоквадратичним відхиленням **σ** достатньо нормоване випадкове число **z** помножити на потрібне середньоквадратичне відхилення і додати потрібне математичне очікування.

Формула для отримання нормально розподілених випадкових чисел Y із заданими параметрами **m** і **σ** буде такою:

$$Y = ((\sum_{i=1}^{12} r_i) - 6)\,\sigma_y + m_y \qquad (2.11)$$

Для генерації таких чисел пакет rnd містить клас Norm.

2.1.2.7 **Трикутний закон розподілення.**

Інтегральна функцію для трикутного має вигляд виразу 1.15.

$$F(x) = \begin{cases} 0, \text{ якщо } x <= a \\ \dfrac{(x-a)}{(c-a)(b-a)} \quad \text{якщо } a<x<=b \\ \dfrac{(c-x)}{(c-a)(c-b)} \quad \text{якщо } b<x<=c \\ 1, \text{ якщо } x>=c \end{cases} \qquad 1.15)$$

Такий вираз дозволяє отримати обернену функцію.

$$x = \begin{cases} a + \sqrt{r(c-a)(b-a)} & \text{якщо } r \le \dfrac{b-a}{c-a} \\ c - \sqrt{(1-r)(c-a)(c-b)} & \text{якщо } r > \dfrac{b-a}{c-a} \end{cases} \qquad (3.18)$$

Для генерації таких чисел пакет rnd містить клас Triangular.

2.1.3 *Потоки випадкових подій*

Потік випадкових подій – це послідовність якихось подій у часі, розділених випадковими інтервалами часу.

Існує декілька класифікацій таких потоків.

Перш за все слід розрізняти стаціонарні потоки подій і нестаціонарні. У стаціонарному потоці закон розподілу випадкової величини інтервалу між

подіями та його параметри не змінюються з часом. Якщо ж закон розподілу для інтервалу, або його параметри з час змінюються, то такий потік буде нестаціонарним. Наприклад, потік тролейбусів на зупинці у години «пік» можна вважати стаціонарним, але той же потік на протязі доби явно нестаціонарний, хоча б тому, що вночі тролейбуси зовсім не ходять.

Для стаціонарних потоків основна ознака для класифікації - це закон розподілу випадкової величини інтервалу часу між двома послідовними подіями.

Експоненціальний потік – це потік у якому інтервал між подіями підкоряється експоненціальному закону розподілу. Це дуже розповсюджений потік подій. За звичай він виникає у тих випадках, коли події виникають під впливом різноманітних, не пов’язаних між собою обставин. Прикладами таких подій можуть бути поява покупця у магазині, виклик лікаря швидкої допомоги і таке інше. Внаслідок своєї популярності потік має і декілька інших назв – пуасонівський потік, найпростіший потік.

Потік Ерланга – цей потік у якому інтервал між подіями підпорядковується закону Ерланга. Він формується з попереднього, шляхом вилучення із нього деякої кількості послідовних подій. Внаслідок цього потік ще звуть просіяним пуасонівським потоком. Такий потік може виникнути, наприклад, під час опитування перехожих на вулиці, коли опитують кожного третього.

Нормальний, або гаусовський потік - це потік у якому інтервал підпорядковується нормальному розподілу. Інтервали між подіями у такому потоці хоча і випадкові, але не дуже відрізняються один від одного. Приклад такого потоку – поява покупців на виході з супермаркету, де вони перед цим стояли у черзі до каси, поява тролейбуса на зупинці, у разі, якщо розкладом передбачено сталий інтервал руху. Такий потік може сформуватися і з найпростішого, якщо коефіцієнт просіювання достатньо великий. Так, наприклад, потік бажаючих виїхати до Києва у недільний вечір буде пуасонівським, але потік автобусів, що від’їжджають, як тільки завантажаться, можна вважати нормальним.

2.2 Опис програмного комплексу

До складу програмного комплексу, що забезпечує виконання лабораторної роботи, входить клас Randoms та візуальний клас TestRandomView.

2.2.1.1 Клас Randoms

У зв'язку з тим, що стандартний клас java.util.Random не допускає налаштування параметрів генератора, що необхідно при виконанні лабораторної роботи, виникла необхідність створення класу, подібного до класу java.util.Random, який дозволяє налаштування параметрів. Шляхом наслідування цю задачу вирішити було неможливо, бо ті параметри класу java.util.Random, що нас цікавлять, мають специфікатор **final**. Тому був

створений клас Randoms, що успадковує клас Object.

Клас створювався виходячи з вимог лабораторної роботи і тому трохи простіше свого прототипу. Спрощення, насамперед, торкнулося максимальної довжини формованих чисел. Максимальна кількість біт, що визначають довжину числа, була взята 31. Це дозволяє формувати цілі додатні числа типу **int**. Дійсні числа створюються в діапазоні [0..1] і відповідають типу float.

Стандартні константи адитивно-мультиплікативного методу взяті такі ж, як і в класі java.util.Random.

Конструктор класу Randoms забезпечує затримку перед ініціалізацією «зерна» генератора. Така затримка запобігає однаковій ініціалізації послідовно створюваних генераторів.

2.2.1.2 **Класу TestRandomView**

Цей клас забезбечує тестування генератора рівномірно розподілених чисел і використовується як застосування для виконання лабораторної роботи.

На рисунку 2.5 показано інтерфейс користувача класу TestRandomView.

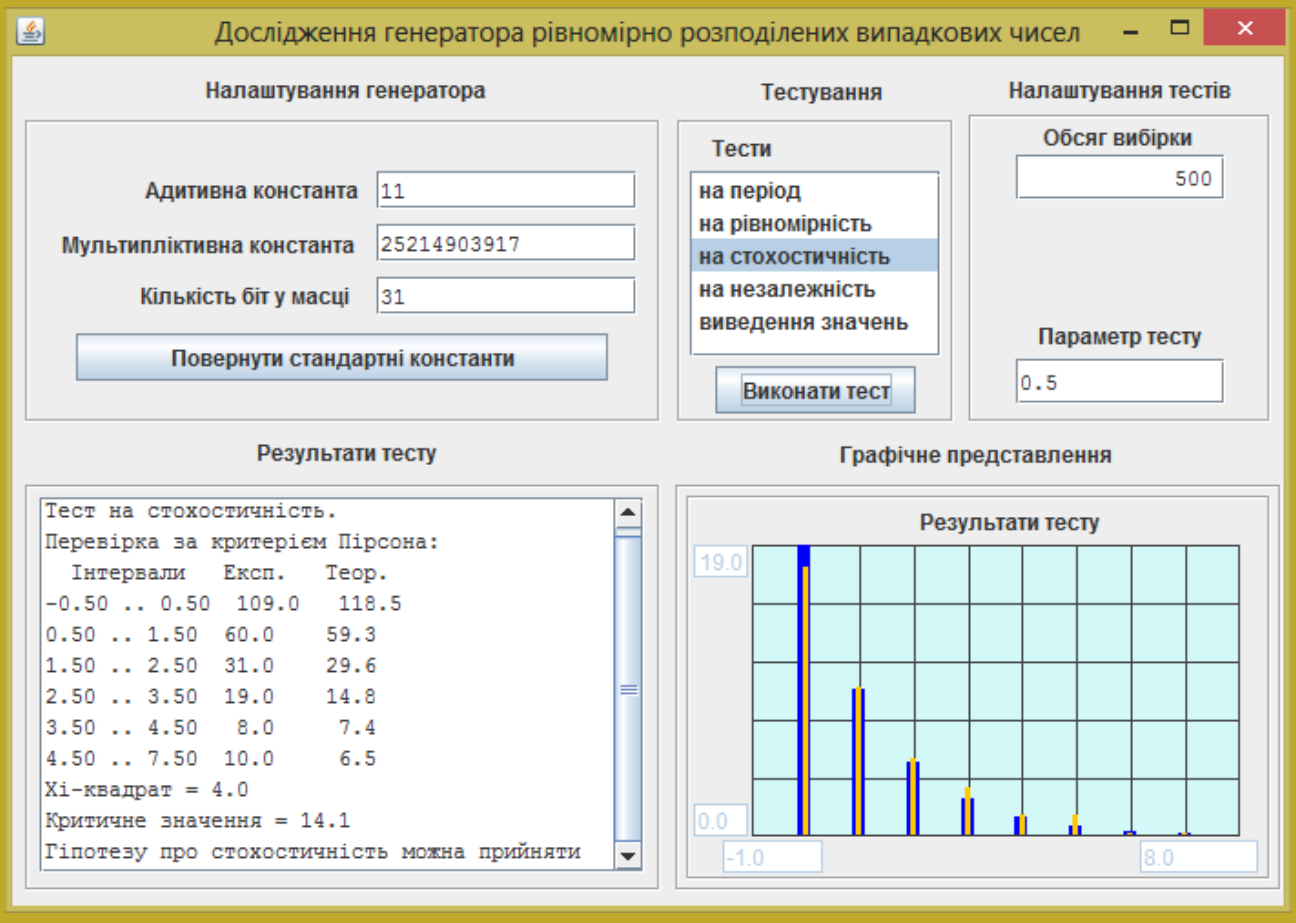

Рисунок 2.5 – Інтерфейс користувача класу TestRandomView після виконання тесту на стохостичність

Група елементів «Налаштування генератора» містить поле для введення кількості біт у масці, що визначають модуль випадкових чисел. Ця величина повинна бути в межах від 1 до 31, відповідно маска буде приймати значення від 1 до 2^{31} -1, а модуль може змінюватися від 2 до 2^{31}. Тут же розташовані поля для

введення адитивної і мультиплікативної констант генератора. Кнопка «Повернути стандартні налаштування» дозволяє відновити значення параметрів генератора, що визначені у класі Random.

Група елементів «Тести» містить список тестів і кнопку для виклику обраного тесту.

Група елементів «Налаштування тестів» містить поля для введення параметрів, необхідних для виконання тестів. При виборі тесту вигляд панелі змінюється відповідно до обраного тесту.

Поле «Обсяг вибірки» визначає, скільки випадкових чисел буде досліджуватися, і використовується для всіх тестів, крім тесту «на період»,. При виборі тесту «на період», назва поля змінюється, і в цьому випадку воно визначає, скільки перших випадкових чисел буде пропускатися перед визначенням періоду.

Поле «Кількість інтервалів» (на рисунку 2.5 його не видно) використовується в тесті на рівномірність і визначає, на скільки частин буде розбитий інтервал від 0 до 1 при побудові гістограми. Значення цього поля автоматично корегується при зміні поля «Обсяг вибірки», але може встановлюватися й довільно. При виборі тесту «на період», назва поля змінюється, і в цьому випадку воно визначає максимальний час очікування в мілісекундах при виконанні тесту.

Під назвою «Параметр тесту» може з'являтися одне із двох значень, залежно від виду тесту.

При виборі тесту «на стохастичність», з'являється поле для введення дробового числа з інтервалу 0..1, що розділяє цей інтервал на дві частини. Це число є параметром методу серій.

При виборі тесту «на незалежність», з'являється поле для введення цілого числа, що визначає найбільшу величину зміщення між числами в послідовності, які порівнюються. За замовчуванням ця величина дорівнює 100.

Результати тестів виводяться на текстову й графічну панелі вікна.

2.3 Порядок виконання роботи

2.3.1 *Підготовка до роботи*

– Активізуйте файл SimulationAllLab2018.jar, або завантажте проект з архіву SimulationAllLab2018.zip та активізуйте стартовий файл застосування mainUI, що знаходиться у папці main проекту.

2.3.2 *Перевірка працездатності програмного комплексу*

– Перейдіть до теми «Генератори випадкових чисел»

– Натисніть на кнопку «Лабораторне застосування». Переконайтеся в працездатності додатку.

– Натисніть на кнопку «Тестування файлів випадкових величин». Переконайтеся в працездатності додатку.

– Натисніть на кнопку «Демонстрація потоків випадкових подій». Переконайтеся в працездатності додатку

2.3.3 *Дослідження генератора випадкових чисел*

Активізуйте додаток «Лабораторне застосування».

2.3.3.1 **Візуальний аналіз послідовностей випадкових чисел**

– Встановіть стандартні налаштування генератора. Виведіть послідовність випадкових чисел на перегляд. Подивіться, як вона змінюється від тесту до тесту.

– Зробіть кількість біт у масці невеликою, наприклад, рівною 8, і проаналізуйте кілька послідовностей. Пояснить причину зміни довжини чисел.

– При такій же кількості біт встановіть значення мультиплікативної константи рівним 1 і проаналізуйте кілька послідовностей. Поясніть причину зміни характеру послідовності.

– При такій же кількості біт встановіть значення мультиплікативної константи рівним 4 і проаналізуйте кілька послідовностей. Пояснить причину зміни характеру послідовності.

– При такому ж модулі й значенні мультиплікативної константи встановіть значення адитивної константи рівним 0 і проаналізуйте кілька послідовностей. Поясніть причину зміни характеру послідовності.

– Для всіх розглянутих варіантів зафіксуйте найбільш характерні послідовності у звіті.

2.3.3.2 **Дослідження періоду послідовності**

– Встановіть стандартні налаштування генератора. Проведіть тести «на період» відповідно до таблиці 2.4. Зробіть висновок про вплив модуля генератора на його період генератора й зафіксуйте висновок у звіті.

Таблиця 2.4 – Залежність періоду генератора від розмірів маски при стандартних налаштуваннях

Номер досліду	Довжина маски в бітах		
	8	16	24
1			
2			
3			

– Встановіть стандартне значення мультиплікативної константи й довжину маски рівну 16. Проведіть тест «на період» для значень адитивної константи відповідно до таблиці 2.5. Зробіть висновок про вплив значень константи на період генератора й зафіксуйте його у звіті.

Таблиця 2.5 – Залежність періоду генератора від адитивної константи при розмірі маски 16 і стандартній мультиплікативній константі

Номер досліду	Значення адитивної константи			
	11(стандарт)	16	100	173

1				
2				
3				

– Встановіть стандартне значення адитивної константи й довжину маски рівну 16. Проведіть тест «на період» для значень мультиплікативної константи відповідно до таблиці 2.6. Зробіть висновок про вплив значень константи на період генератора й зафіксуйте його у звіті.

Таблиця 2.6 – Залежність періоду генератора від мультиплікативної константи при розмірі маски 16 і стандартної адитивної константи

Номер досліду	Значення мультиплікативної константи			
	стандарт	17777	34367	34366
1				
2				
3				

У тих випадках, коли період виходить дуже маленьким, проаналізуйте послідовність, викликавши її за допомогою пункту «виведення значень».

2.3.3.3 **Дослідження рівномірності послідовності**

– Встановіть стандартні налаштування генератора й виберіть тест «на рівномірність». Проведіть тести для різних значень розміру вибірки й кількості інтервалів відповідно до таблиці 2.7. Зробіть висновок про рівномірність отримуваних послідовностей і вплив параметрів тесту на результат тестування.

Таблиця 2.7 – Результати тестування генератора на рівномірність при стандартних налаштуваннях

Параметри тесту		Загальна кількість тестів	Кількість тестів з негативним результатом	Оцінка ймовірності появи нерівномірності
Обсяг вибірки	Кількість інтервалів			
200	8	20		
	5	20		
	15	20		
1000	11	20		
	5	20		
	20	20		

2.3.3.4 **Дослідження стохастичності послідовності**

– Встановіть стандартні налаштування генератора й виберіть зі списку тест «на стохастичність». Проведіть тести для різних значень розміру вибірки й параметра тесту відповідно до таблиці 2.8. Зробіть висновок про рівномірність отримуваних послідовностей і вплив параметрів тесту на результат тестування.

Таблиця 2.8 – Результати тестування генератора на стохастичність при стандартних налаштуваннях

Параметри тесту		Загальна кількість тестів	Кількість тестів з негативним результатом	Оцінка ймовірності стохастичності послідовності
Обсяг вибірки	Параметр тесту			
200	0,3	20		
	0,5	20		
	0,7	20		
1000	0,3	20		
	0,5	20		
	0,7	20		

2.3.3.5 **Дослідження незалежності послідовностей**

– Виберіть зі списку тест «на незалежність».

– Встановіть значення параметра тесту «на незалежність» рівним 100. У цьому випадку досліджується незалежність послідовностей, які створюються з однієї послідовності шляхом зміщення початку послідовностей. Розмір зміщення змінюється у тесті від 1 до значення, що задається параметром тесту.

– Встановіть стандартні налаштування генератора. Проведіть тест кілька разів і оцінить ймовірність перевищення критичного значення коефіцієнта кореляції. Результат зафіксуйте в таблиці 2.9.

Таблиця 2.9 – Результати дослідження незалежності послідовностей випадкових чисел. Загальна кількість точок N=100

	Дослід 1	Дослід 2	Дослід 3
Перевищень критичного значення(n)			
Імовірність перебільшення(n/N)			

2.3.4 *Робота з програмою «Тестування файлів випадкових величин»*

Активізуйте додаток «Тестування файлів випадкових величин».

Цей додаток надає можливість аналізувати вибірки випадкових чисел, що записані у текстових файлах, де одне число займає один рядок. Інтерфейс користувача представлений на рисунку 2.6.

Група елементів у лівій верхній частині форми містить компоненти через які задаються параметри файлу.

Ім’я файлу вводиться в поле «Ім’я вибраного файлу». У разі необхідності, за допомогою кнопки «Вибір файлу» можна відкрити діалогове

вікно вибору файлу. Ім’я вибраного файлу буде зафіксовано.

Поле «Обсяг вибірки у файлі» показує розмір вибірки файлу, що обробляється.

Перелік можливих операцій із файлом випадкових чисел визначається набором компонентів у верхній правій частині форми.

Результати тесту виводяться як у вигляді тексту, так і у вигляді гістограм.

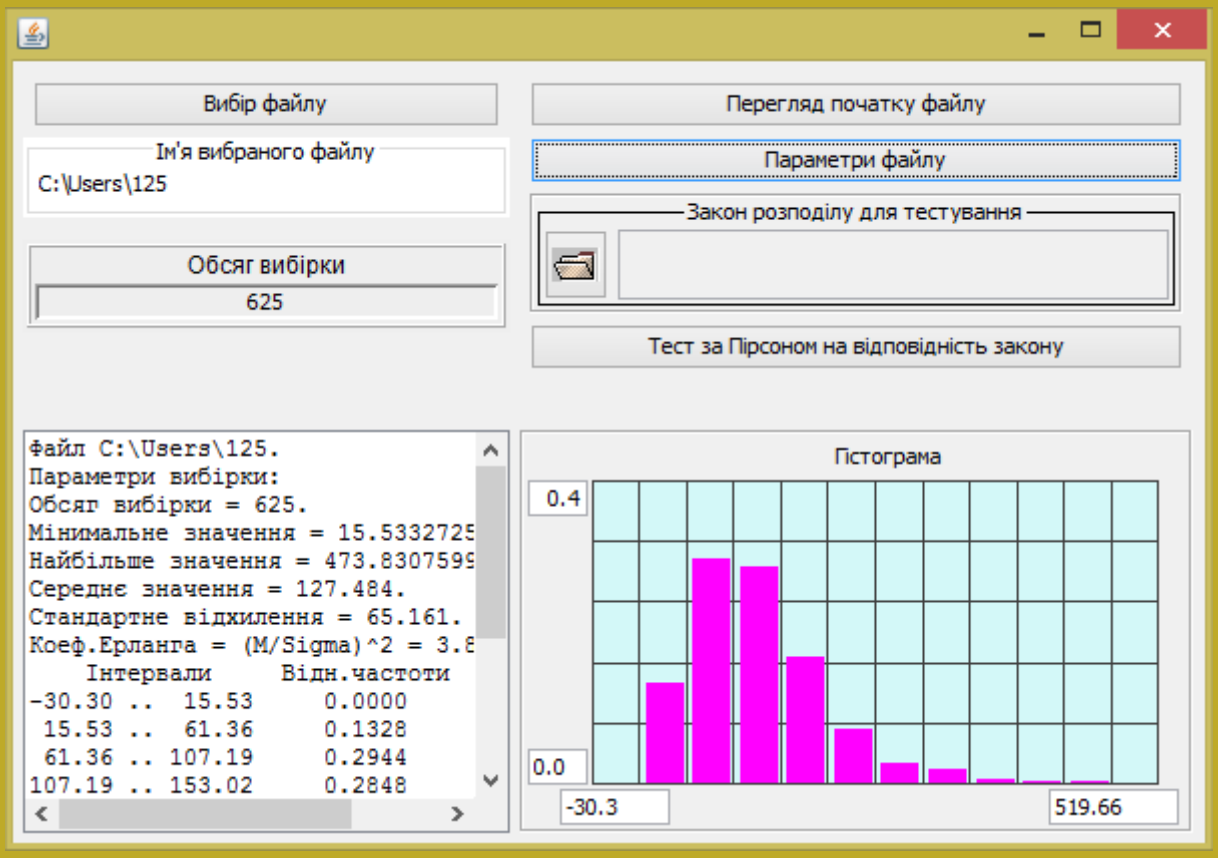

Рисунок 2.6 – Інтерфейс додатку для тестування файлів з числами

Порядок роботи із застосуванням може бути наступним:

– Обираємо файл з яким необхідно провести тест.

– За допомогою функції «Перегляд початку файлу» аналізуємо дані, і визначаємо, з якою випадковою величиною доводиться мати справу – дискретною або безперервною.

– За допомогою функції «Параметри файлу» отримуємо емпіричне розподілення випадкової величини у вигляді гістограми і числові характеристики вибірки випадкових чисел.

– На підставі аналізу отриманих даних висовуємо гіпотезу про закон розподілення, якому відповідає досліджувана вибірка і налаштовуємо компонент ChooseRandom згідно з гіпотезою.

– Перевіряємо висунуту гіпотезу за критерієм Пірсона.

2.3.4.1 **Завдання до цього етапу лабораторної роботи**

Обробіть файли з даними, і визначте, якому закону розподілення вони відповідають. Файли можна знайти в архіві filesForLab2.zip. Імена файлів

отримуємо шляхом додавання числа, яке відповідає останнім двом цифрам залікової книжки, до чисел 0, 50, 100.

Гіпотезу про відповідний закон розподілення слід висувати на підставі вигляду гістограми та числових характеристик розподілу (середнє значення, стандартне відхилення, коефіцієнт Ерланга, мінімальне та максимальне значення).

2.3.5 *Робота з програмою «Тестування потоків випадкових подій»*

Активізуйте додаток «Тестування файлів випадкових величин».

Цей додаток надає можливість візуально досліджувати потоки випадкових подій. Інтерфейс користувача додатку представлено на рисунку 2.7.

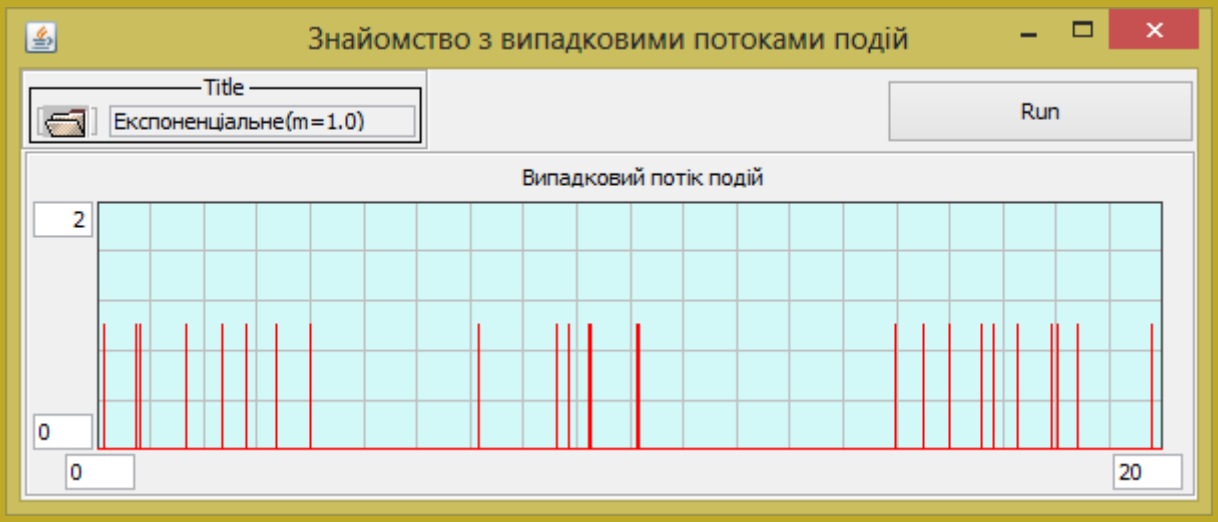

Рисунок 2.7 – Інтерфейс додатку для демонстрації потоків випадкових подій

У лівому верхньому куті знаходиться компонент для вибору і налаштування потрібного розподілу випадкової величини інтервалу між подіями. Праворуч знаходиться кнопка для створення потоку подій.

Кожна подія візуалізується вертикальною рискою.

Запустіть програму і проаналізуйте особливості потоків подій з різними законами для інтервалу між подіями.

Зафіксуйте у звіті вигляд цих діаграм.

2.3.6 *Тестування*

Запустіть програму тестування і дайте відповіді на запитання. Робота може бути зарахована, якщо ви захистили звіт, виконали завдання для самостійної роботи та отримали по тесту не менше 30 балів.

2.4 Завдання для самостійної роботи

2.4.1 *Реалізація наступного кроку у розробці РГР*

Налаштуйте закони розподілів та інші параметри моделі, які мають з’являтися після запуску додатку.

Додайте до візуального класу (див. зразок РГР), панель динамічної індикації роботи моделі (панель “Test”).

Визначте перелік параметрів, що підлягає динамічній індикації у режимі «Тест». Додайте на панель і налаштуйте відповідні діаграми.

Діаграми мають автоматично змінювати свої налаштування у разі зміни налаштувань моделі. Копію інтерфейсу користувача з налаштуваннями додайте до звіту з лабораторної роботи.

2.5 Зміст звіту

- Найменування й мета роботи
- Результати дослідження генератора РРВЧ відповідно до порядку виконання роботи у вигляді таблиць і пояснень отримуваних результатів.
- Результати тестування файлів з числовими даними у вигляді копій екранів з результатами тестування.
- Результати дослідження випадкових потоків подій у вигляді копій екранів.
- Висновки про вплив налаштувань генератора на його роботу
- Висновок про можливість застосування об'єктів класу Random для моделювання.

2.6 Контрольні питання

- Методи формування випадкових чисел.
- Характеристики рівномірного розподілу.
- Конгруентний адитивно-мультиплікативний метод формування випадкових чисел. Отримати вручну послідовність 10 випадкових чисел для модуля 10.
- Вплив обраних значень констант генератора на характеристики послідовності й рекомендації з вибору цих констант.
- Порівняння класів Random і Randoms.
- Що таке період генератора випадкових чисел і від чого залежить його величина. Методика визначення періоду генератора.
- Методика перевірки вибірки випадкових чисел на рівномірність за допомогою критерію Пірсона. Показати на конкретному прикладі.
- Методика перевірки вибірки випадкових чисел на стохастичність. Показати на конкретному прикладі.
- Що означає вираз 'послідовності випадкових чисел незалежні'. Приведіть приклади залежних послідовностей. Методика перевірки послідовностей на незалежність.
- Суть методу оберненої функції для формування випадкових чисел, які розподілені за певним законом. Проблеми, що виникають під час застосування цього методу.

– Методика отримання випадкових чисел, розподілених за експоненціальним законом. Описання класу Negexp.

– Методика отримання випадкових чисел, розподілених за законом Ерланга. Описання класу Erlang.

– Методика отримання випадкових чисел, розподілених за нормальним законом. Описання класу Norm.

– Методика отримання випадкових чисел, розподілених за законом користувача. Описання класу Linear.

– Методика отримання дискретних, випадкових величин розподілених за заданим законом. Описання класу Discret.

– Методика отримання рівномірно розподілених в деякому діапазоні випадкових величин. Описання класу Uniform.

– Методика тестування генераторів випадкових величин.

Рекомендована література

1. Томашевський В.И.Моделювання систем. – К.: Видавнича група BHV, 2005. –352с.:іл.

2. Советов Б.Я., Яковлев С.А. Моделирование систем: Учебник для вузов. – М.: Высш.шк., 1998. –320с.

3 ЛАБОРАТОРНА РОБОТА № 3. ПОБУДОВА ІМІТАЦІЙНИХ МОДЕЛЕЙ СИСТЕМ МАСОВОГО ОБСЛУГОВУВАННЯ

Мета роботи:

– Знайомство з особливостями моделей систем масового обслуговування (СМО).

– Знайомство з методикою побудови моделі СМО.

3.1 Системи масового обслуговування

Система масового обслуговування (СМО), або система з чергами (queued system) - це система, для якої характерна наявність таких компонент:

- випадковий потік заявок (транзакцій) на обслуговування;
- черги заявок, що чекають обслуговування;
- пристрої, що обслуговує ці заявки.

На рисунку 4.1 представлено схематичне зображення найпростішої системи масового обслуговування.

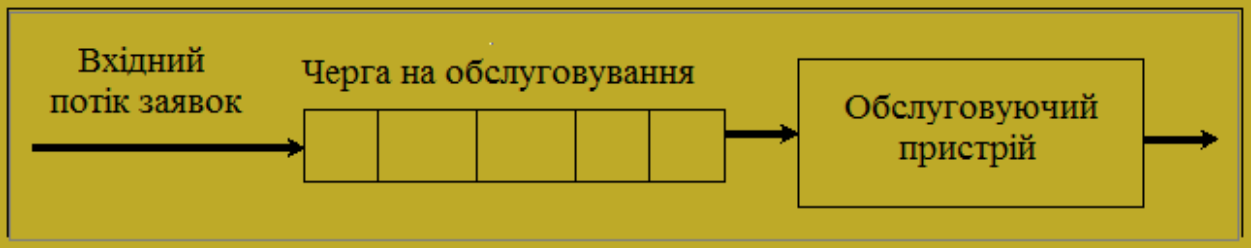

Рисунок 3.1 – Схема найпростішої СМО

Але СМО може мати и складнішу структуру. На рисунку 3.2 показано схему багатоканальної двофазної СМО.

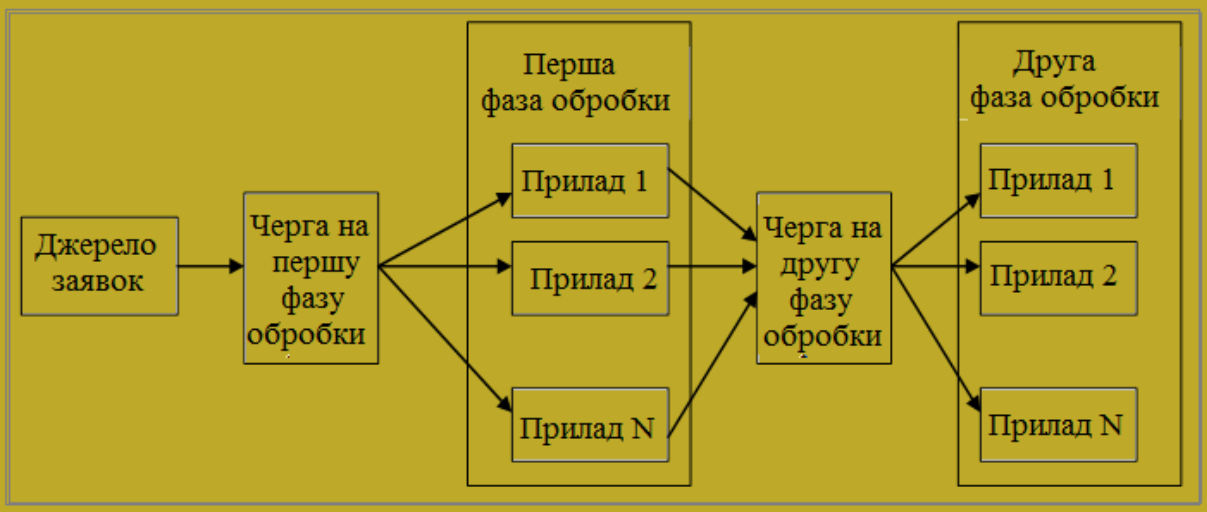

Рисунок 3.2 – Схема двофазної багатоканальної СМО

Ця система є багатоканальною з тієї причини, що кожна фаза обробки заявок виконується паралельно декількома обслуговуючим приладами.

А внаслідок того, що кожна заявка потребує обробки спочатку на одному з приладів першої групи, по потім на приладі другої групи, система є двофазною.

На рисунку 3.3 зображено СМО із каналом зворотного зв’язку. Такі СМО називають СМО із зворотними зв’язками.

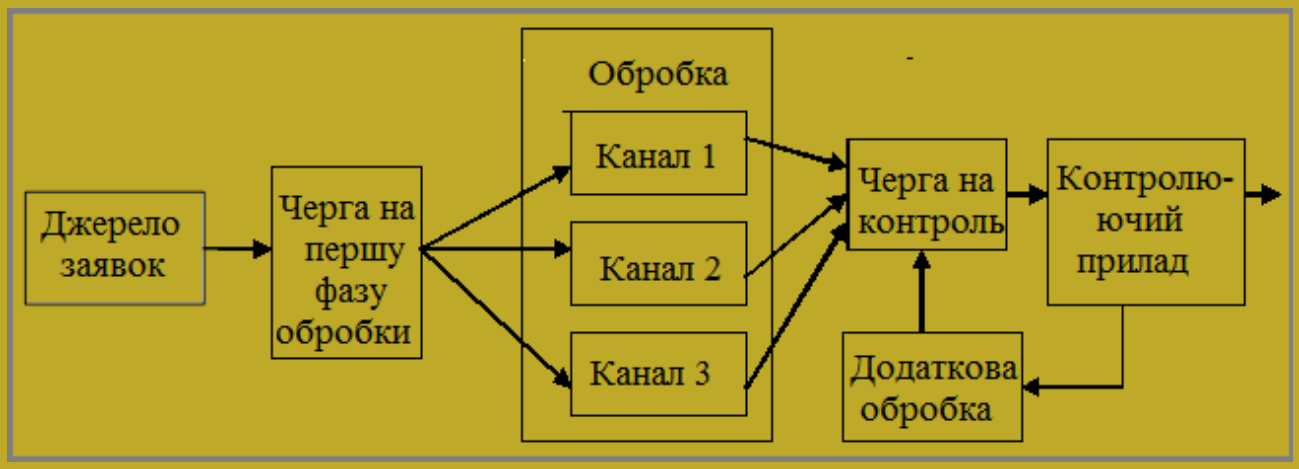

Рисунок 3.3 – Схема двофазної багатоканальної СМО із зворотними зв’язками

Існують і мережі СМО. Прикладом такої СМО може бути комп’ютерна мережа.

В усіх розглянутих вище прикладах СМО являє собою деяку структуру до складу якої входять джерела заявок, черги та обслуговуючі пристрої. При цьому джерела заявок і обслуговуючі пристрої є активними елементами, що керують переміщенням транзакцій і створюють події у часі. Джерело заявок затримується на деякий час і після цього формує заявку і додає її до черги – це і є подія. Обслуговуючий пристрій чекає появи заявки у черзі, а дочекавшись забирає її і витрачає деякий час на роботу із заявкою. Момент завершення обслуговування теж є подією. Черги ж є пасивними елементами і використовуються як місце перебування заявок між фазами обробки. Їх стан (заповнення) залежіть від взаємодії активних елементів. Поява заявки в черзі – це теж подія і вона відбувається у певний час, але цю подію створює не черга, а той об’єкт, що додає заявку до черги.

Транзакція у такій СМО теж є пасивним елементом. Вона може зберігати у собі деяку інформацію, але не керує своїм переміщенням у системі.

Однак у деяких СМО транзакції можуть бути активними, і у окремих фазах свого життєвого циклу можуть самостійно приймати рішення про переміщення по системі. Так само і активні елементи системи у деяких випадках можуть ставати трансакціями і чекати обслуговування. Так, наприклад, хворий (заявка), втомившись у черзі чекати обслуговування може покинути чергу і піти із скаргою до головного лікаря. Лікар, що веде прийом хворих і являє собою обслуговуючий пристрій СМО, може на деякий час припинити обслуговування хворих і піти на консультацію до іншого лікаря, таким чином перетворившись у заявку. Для таких СМО структурне зображення буде неповним. Поведінка таких нестандартних елементів СМО може бути представлена схемою алгоритму або діаграмою діяльності.

При моделюванні СМО досліднику відомі характеристики окремих компонент системи, а невідомим є результат їх взаємодії. У результаті дослідження найчастіше отримують таку інформацію:

- статистичні характеристики для довжини черг;
- статистичні характеристики для часу очікування в чергах;
- статистичні характеристики для часу обслуговування заявок;
- коефіцієнт завантаження обслуговуючих приладів.

Окрім того метою дослідження може бути отримання залежності наведених вище характеристик від параметрів СМО (кількості обслуговуючих пристроїв, характеристик закону обслуговування і таке інше).

Перелічені вище задачі стосуються сталого режиму роботи СМО, але інколи досліджують і перехідні процеси у СМО.

3.2 Компоненти фреймворку Simulation для побудови моделі

Фреймворк Simulation надає розробнику моделі достатньо ефективні засоби побудови складових частин моделі. Класи ChooseRandom, ChooseData, Histo, DiscretHisto, Diagram, Painter уже розглядалися у першій лабораторній роботі. Ці класи спрощують створення інтерфейсу користувача і використовуються для генерації випадкових величин, накопичення та відображення статистичної інформації.

Нижче наводиться стислий опис ще декількох важливих класів, що використовуються для створення моделі та її складових частин.

3.2.1 *Клас Actor*

На базі цього класу можна створювати класи для активних компонентів моделі. Активним компонентом є такий компонент моделі, який має поведінку, що характеризується затримками у часі або затримками до виконання деякої умови. Клас передбачає реалізацію таких правила дії у класах спадкоємцях у вигляді методів rule().

У наступній лабораторній роботі цей клас розглядається докладніше.

3.2.2 *Клас MultiActor*

Цей клас використовується для створення груп однакових об'єктів (бригад), що моделюють багатоканальну обробку в СМО. При такому обслуговуванні група паралельно працюючих приладів працює з однією чергою. Для налаштування об’єкту цього класу потрібно визначити значення двох атрибутів:

nClone – визначає кількість копій.

original – містить посилання на зразок, з якого будуть зроблені копії.

Посилання на ці атрибути можна передати за допомогою конструктора, або через методи setNumberOfClones(int) та setOriginal(Actor)

Клас успадковує клас Actor і у ньому визначено метод rule(). Під час виконання цього методу створюються копії оригіналу шляхом клонування і ці копії одразу записуються до стартового списку диспетчера.

Для клонування використовується метод clone() класу Actor. Тут слід зазначити, що операція клонування в Java є операцією поверхневого копіювання (shallowCopy). При такому копіюванні створюється копія області пам'яті, що містить інформацію про об'єкт. Внаслідок цього ми отримуємо копії значень для примітивних типів і типу String, а для решти класів отримуємо не копії об’єктів, а копії посилань на об'єкти. Таким чином, поля всіх клонованих об'єктів посилаються на ті самі об'єкти, що визначаються зразком.

При клонуванні акторів, що забезпечують багатоканальну обробку, такий спосіб клонування нас майже влаштовує. Всі клони будуть посилатися на ту саму чергу, на той самий генератор випадкових чисел, у всіх буде той самий диспетчер. Імена клонів можна робити різними. Єдиний недолік поверхневого копіюванні в тім, що всі клони будуть мати той самий семафор (об'єкт класу process.Semaphore). Для того, щоб усунути цей недолік, у класі Actor перевизначений базовий метод clone(), у якому для кожного клону створюється власний об'єкт класу process.Semaphore.

Цей метод слід перевизначити і у спадкоємців класу Actor, якщо вони мають посилання на об’єкти, що мають бути унікальними для кожної копії.

3.2.3 *Клас Dispatcher*

Клас Dispatcher визначає властивості і поведінку об’єкту, що забезпечує синхронізацію роботи у часі об’єктів типу Actor, а також керування модельним часом. Докладно клас буде розглянуто у наступній лабораторній роботі

3.2.4 *Клас QueueForTransactions*

Клас використовується для моделювання черг у системах масового обслуговування. Для збереження об’єктів, що потрапляють до черги, клас має поле queue типу ArrayDeque, але робота з цим об’єктом забезпечується через методи класу QueueForTransactions.

Об'єкти класу QueueForTransactions можуть відображати свій стан у вигляді діаграми, використовуючи об'єкти класу paint.Painter, накопичувати інформацію про поточні розміри черги у об’єкті класу DiscretHisto, а також виводити інформацію про зміну розмірів до протоколу роботи моделі.

Для виконання цих завдань черга має мати посилання на диспетчера, під керівництвом якого працює імітаційна модель, щоб отримувати значення поточного часу.

Окрім того, для відображення стану черги на діаграмі, цій черзі слід передати посилання на об'єкт класу paint.Painter, за допомогою методу setPainter(Painter). А цей об'єкт класу paint.Painter, у свою чергу, має мати посилання на якусь діаграму.

А для накопичення статистичних даних про розмір черги їй слід передати посилання на об'єкт класу DiscretHisto за допомогою методу setDiscretHisto(DiscretHisto).

Для об’єктів даного класу використовується також поняття максимального розміру, вище якого черга заповнена бути не може. У випадку спроби додавання об'єкта до заповненої черги формується подія

QueueOverflowEvent, але об'єкт не додається. Доступ до неприйнятої події можна отримати через об’єкт класу QueueOverflowEvent, створивши слухача даної події.

Перед початком роботи моделі черга повинна бути проініціалізована, якщо вона до цього використовувалася. Для цього існує метод init(). Під час ініціалізації черга очищається, а об'єкт «painter», якщо на нього є посилання, і існує діаграма, на якій буде зображуватися черга, переводиться у положення з координатами 0,0.

Зміна розмірів черги відбувається за допомогою методів addLast(Object), remove(Object), removeFirst().

Особливість методу addLast(Object) полягає в тому, що в ньому аналізується ступінь заповнення черги й, у випадку неможливості її подальшого збільшення, формується подія QueueOverflowEvent.

У методах додавання й видалення об'єктів перед зміною черги та після зміни викликається один з наступних методів - beforeAdd, beforeRemove, afterAdd, afterRemove, залежно від виконаної операції.

Методи beforeAdd, beforeRemove виводять до протоколу диспетчера інформацію про новий розмір черги, а також забезпечують відображення стану черги перед її зміною, та накопичення інформації у гістограмі, якщо визначені необхідні для цього об’єкти. Ці методи мають специфікатор доступу protected і тому можуть бути довизначені у спадкоємців класу.

3.2.5 *Клас Store*

Цей клас схожий на попередній. Різниця полягає у тому, що об’єкти цього класу накопичують значення типу double у полі size. Так само як і у попередньому класі можливо відображення поточного значення size на діаграмі і накопичення цих значень у гістограмі, тільки гістограма має бути типа Histo.

3.3 Методика побудови моделі СМО

Сучасні технології проектування програмних систем рекомендують створювати програмний продукт, як поєднання декількох шарів. Для побудови застосування, що забезпечує моделювання СМО, можна рекомендувати використовувати такі шари:

- шар подання (presentation layer);
- шар моделі;
- шар компонентів.

Схематичне зображення проекту для моделювання у вигляді взаємопов’язаних шарів представлено на рисунку 3.4.

Рисунок 3.4 - Структура застосування для моделювання СМО

3.3.1 *Шар подання*

За звичай першим шаром програмного продукту є шар подання. У нашому випадку це не щось інше, як графічний інтерфейс користувача, що є посередником між користувачем і самою моделлю. Основні завдання цього шару такі:

- отримання від користувача налаштувань компонент моделі;
- надання доступу до компонент, з налаштуваннями користувача;
- забезпечення запуску моделі;
- надання засобів для динамічної індикації процесу моделювання;
- надання засобів для відображення результатів моделювання.

У випадку використання фреймворку Simulation для побудови моделі рекомендовано у шарі подання реалізувати такі завдання:

- створення диспетчера;
- створення фабрики моделей;
- створення моделі;

- передачу моделі посилання на диспетчера;
- передачу моделі посилання на інтерфейс користувача;
- відображення результатів моделювання, отриманих від моделі.

Таким чином, шар подання реалізується візуальним класом з необхідним набором візуальних елементів і має містити метод у якому створюються диспетчер, фабрика моделей, модель та відбувається запуск процесу моделювання.

Створення моделі перед кожним її запуском спрощує програмування, хоча дещо уповільнює роботу програми, бо усі об’єкти створюються знов.

Для створення моделі слід використовувати фабрику моделей, що має реалізувати інтерфейс IModelFactory. Інколи без використання фабрики можна обійтись, але це є обов’язковим, якщо у програмному додатку будуть використовуватися компоненти, що забезпечують автоматизоване проведення серій багаторівневих експериментів з моделлю або дослідження перехідних процесів.

Відповідно до вимог інтерфейсу IModelFactory у фабриці має бути реалізований усього один метод – createModel(Dispatcher). Але, окрім того фабрика має ще передавати моделі, через конструктор моделі, посилання на візуальну частину, або інший об’єкт, з якого модель буде отримувати налаштування.

Для спрощення завдань, пов’язаних із створенням моделі, виходячи із того, що інтерфейс IModelFactory має усього один метод, можна рекомендувати створювати фабрику за допомогою лямбда функції.

Для запуску моделі використовується метод диспетчера start().

Відображення результатів моделювання, отриманих від моделі після завершення її роботи, можна пов’язати із подією DispatcherFinishEvent, створивши відповідного слухача цієї події. Інший варіант розв’язання цієї задачі полягає у тому, щоб після запуску моделі створити потік виведення результатів моделювання і приєднати його до потоку диспетчера за допомогою методу join().

Взагалі, для побудови інтерфейсу користувача достатньо стандартних засобів, що надає Java, але використання компонентів фреймворку Simulation спрощує цю задачу.

На етапі проектування шару подання першочергову увагу слід приділити визначенню публічних методів класу (або інтерфейсу), що будуть надавати доступ до компонентів моделі, що містять налаштування користувача. Це дозволить доручити розробку класу одному з членів команди.

3.3.2 *Шар моделі*

Модель створює і містить у собі посилання на усі об’єкти, що моделюють складові частини досліджуваної системи і засоби для накопичення статистичної інформації, а також має створювати ці об’єкти, передавати їм посилання на себе та візуальну частину, і надавати доступ до результатів моделювання у шар подання.

Нижче наведено перелік основних завдань цього шару:

– створити усі необхідні на момент старту об’єкти моделі;
– створити засоби для накопичення статистичної інформації;
– надати публічний (або пакетний) доступ до компонент моделі;
– завантажити «акторів» моделі до стартового списку диспетчера;
– надавати шару подання доступ до результатів моделювання;
– надати методи ініціалізації компонентів у різних режимах роботи.

Шар моделі має декілька особливостей.

Перша особливість полягає у тому, що незважаючи на те, що модель нібито весь час працює, насправді це найбільш пасивний елемент. Модель - це перш за все сховище посилань на складові частини системи. У моделі має бути також метод ініціалізації, що викликається перед запуском диспетчера.

Тобто методи моделі працюють тільки на початку моделювання та після його завершення, а в процесі моделювання виконуються методи складових її частин, що утворюють третій шар програмної системи.

Можна рекомендувати таку послідовність програмної реалізації моделі:

– визначити поля класу, що відповідають усім складовим моделі (акторам, чергам, гістограмам та іншим складовим) без створення відповідних об’єктів. Зважаючи на те, що тільки модель відповідає за створення цих об’єктів, доступ до всіх складових не слід робити публічним. Слід також передбачити поля для посилань на диспетчера та візуальну частину, хоча ці компоненти не створюються у моделі;

– визначити конструктор для моделі, через параметри якого передавати посилання на диспетчера та візуальну частину. Пустий конструктор краще не визначати. Це унеможливить створення моделі без посилань на диспетчера та візуальну частину. У конструкторі окрім визначення посилань на диспетчера та візуальну частину можна завантажити усіх акторів до стартового списку диспетчера. Для цього доцільно реалізувати метод componentsToStart, у якому передати диспетчеру усіх акторів, що мають почати роботу після старту моделі, але для звернення до акторів слід використовувати методи get…();

– створити об’єкти для складових частин моделі. Цю частину класу доцільно реалізувати за методикою відкладеного створення об’єктів. Відповідно до цієї методики об’єкт створюється під час першого звернення до нього, у методі get…(), після аналізу, чи дорівнює посилання на цей об’єкт null. Більшість цих методів має бути публічними, для того, щоб об’єкти мали можливість спілкуватися між собою через модель.

Реалізуючи останній пункт доцільно використовувати конструктори з параметрами.

3.3.3 *Шар компонентів*

Цей шар складається з класів, що реалізують моделі складових частин досліджуваної системи.

Основні завдання цього шару такі:

– моделювати складові частини системи;
– реалізувати правила дії активних компонент моделі;

– передавати дані про роботу до накопичувачів статистики;
– надавати дані для динамічної індикації процесу моделювання.

Особливість цього шару полягає у тому, що до його складу входять як класи, що потребують створення, так і вже існуючі класи, які надає користувачеві фреймворк Simulation та Java.

Найчастіше, під час реалізації цього шару доводиться створювати класи для активних компоненти системи, які мають успадковувати клас Actor і у цих класах має бути реалізований метод rule(). Для реалізації черги можна використовувати колекції, але краще використовувати класи черг фреймворка Simulation. Теж саме стосується і класів для накопичення статистичної інформації.

3.4 Приклад побудови програмної системи для моделювання

Розглянемо проект, який ми будемо використовувати для досліджень у наступних лабораторних роботах. Це модель найпростішої системи масового обслуговування.

Схему такої системи зображено на рисунку 3.1. Вона складається з черги для транзакцій та обслуговуючого пристрою. Окрім цього в моделі доведеться використовувати генератор транзакцій, який на схемі відсутній.

Нам потрібно створити Java застосування для імітаційного моделювання такої системи, яке дозволить:

– налаштування параметрів моделі, а саме: закони розподілення для випадкових величин інтервалу часу між появами транзакцій та часу обслуговування транзакції, кількість обслуговуючих пристроїв, тривалість моделювання;

– проведення тестових запусків моделі при різних налаштуваннях з динамічною індикацією розміру черги, та виведення протоколу роботи моделі під час тестових запусків;

– проведення експериментів для отримання статистичних даних про довжину черги, час перебування транзакції у черзі та загальний час перебування у системі, час простою обслуговуючого пристрою.

3.4.1 *Аналіз системи*

Аналіз системи полягає у визначенні абстракцій, що входять до складу системи і дозволяють вирішувати поставлені завдання.

3.4.1.1 **Абстракція «генератор заявок»**

Абстракція «генератор заявок» моделює зовнішнє середовище системи з якого з’являються транзакції. Завдання «генератора заявок» полягає у тому, щоб через випадкові інтервали часу створювати нові транзакції і додавати їх до черги. Характерною рисою цієї абстракції є наявність затримок на випадковий час перед створенням кожної транзакції. Тому для програмної реалізації абстракції слід створити клас, який має бути нащадком класу process.Actor.

3.4.1.2 **Абстракція «обслуговуючий пристрій»**

Завдання «обслуговуючого пристрою» – обробка транзакцій, що чекають на обслуговування у черзі. Якщо черга пуста, «обслуговуючий пристрій» чекає на появу транзакції. Після появи транзакції він забирає її з черги і затримується у на випадковий час, імітуючи обробку транзакції. Далі цикл повторюється. Наявність затримок на деякий час та можливість переходу у режим чекання свідчить про те, що для програмної реалізації цієї абстракції слід створити клас, який має бути нащадком класу process.Actor.

3.4.1.3 **Абстракція «транзакція»**

Ця абстракція моделює якусь сутність, що потребує обслуговування. «Транзакція» може бути як пасивним елементом, так і активним. Ми зробимо її активною, хоча це збільшить вимоги до ресурсів комп'ютера. Цю сутність буде створювати «генератор заявок» і він же буде передавати її диспетчеру. Але транзакція сама буде додавати себе до черги, та визначати час перебування у черзі та повний час обслуговування, реагуючи на відповідні події. Виходячи з того, що транзакція може перебувати у стані очікування на ці події, вона має бути об'єктом класу Actor.

Ще раз наголосимо, що транзакція могла б бути і об'єктом звичайного класу з полем типу double, або навіть просто об'єктом типу Double. У цьому випадку проблему накопичення інформації можна було б покласти на генератор та обслуговуючий пристрій.

3.4.1.4 **Абстракція «черга транзакцій»**

Завдання цієї абстракції – накопичувати «транзакції», що чекають на обслуговування. Для програмної реалізації таких абстракцій у фреймворці Simulation існує клас process.QueueForTransaction.

3.4.1.5 **Абстракція «накопичувач статистичної інформації про розміри черги»**

Ця абстракція потрібна для вирішення завдання щодо накопичення статистичної інформації про довжину черги. Клас stat.DiscretHisto фреймворку Simulation забезпечує вирішення такого завдання. Достатньо об'єкт цього класу підключити до абстракції «черга транзакцій».

3.4.1.6 **Абстракція «накопичувач статистичної інформації про час перебування транзакції у черзі »**

Ця абстракція потрібна для вирішення завдання щодо накопичення статистичної інформації про час перебування транзакції у черзі. Клас stat.Histo фреймворку Simulation забезпечує накопичення цієї інформації. Час перебування у черзі буде визначати сама транзакція шляхом обчислення різниці між моментами часу початку обслуговування та часом появи транзакції у системі.

3.4.1.7 **Абстракція «накопичувач статистичної інформації про час перебування транзакції у системі»**

Ця абстракція потрібна для вирішення завдання щодо накопичення статистичної інформації про час перебування транзакції у системі. Клас stat.Histo фреймворку Simulation забезпечує вирішення такого завдання. Час перебування транзакції у системі буде визначати сама транзакція шляхом обчислення різниці між часом завершення обслуговування та часом появи транзакції у системі.

3.4.1.8 **Абстракція «накопичувач статистичної інформації про час простою обслуговуючого пристрою»**

Ця абстракція потрібна для вирішення завдання щодо накопичення статистичної інформації про час простою обслуговуючого пристрою. Клас stat.Histo фреймворку Simulation забезпечує вирішення такого завдання. Час простою об’єктів типу Actor у фреймворці фіксується автоматично, якщо об’єктові передано посилання на об’єкт типу stat.Histo.

3.4.1.9 **Результати аналізу системи**

Результати аналізу системи наведено у таблиці 3.1

Таблиця 3.1 – Абстракції системи, що входять до складу моделі

Абстракція	Перелік завдань	Клас
generator	Створювати через випадкові інтервали часу транзакції і додавати їх до черги.	Generator (Actor)
device	Забирати транзакції з черги і імітувати їх обробку затримкою у часі. У разі відсутності транзакцій, чекати на їх появу. Визначати час перебування транзакції у системі.	Device (Actor)
transaction	Моделює сутність, що потребує обслуговування. Містить дані про час появи у системі.	Transaction (Actor)
queue	Зберігати транзакції, що чекають на обслуговування. Взаємодіяти з накопичувачем DiscretHisto та діаграмою.	QueueFor-Transaction
discretHisto-ForQueue	Накопичувати інформацію про розмір черги.	Discret-Histo
histoTrans-actionWaitIn Queue	Накопичувати інформацію про час перебування у черзі	Histo
histoTrans-actionService	Накопичувати інформацію про час перебування у системі.	Histo
histoDevice-WaitTime	Накопичувати інформацію про час простою обслуговуючого пристрою.	Histo

До складу абстракцій додаємо також таку абстракція, як модель системи у цілому, що об’єднує інші абстракції і є об’єктом експериментального дослідження під час моделювання.

3.4.2 *Реалізація шару подання*

Основою шару подання є інтерфейс користувача, який представлено на рисунках 3.1 – 3.2. Інтерфейс був створений відповідно до завдань, що були визначені вище. Інтерфейс спроектовано як сукупність декількох основних панелей.

Основою інтерфейсу є компонент JSplitPanel.

Ліву частину цього компоненту займає панель для розміщення елементів налаштування моделі і присутня на екрані у всіх режимах роботи. У якості менеджера компоновки цієї панелі вибрано GridLayout.

Праворуч розташований компонент JTabbedPane, на закладках якого розташовані панелі JPanel, що з’являються після вибору відповідного режиму роботи.

3.4.3 *Режим тестування моделі*

Закладка «Test», рисунок 3.2, використовується для тестування роботи моделі із динамічною індикацією зміни розмірів черг у часі та виведенням протоколу роботи моделі.

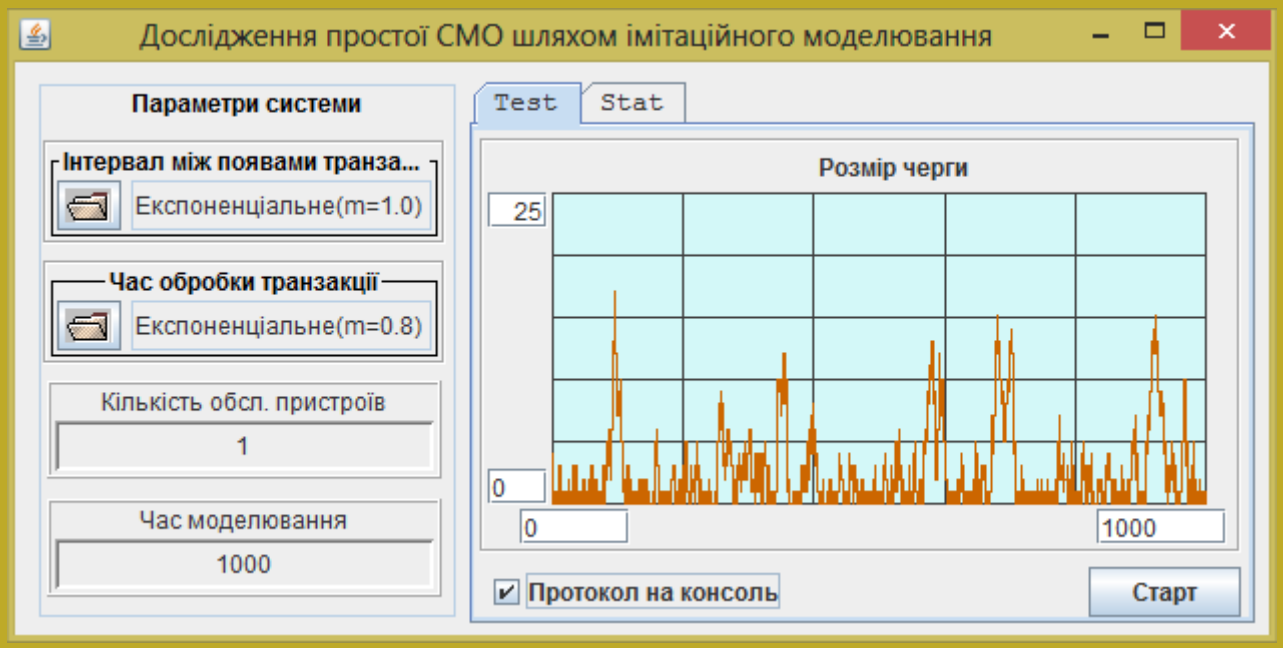

Рисунок 3.2 – Інтерфейс користувача моделі у режимі тестування

Для панелі застосовано менеджер компоновки GridBagLayout, що надає багато можливостей для розташування компонентів на панелі та дозволяє змінювати розміри панелі не порушуючи її дизайн.

Для того, щоб налаштування діаграм відповідали налаштуванням моделі, подія caretUpdate компоненту, що використовується для налаштування часу

моделювання, пов’язані з налаштуваннями параметру діаграми horisontalMaxText. Але цей зв’язок має підтримуватися тільки у тому випадку, якщо відкрита панель «Test».

А для того, щоб відновити цей зв’язок після відкриття панелі «Test» використовується подія цієї панелі componentShown.

Код методів, що реалізують ці зв’язки, наведено у лістингу 3.1.

Лістинг 3.1 - Методи оновлення налаштувань діаграми

```
protected void onModelingTimeCaretUpdate() {
    if (getJPanelTest().isShowing()) {
        getDiagramQueueSize().setHorizontalMaxText(
                chooseDataFinishTime.getText());
    }
}
protected void onPanelTestComponentShown() {
    // Штучно формуємо подію CaretUpdate,
    // щоб обновити налаштування діаграми
    getChooseDataFinishTime().select(0,0);
}
```

Для запуску моделі у режимі тестування використовується кнопка «Старт», з якою пов’язано виклик методу startTest().

Текст методу наведено у лістингу 3.2.

Лістинг 3. 2 - Метод запуску процесу моделювання у режимі тестування

```
private void startTest() {
    //Готуємо діаграму для виведення графіку
    getDiagramQueue().clear();
    //Створюємо диспетчера
    Dispatcher dispatcher = new Dispatcher();
    //Створюємо модель за допомогою фабрики
    IModelFactory factory = (d)-> new Model(d, this);
    Model model =(Model) factory.createModel(dispatcher);
    // Робимо кнопку «Старт» недосяжною на період роботи моделі
    getJButtonStart().setEnabled(false);
    dispatcher.addDispatcherFinishListener(
            ()->getJButtonStart().setEnabled(true));
    //Готуємо модель до роботи у режимі тестування
    model.initForTest();
    //Запускаємо модель
    dispatcher.start();
}
```

3.4.4 *Публічний програмний інтерфейс шару подання*

Ще одна важлива функція шару подання – надання доступу до своїх компонент іншим класам. Перелік відповідних публічних методів наведено на рисунку 3.3.

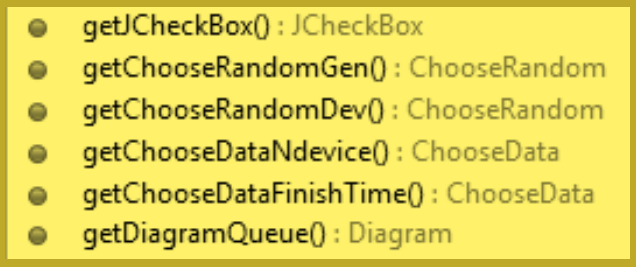

Рисунок 3.3 – Публічний інтерфейс шару подання

3.5 Реалізація шару моделі

Цей шар складається тільки з класу моделі.

3.5.1 *Клас Model*

Клас Model побудовано виходячи з того, що модель буде створюватися перед кожним її запуском. Це значно спрощує програмування і підвищує його надійність, тому що у цьому випадку усі компоненти моделі теж створюються заново і не потребують ініціалізації.

Перелік атрибутів моделі наведено у лістингу 3.3.

Зверніть увагу, об’єкти типу Transaction у моделі не створюються. Їх буде створювати генератор транзакцій у процесі своєї роботи.

Лістинг 3.3 - Перелік атрибутів моделі

```
//Посилання на диспетчера
private Dispatcher dispatcher;
//Посилання на візуальну частину
private GUI gui;

////////Актори\\\\\\\\\
// Генератор транзакцій
private Generator generator;
// Обслуговуючий прилад
private Device device;
//Бригада обслуговуючих пристроїв
private MultiActor multiDevice;

////////Черги\\\\\\\\\
// Черга транзакцій
private QueueForTransactions<Transaction> queue;

////////Гістограми\\\\\\\\\\\\
// Гістограма для довжини черги
private DiscretHisto discretHistoQueue;
// Гістограма для часу перебування у черзі
private Histo histoTransactionWaitInQueue;
// Гістограма для часу обслуговування
private Histo histoTransactionServiceTime;
// Гістограма для часу чекання Device
private Histo histoWaitDevice;
```

Для створення моделі використовується конструктор з двома параметрами, який забезпечує гарантовану передачу моделі посилань на візуальну частину і диспетчера. Решта компонентів створюється безпосередньо у моделі. Конструктор наведено у лістингу 3.4.

Конструктор забезпечує також передачу акторів моделі до стартового списку диспетчера за допомогою методу componentsToStartList().

Лістинг 3.4 - Конструктор моделі

```
public Model(Dispatcher d, GUI g) {
        if (d == null || g == null) {
            System.out.println("Не визначено диспетчера або GUI для Model");
            System.out.println("Подальша робота неможлива");
            System.exit(0);
        }
        dispatcher = d;
        gui = g;
        //Передаємо акторів до стартового списку диспетчера
        componentsToStartList();
    }
```

Такий конструктор можна вважати стандартним для виконання РГР, за виключенням назви класу.

Натомість, вміст методу componentsToStartList() залежіть від конкретного завдання. У нашому випадку цей метод має вигляд, представлений у лістингу 3.5.

Лістинг 3.5 - Метод componentsToStartList()

```
    public void componentsToStartList() {
        // Передаємо акторів диспетчеру
        dispatcher.addStartingActor(getGenerator());
        dispatcher.addStartingActor(getMultiDevice());
    }
```

Слід звернути увагу на те, що для звертання до акторів використовуються методи get…(), у яких реалізовано відкладене створення об’єктів.

Як приклад методу для створення актора наведемо метод getGenerator(), лістинг 3.6. Для створення усіх акторів ми будемо використовувати однаковий підхід, що полягає у використанні конструктора з параметрами, які передають посилання на візуальну частину та на модель. Маючи ці посилання, актор може отримати доступ до інформації, що потрібна йому для функціонування.

Лістинг 3.6 - Метод відкладеного створення об’єкту generator

```
    public Generator getGenerator() {
        if (generator == null) {
            generator = new Generator("Generator", gui, this);
        }
        return generator;
    }
```

Якщо ми бажаємо накопичувати інформацію про час перебування у стані очікування якогось актора, то метод get… буде виглядати трохи інакше, так як виглядає метод відкладеного створення об'єкту device, лістинг 3.7.

Лістинг 3.7 - Метод відкладеного створення об'єкту device

```
public Device getDevice() {
    if (device == null) {
        device = new Device("Device", gui, this);
        device.setHistoForActorWaitingTime(getHistoWaitDevice());
    }
    return device;
}
```

Об'єкти класу MultiActor створюються дещо інакше. Такому об'єкту потрібно передати посилання на зразок, що буде клонуватися, та задати кількість клонів. Зразок доцільно створювати у вигляді іменованого об'єкту. Це спростить процедуру ініціалізації об'єкта, якщо така буде потрібна. Наводимо тут метод створення бригади обслуговуючих пристроїв, лістинг 3.8.

Лістинг 3.8 – Метод відкладеного створення бригади пристроїв

```
public MultiActor getMultiDevice() {
    if (multiDevice == null) {
        multiDevice = new MultiActor();
        multiDevice.setNameForProtocol("MultiActor для бригади пристроїв");
        multiDevice.setOriginal(getDevice());
        multiDevice.setNumberOfClones(gui.getChooseDataNdevice().getInt());
    }
    return multiDevice;
}
```

Об'єкти для черг створюються також інакше. Посилання, що необхідні чергам слід передавати або через конструктори з параметрами, або через методи set…(). Для прикладу наведемо метод getQueue (), лістинг 3.9.

Лістинг 3.9 – Метод відкладеного створення черги

```
public QueueForTransactions<Transaction> getQueue() {
    if (queue == null) {
        queue = new QueueForTransactions<>("Queue", dispatcher,
                getDiscretHistoQueue());
    }
    return queue;
}
```

Об'єкти для гістограм, зазвичай, не потребують додаткових налаштувань. Слід тільки розрізняти класи Histo та DiscretHisto. Як приклад, у лістингу 3.10 наведено метод доступу до гістограми для черги.

Лістинг 3.10 – Метод відкладеного створення гістограми

```
    public DiscretHisto getDiscretHistoQueue() {
    if (discretHistoQueue == null) {
        discretHistoQueue = new DiscretHisto();
    }
    return discretHistoQueue;
}
```

Завершується робота над класом моделі створенням методів ініціалізації моделі для можливих режимів роботи. Ці методи налаштовують модель до вимог конкретного режиму.

Так у методі initForTest(), лістинг 3.11, діаграмам передаються посилання на об'єкти класу Painter, що забезпечує відображення цих черг на гістограмах.

Лістинг 3.11 – Метод ініціалізації моделі у режимі тестування

```
public void initForTest() {
    // Передаємо чергам painter-ів для динамічної індикації
    getQueue().setPainter(gui.getDiagramQueue().getPainter());
    //Налаштовуємо можливість виведення протоколу на консоль
    if (gui.getJCheckBox().isSelected())
        dispatcher.setProtocolFileName("Console");
    else
        dispatcher.setProtocolFileName("");
}
```

У інших режимах виникає потреба реалізації методів відповідних інтерфейсів. Якщо методи ініціалізації з цих інтерфейсів передають якісь налаштування компонентам моделі, то ці налаштування доцільно передавати компонентам через спеціально для цього створені методи. Перенесення цих налаштувань до шару подання не зовсім коректне, бо компоненти їх можуть не отримати.

3.6 Порядок виконання роботи

3.6.1 *Підготовка до роботи*

– Завантажте проекти, що знаходяться в архіві SimulationAllLab.zip до Eclipse скориставшись функцією меню Import→Existing Project into Workspace→ Select archive file.

3.6.2 *Знайомство з класами, які моделюють компоненти СМО*

Відкрийте проект Simulation2018.

– Ознайомтеся з класом process.QueueForTransactions.
– Ознайомтеся з класом process.Store.
– Ознайомтеся з класом process.MultiActor.

3.6.3 *Знайомство з прикладом побудови моделі СМО*

– Ознайомтеся з класами пакету testModel проекту SimulationAllLab. Саме цей проект розглядається як приклад у методичних вказівках до лабораторної роботи. Активізуйте клас GUI і перевірте працездатність проекту.

3.6.4 *Знайомство з прикладом виконання РГР*

– Ознайомтеся з класами пакету buldo2018 проекту SimulationAllLab. Саме цей проект розглядається як приклад у методичних вказівках до РГР. Активізуйте клас BuldGUI і перевірте працездатність проекту.

– Почніть за зразком створювати модель для своєї РГР.

3.7 Завдання для самостійної роботи

З метою отримання навичок побудови імітаційної моделі, створіть модель системи відповідно до варіанту розрахунково-графічної роботи. Роботу виконує команда з трьох студентів.

Дизайнер перш за все розробляє перелік публічних методів (програмний інтерфейс) для шару подання і після цього створює візуальну частину проекту.

Team leader перш за все розробляє перелік публічних методів (програмний інтерфейс) для моделі, створює цей клас та займається узгодженням питань між членами команди і тестуванням проекту.

Кодер створює класи для складових частин моделі.

Кожен з членів команди готує звіт по своїй частині роботи. Об’єднує звіти керівник команди.

3.8 Зміст звіту

– Звітом по лабораторній роботі є частина звіту з РГР, що стосується візуальної частини та моделі.

3.9 Контрольні питання

– Компоненти фреймворку для накопичення статистичних даних.

– Компоненти фреймворку для візуалізації роботи моделі та накопичення статистичних даних.

– Опис класу QueueForTransactions, його змінні й методи.

– Як створюється подія, пов'язана з переповненням черги і як її може бути оброблено.

– Клас Store.

– Опис класу MultiActor. Особливості клонування об'єктів в Java.

– Структура застосування для моделювання СМО.

– Характеристика шару подання.

– Характеристика шару моделі.

– Характеристика шару компонентів.

4 ЛАБОРАТОРНА РОБОТА №4. ЗАСОБИ ІМІТАЦІЙНОГО МОДЕЛЮВАННЯ ПАРАЛЕЛЬНИХ ПРОЦЕСІВ

Мета роботи:

– Знайомство з класами та методами, які забезпечують організацію динамічної взаємодії псевдопаралельних процесів.

– Знайомство з прикладами опису правил дії активних компонентів.

4.1 Короткі теоретичні відомості

4.1.1 *Особливості моделювання паралельних процесів*

Розглянемо як приклад, поведінку обслуговуючого пристрою з прикладу, що розглядався у попередній лабораторній роботі.

Правила дії цього об’єкту полягають у багаторазовому повторенні наступних операцій

– Очікування на появу транзакції у черзі.

– Вилучення транзакції з черги.

– Затримка у часі , що імітує обробку транзакції.

– Перехід до першого пункту.

Аналізуючи правила дії, що описані вище, неважко побачити важливу особливість цього алгоритму – періодичні зупинки виконання коду, або на якийсь час, або до виконання умови.

Друга особливість цього алгоритму полягає у тому, що він має виконуватися одночасно з правилами дії генератора транзакцій, які мають також затримки у часі.

Правила дії мають бути перетворені у деякий код, що виконується комп'ютером. Для того щоб реалізувати паралельне виконання коду та забезпечити зупинки у процесі виконання коду, правила дій об’єктів можна представити у вигляді потоків.

Мова програмування Java надає багато можливостей для програмної реалізації розглянутих алгоритмів. Один із шляхів, який, можливо, не є найпростішим, полягає у моделюванні в реальному часі. У цьому випадку затримки у часі можна забезпечити за допомогою метода sleep(). Але недолік такого способу полягає в тому, що імітаційна модель буде працювати з великою кількістю зупинок, що призведе до уповільнення роботи програми. Тривалість зупинок можна зменшити за рахунок зміни масштабу часу, але у цьому випадку затримки, що імітують виконання завдань, будуть близькими за величиною до часу виконання коду, що призведе до великих похибок.

Інший спосіб полягає в тому, що затримки створюються в деякому віртуальному часі, який змінюється дискретно від події до події, і ніяк не пов'язаний з реальним часом, і внаслідок цього не залежить від часу виконання коду. Але для реалізації такого способу необхідно мати об’єкт, який буде керувати цим віртуальним часом. Назвемо цей об’єкт диспетчером. Диспетчер визначає час на підставі списку майбутніх подій. Цей список уявляє собою

колекцію, у якій знаходяться посилання на об'єкти, що затримані на деякий час. Кожен з об’єктів у цьому списку знає час відновлення своїх правил дії, і список впорядковано саме за цим атрибутом об’єкту. Об’єкти самі заносять себе до цього списку перед тим як призупинитися.

Диспетчер активізується тоді, коли припиняються правила дії об'єктів, якими він керує. Диспетчер вилучає з управляючого списку об’єкт з найменшим часом відновлення правил дії і присвоює віртуальному часу значення, що відповідає запланованому моменту відновлення правил дії вилученого об’єкту. Тобто диспетчер, побачивши яким повинен бути час, таким його і встановлює. Виходить, що об’єкти управляють часом, а не навпаки, і час змінюється дискретно від події до події. Змінивши час, диспетчер відновлює правила дії об’єкту, а сам призупиняється.

Окрім списку майбутніх подій диспетчер має ще список об’єктів, правила дії яких зупинені до виконання деяких умов. Так само як і у попередньому випадку, об’єкти знають, чого чекають і самі заносять себе до цього списку перед зупинкою. Диспетчер переглядає цей список кожного разу перед зміною модельного часу, і якщо якась умова виконується, то правила дії відповідного об’єкту поновлюються без зміни часу.

4.1.2 *Особливості програмування імітаційних моделей*

Перша особливість полягає у тому, що для виконання правил дії паралельно працюючих об’єктів ми створюємо потоки.

Базовим класом потоків у мові Java є клас Thread. Але потік виконання правил дії об’єкту є тільки складовою частиною об’єкту. Диспетчер і об’єкти, що діють паралельно, можуть існувати і до створення потоку, і після того, як потік завершився. Тому створювати відповідні класи ми будемо не шляхом наслідування класу Thread, а шляхом реалізації інтерфейсу Runnable у цих класах. Як відомо, цей інтерфейс передбачає реалізацію публічного методу run(), який і є точкою входу до потоку, що створюється.

Для того, щоб створити новий потік, клас, який реалізує Runnable, повинен виконати у одному із своїх методів код, що представлений у лістингу 5.1.

Лістинг 4.1 – Створення потоку для об’єкту, що реалізує інтерфейс Runnable

```
…
      Thread  thread = new Thread(this);
      thread.start();
…
```

Щоб призупинити виконання правил дії об’єктів, або диспетчера будемо використовувати метод wait() класу Object. Для відновлення виконання правил дії цих об’єктів будемо використовувати метод notify() класу Object. Слід нагадати, що методи wait() і notify() класу Object не можна використовувати прямо, викликаючи їх для потоків, якими ми хочемо керувати. Це можна робити за допомогою деякого іншого, синхронізованого об'єкта з публічним доступом, що називають монітором. Саме цей об’єкт приймає повідомлення

wait() і notify(). Для породження таких об'єктів і забезпечення їх функціонування у фреймворці Simulation створено клас process.BooleanSemaphore.

Особливість використання потоків при програмуванні імітаційних моделей полягає також і у тому, що під час роботи моделі не існує більше одного активного потоку, пов’язаного з роботою моделі. З цієї причини потоки, що імітують паралельне виконання правил дії об’єктів, називають псевдопаралельними. Це має місце внаслідок того, що диспетчер, породжуючи або активізуючи потік виконання правил дії якогось об’єкту, сам призупиняється доти, поки не призупиниться активізований потік. Таким чином виконується або потік якогось з об’єктів, або потік диспетчера. Завдяки цьому підтримується сувора послідовність подій у віртуальному (модельному) часі.

Внаслідок псевдопаралельного виконання потоків у імітаційних моделей практично не існує проблеми використання спільних ресурсів.

Ще одна проблема програмування імітаційних моделей полягає в тому, що об'єкти, які чекають виконання деяких умов, повинні ці умови пам'ятати. Причому, таких умов у одного об'єкта може бути декілька. У Java ми можемо оперувати тільки об'єктами, тому умова може бути створена тільки як метод об'єкта деякого класу. Для створення таких об'єктів доцільно використовувати функціональний інтерфейс BooleanSupplier, який потребує реалізації методу getAsBoolean(), що забезпечує перевірку необхідної умови. Для спрощення коду об’єкт цього типу доцільно створювати за допомогою механізму лямбда функцій.

Таким чином, якщо виникає потреба у передачі або зберіганні деякої умови, то за допомогою лямбда функції, яка забезпечує перевірку цієї умови, створюється об’єкт типу BooleanSupplier. Цей об’єкт може бути і анонімним.

4.1.3 *Класи, що забезпечують динамічну взаємодію псевдопаралельних процесів*

Класи фреймворку Simulation, що забезпечують динамічну взаємодію псевдопаралельних процесів об'єднані у пакеті process.

Найважливіші з них, що необхідні для розуміння особливостей роботи імітаційних моделей розглядаються нижче.

Клас BooleanSemaphore використовується для створення об’єктів, що дозволяють призупиняти і відновлювати потоки виконання правил дії об’єктів.

Клас Actor визначає найбільш загальні властивості й особливості поведінки об'єктів, які повинні виконувати свої правила дії в часі.

Клас Dispatcher описує поведінку і властивості об'єкта, що забезпечує синхронізацію виконання правил дії компонент моделі, зміну модельного часу й формування протоколу роботи моделі.

4.1.3.1 **Клас BooleanSemaphore**

Об'єкти цього класу використовуються для керування потоками. Клас має всього одне приватне поле логічного типу з ім'ям value, що може бути використане як індикатор стану потоку.

Для затримки потоків за допомогою об'єктів даного класу використовується метод waitForValue(boolean), наведений у лістингу 4.2.

Лістинг 4.2 – Метод waitForValue(boolean)

```
public synchronized void waitForValue(boolean state) {
   while (value != state) {
      try    {
                  wait();
            } catch (java.lang.InterruptedException e) {
                  e.printStackTrace();
      }
   }
}
```

Цей метод використовує стандартний метод wait(), який забезпечує затримку потоку доти, доки індикатор не перейде в необхідний стан.

Для відновлення роботи призупиненого потоку використовується метод setValue(boolean).

Лістинг 4.3 – Метод setValue(boolean)

```
   public synchronized void setValue(boolean newValue) {
      if (newValue != value) {
         value = newValue;
         notify();
      }
   }
```

Наведений метод не тільки змінює стан індикатора, але й сповіщає про це зацікавлений потік за допомогою стандартного повідомлення notify(), що приводить до поновлення виконання правил дії потоку, який чекав зміни стану індикатора.

4.1.3.2 **Клас Actor**

Цей абстрактний клас містить найбільш загальні властивості й методи об'єктів, які мають правила дії, розподілені в часі.

Клас реалізує два інтерфейси.

Інтерфейс Runnable дає можливість виконувати правила дії об'єктів як потоки. Відповідно до вимог цього інтерфейсу в класі реалізований метод run(), що забезпечує запуск правил дії об'єктів підкласів.

Інтерфейс Clonable дає можливість клонувати об'єкти класу або його підкласів. Відповідно до вимог цього інтерфейсу в класі реалізований метод clone(). Особливості клонування об'єктів в Java розглядалися вище, при розгляді класу MultyActor.

Поля об'єктів класу Actor та їх призначення

Поле activateCondition зберігає посилання на об'єкт типу BooleanSupplier, що у вигляді методу getAsBoolean() містить умову, виконання якої чекає призупинений об'єкт.

Поле activateTime, типу double, містить час поновлення виконання правил дії, призупинених на якийсь час.

Поле dispatcher містить посилання на об'єкт класу Dispatcher, під керуванням якого будуть виконуватися правила дії об'єкта.

Поле nameForProtocol містить ім'я об'єкту, що використовується у протоколі роботи моделі.

Поле suspendIndicator містить посилання на об'єкт класу BooleanSemaphore і використовується для призупинення та відновлення правил дії.

Поле waitingTimeHisto призначено для зберігання посилання на об'єкт класу Histo. Якщо таке посилання передано актору, то у відповідній гістограмі буде накопичуватися інформація про час чекання актора.

Для доступу до полів є методи, названі у відповідності зі стандартними правилами.

Методи, що забезпечують створення потоку виконання правил дії

Створення та запуск потоку виконання правил дії для об'єктів класу Actor виконується за допомогою методу start(), що наведений у лістингу 4.4. Новий потік створюється за допомогою конструктора класу Thread, куди як параметр передається посилання на «актора», чиї правила дії повинні виконуватися в цьому потоці. Потоку привласнюється ім'я «актора». Цей метод призначений для використання тільки об’єктом класу Dispatcher, тому не має публічного доступу.

Лістинг 4.4 – Текст методу start() класу Actor

```
final void start() {
    // Створюємо і запускаємо потік виконання правил дії актора
    Thread thread = new Thread(this, getNameForProtocol());
    thread.start();
}
```

Публічний метод з ім'ям run(), є точкою входу потоку виконання правил дії «актора». Метод реалізований у класі Actor відповідно до вимог інтерфейсу Runnable. Цей метод, наведений у лістингу 4.5.

Лістинг 4.5 – Текст методу run() класу Actor

```
public void run() {
    dispatcher.printToProtocol("  " + nameForProtocol + " стартує");
    try {
        rule();
    } catch (DispatcherFinishException e) {
        dispatcher.printToProtocol("  " + nameForProtocol
            + " не дочекався, бо диспетчер закінчив роботу.");
        return;
    } finally {
        dispatcher.printToProtocol("  " + nameForProtocol + " роботу завершив");
        // Переводимо індикатор у стан "Актора призупинено" і таким чином
```

```
            // повідомляємо "Диспетчера", що виконання правил дії даного "актора"
            // завершено, що дає можливість "Диспетчеру" продовжити роботу
            suspendIndicator.setValue(true);
        }
    }
```

У методі run() викликаються правила дії об'єкта шляхом звертання до методу rule(). Метод rule() у класі Actor є абстрактним. Передбачається, що в підкласах цей метод буде реалізований відповідно до правил дії об’єктів підкласів.

Перед викликом правил дії метод run() виводить до протоколу диспетчера повідомлення про початок роботи об'єкта. Після завершення виконання правил дії метод виводить до протоколу повідомлення про те, що об'єкт роботу закінчив.

У методі також обробляється виключна ситуація DispatcherFinishException, яка створюється об’єктами, які не дочекалися виконання умови, коли диспетчер завершив роботу.

Закінчується метод run() викликом методу, що встановлює індикатору стану потоку значення, яке сигналізує про те, що виконання потоку завершено. Цей сигнал дозволяє активізувати роботу диспетчера.

Методи, що забезпечують призупинення правил дії об'єктів

Для об'єктів класу Actor передбачено три види зупинок.

Перший вид зупинки – це зупинка на якийсь час (мається на увазі віртуальний модельний час), що за звичай імітує роботу об'єкта, на виконання якої потрібен певний час.

Затримка реалізується за допомогою методу holdForTime(double), текст якого наведено у лістингу 4.6.

У метод передається параметр holdTime, значення якого дорівнює необхідному часу затримки.

Лістинг 4.6 – Текст методу holdForTime(double) класу Actor

```
    protected final void holdForTime(double holdTime) {
        // Затримка не має сенсу, якщо диспетчер закінчив роботу.
        if (!dispatcher.isActiv()) {
            return;
        }
        // Обчислюємо час відновлення правил дії "актора"
        activateTime = dispatcher.getCurrentTime() + holdTime;
        // Заносимо посилання на "актора"
        // у список акторів, що затримані на деякий час.
        dispatcher.getTimingActorQueue().add(this);
        dispatcher.printToProtocol("  " + getNameForProtocol()
                + " затриманий до " + activateTime);
        // Переводимо індикатор у стан "Актора призупинено",
        // внаслідок чого "Диспетчер" може продовжити  работу.
```

```
        suspendIndicator.setValue(true);
        // Призупиняємо потік виконання правил дії "актора"
        suspendIndicator.waitForValue(false);
        // Тут актор колись відновить виконання правил дії
        dispatcher.printToProtocol("  " + getNameForProtocol()
                + " активізувався ");
    }
```

При виконанні методу, визначається час майбутньої активізації об'єкта – activateTime. Значення цього поля встановлюється рівним сумі поточного значення модельного часу і триалості затримки holdTime.

Після цього посилання на об'єкт додається до списку майбутніх подій диспетчера.

Виклик методу suspendIndicator.setValue(true) приводить до того, що об'єкти, що чекають зміни стану об’єкту, можуть відновити роботу. Найчастіше таким об'єктом є диспетчер.

Виклик методу suspendIndicator.waitForValue(false) приводить до призупинення правил дії об'єкта.

Другий вид зупинки виконання правил дії – це затримка до виконання деякої умови. Така затримка реалізується за допомогою методу waitForCondition(BooleanSupplier, String).

Першим параметром методу є об'єкт інтерфейсного типу BooleanSupplier. Цей інтерфейс передбачає реалізацію методу getAsBoolean(), що забезпечує перевірку необхідної умови. Таким чином, у метод waitForCondition() передається не сама умова (що в Java неможливо), а об'єкт класу, через який можна викликати метод перевірки цієї умови.

Другим параметром методу є рядок символів, що використовується для ідентифікації умови у протоколі.

Лістинг 4.7 – Метод waitForCondition(BooleanSupplier, String) класу Actor

```
    protected final void waitForCondition(BooleanSupplier c, String s)
            throws DispatcherFinishException {
        // Якщо умова виконується, затримка не потрібна
        if (c. getAsBoolean() )
            return;
        // Якщо диспетчер закінчив роботу, затримка не має сенсу.
        if (!getDispatcher().getThread().isAlive()) {
            return;
        }
        // Зберігаємо об’єкт, що містить умову.
        activateCondition = c;
        // Передаємо "актора" до списку акторів
        // що затримані до виконання умови
        getDispatcher().getWaitingActorQueue().add(this);
        getDispatcher().printToProtocol(
                "  " + getNameForProtocol() + " чекає "'+ s + """);
        //Запам'ятовуємо час, коли почалося чекання
```

```
        double stopTime=dispatcher.getCurrentTime();
        // Переводимо індикатор у стан "Актора призупинено",
        // надаючи  "Диспетчеру" можливість працювати
        suspendIndicator.setValue(true);
        // Переводимо потік виконання правил дії "актора"
        // у стан чекання.
        // Коли умова виконується,
        // диспетчер перемикає індикатор у стан false
        suspendIndicator.waitForValue(false);
        if(waitingTimeHisto!=null)
            //Обчислюємо і передаємо до гістограми час чекання
            waitingTimeHisto.add(dispatcher.getCurrentTime()-stopTime);
        if (activateCondition. getAsBoolean())
            getDispatcher().printToProtocol(
                    "  " + getNameForProtocol() + " дочекався '"+ s + "'");
        else
            throw new DispatcherFinishException();
    }
```

При виконанні методу waitForCondition() перш за все буде проводитись перевірка умови за допомогою виклику методу с. getAsBoolean(). Якщо умова виконується, то робота методу закінчується. Якщо ж умова не виконується, то посилання на об'єкт типу BooleanSupplier заноситься в поле activateCondition «актора», а посилання на самого «актора» передається диспетчерові.

Виклик методу suspendIndicator.setValue(true) приводить до того, що об'єкти, що чекають цього перемикання, можуть відновити роботу. Частіше за все таким об'єктом є диспетчер.

Виклик методу suspendIndicator.waitForValue(false) приводить до припинення правил дії об'єкта.

Особливість виконання цього методу полягає у тому, що умова, на виконання якої очікує актор, можливо не буде виконана до закінчення роботи диспетчера. У цьому випадку диспетчер перед закінченням роботи примусово активізує потік незважаючи на невиконання умови. Для того, щоб цей випадок можна було б належним чином обробити, метод викликає виключну ситуацію DispatcherFinishException, яка має бути опрацьована у правилах дії об’єкту. Фактично вона полягає у припиненні правил дії (return).

Третій вид зупинки виконання правил дії – це затримка до виконання деякої умови але не довше, ніж на визначений час. Така затримка реалізується за допомогою методу waitForConditionOrHoldForTime(BooleanSupplier, String, double), або методу holdForTimeOrWaitForCondition (double, BooleanSupplier, String,). Методи дуже схожі на обидва попередні методи. Різниця полягає тільки у тому, що посилання на актора розміщується у обох списках диспетчера і тому продовження правил дії відновляється або після виконання умови, або у призначений час. Наводити текст цього методу тут не будемо

4.1.3.3 **Клас Dispatcher**

Головне призначення цього класу – керувати модельним часом і забезпечити синхронізацію виконання правил дії «акторів». Крім того диспетчер формує протокол роботи «акторів», що істотно спрощує налагодження застосувань.

Поля об'єктів цього класу мають таке призначення:

currentTime – поточне значення модельного часу.

startList – список об'єктів, правила дії яких диспетчер повинен активізувати. На початку роботи цей список повинен містити посилання хоча б на один об'єкт, яким повинен керувати диспетчер. Поповнення цього списку можливо за допомогою методу addStartingActor(Actor).

timingActorQueue – список майбутніх подій, де перебувають посилання на об'єкти, правила дії яких припинені на деякий час. На початку цього списку має бути актор з найменшим часом поновлення правил дії. Це забезпечується об’єктом типу Comparator, який порівнює час активізації акторів.

waitingActorQueue – список посилань на об'єкти, у яких правила дії припинені до виконання деяких умов. Об'єкти в цьому списку впорядковані по правилу FIFO.

protocolFileName – рядок, що визначає ім'я файлу, куди буде виводиться протокол роботи моделі. Якщо ця змінна має значення “Console”, протокол виводиться на консоль. Якщо значення цієї змінної пустий рядок, протокол не виводиться.

thread – посилання на потік виконання правил дії диспетчера.

active – поле логічного типу, що має значення true, поки диспетчер працює. Перед закінченням роботи «диспетчер» установлює значення поля рівним false.

dispatcherStartListeners, changeTimeListeners, dispatcherFinishListeners – це колекції, що зберігають посилання на слухачів подій, які генерує диспетчер під час своєї роботи. Подія dispatcherStartEvent сповіщає слухачів про початок роботи диспетчера. Подія changeTimeEvent сповіщає слухачів про зміну модельного часу. Подія dispatcherFinishEvent сповіщає слухачів про завершення роботи диспетчера.

Робота диспетчера

Диспетчер, так само як і «актори», має свої правила дії. Ці правила дії виконуються в окремому потоці, що вимагає реалізації інтерфейсу Runnable у класі Dispatcher і наявності в ньому методу run.

Запуск диспетчера відбувається шляхом виклику публічного методу start() класу Dispatcher. Виклик цього методу означає початок процесу моделювання. Текст методу наведений у лістингу 4.9.

Лістинг 4.9 – Метод start() класу Dispatcher

```
public void start() {
    if (protocolFileName != null && protocolFileName != "Console"
        && protocolFileName.trim() != "") {
```

```
            FileWriter file;
            try {
                file = new FileWriter(protocolFileName, false);
                file.close();
            } catch (java.io.IOException e) {
                System.out.println("Не вийшло відкрити файл протоколу");
            }
        }
        this.thread= new Thread(this, "Dispatcher");;
        this.thread.start();
    }
```

У цьому методі спочатку, при необхідності, створюється файл для виводу протоколу роботи моделі. Далі створюється потік виконання правил дії диспетчера й запускається за допомогою методу start() потоку, який ініціює виконання методу run() диспетчера. Код методу run() наведений у лістингу 4.10

Лістинг 4.10 – Правила дії об'єкта класу Dispatcher

```
public void run() {
     activ = true;
    currentTime = 0;
     printToProtocol("Час " + currentTime);
    Actor readyActor=null;
     while (true) {
        runStartList();
        readyActor = testWaitingQueue();
        if (readyActor == null) {
            if (timingActorQueue.isEmpty()) {
                break;
            }
            readyActor = timingActorQueue.remove();
        setCurrentTime(readyActor.getActivateTime());
        }
        readyActor.getSuspendIndicator().setValue(false);
        readyActor.getSuspendIndicator().waitForValue(true);
    }
    activ = false;
    releaseWaitingQueue();
    fireDispatcherFinishEvent(new DispatcherFinishEvent(this));
}
```

Робота диспетчера починається з ініціалізації полів activ та currentTime. Далі виконується циклу правил дії «диспетчера».

Починається цей цикл викликом методу runStartList(), що створює потоки виконання правил дії «акторів», занесених у стартовий список. Текст цього методу наведений у лістингу 4.11.

Лістинг 4.11 – Метод runStartList() класу Dispatcher

```
private void runStartList() {
    while (!startList.isEmpty()) {
        Actor a = startList.remove(0);
        a.getSuspendIndicator().setValue(false);
        a.start();
        a.getSuspendIndicator().waitForValue(true);
    }
}
```

Правила дії «акторів», що стартують, виконуються по черзі. Новий потік стартує тільки після того, як буде призупинений попередній. Такий порядок забезпечує об’єкт класу BooleanSemaphore, який знаходиться у полі suspendIndicator класу Actor. Коли правила дії чергового «актора» припиняються, цикл активізації акторів продовжується і стартує новий актор. Виконання методу закінчується тільки тоді, коли правила дії останнього «актора», що стартує, будуть призупинені.

Після виконання методу runStartList() посилання на «акторів» потрапляють або у список майбутніх подій timingActorQueue, або в список waitingActorQueue, де знаходяться об'єкти, що чекають виконання умов. На цьому завершується перший етапу циклу правил дії диспетчера.

Наступні етапи циклу робота «диспетчера» полягають у обробці списків timingActorQueue та waitingActorQueue.

Насамперед, аналізується список об'єктів, що чекають виконання умов. Для цього використовується метод testWaitingQueue().

Лістинг 4.12 – Метод testWaitingQueue() для обробки списку об'єктів, що чекають виконання умов

```
private Actor testWaitingQueue() {
    Iterator i = this.waitingActorQueue.iterator();
    while (i.hasNext()) {
        Actor a = i.next();
        if (a.activateCondition.getAsBoolean()) {
            waitingActorQueue.remove(a);
            timingActorQueue.remove(a);
            return a;
        }
    }
    return null;
}
```

Метод забезпечує послідовний перегляд списку «акторів», що чекають виконання умов, і перевірку очікуваної умови для кожного з них. Нагадаємо, що в полі activateCondition акторів зберігаються посилання на об'єкт класу, що реалізує метод getAsBoolean(), відповідно до вимог інтерфейсу BooleanSupplier. Саме метод getAsBoolean() перевіряє необхідну умову.

Якщо умова, що перевіряється, виконується для якогось «актора», то цей «актор» вилучається зі списку і повертається, як результат виконання методу. Посилання на цього актора вилучається також і із списку майбутніх подій, якщо воно там є. У випадку коли актор, для якого виконалась очікувана подія, не знайдений, метод повертає null. Результат виконання методу привласнюється змінній readyActor.

Наступний етап основного циклу роботи диспетчера виконується в тому випадку, якщо значення змінної readyActor дорівнює null. На цьому етапі диспетчер працює зі списком «акторів», правила дії яких призупинені на якийсь час. Якщо цей список порожній, то диспетчер закінчує свою роботу. Обробка не порожнього списку полягає в тім, що із списку вилучається «актор» з мінімальним часом активізації. Посилання на цей об'єкт записується в змінну readyActor Модельному часу currentTime привласнюється значення часу активізації об'єкта readyActor, що зберігається в змінній activateTime. Тобто диспетчер, побачивши, яким повинен бути час найближчої події, таким його й встановлює. Для зміни модельного часу викликається метод setCurrentTime, що забезпечує вивід рядка протоколу з новим значенням часу. Крім того, у методі setCurrentTime створюється подія ChangeTimeEvent, яка може бути використана іншими об'єктами, наприклад, для індикації поточного часу або збору статистичної інформації про роботу об'єкта.

На завершальному етапі виконання циклу змінюється стан індикатора «актора» readyObject, внаслідок чого виконання правил дії «актора» може відновитися. А для того, щоб це дійсно відбулося, робота диспетчера припиняється за допомогою індикатора того «актора», чиї правила дії відновляються, шляхом виклику методу readyActor.getSuspendIndicator(). waitForValue(true).

Робота диспетчера буде відновлена тільки після того як правила дії активізованого «актора» призупиняться або закінчяться.

Цикл виконання правил дії диспетчера триває доти, поки список timingActorQueue не стане порожнім, що свідчить про зупинку модельного часу.

Після завершення циклу правил дії диспетчера виконуються операції, що завершують роботу диспетчера.

Насамперед, у поле active заноситься значення false. Після цього викликається метод releaseWaitingQueue().

Лістинг 4.13 – Метод releaseWaitingQueue(), що звільняє списки об'єктів, що чекають виконання умов

```
private void releaseWaitingQueue() {
      printToProtocol("  Диспетчер звільняє потоки,
                              що чекають виконання умови");
      while (waitingActorQueue.size()>0) {
         Actor a = waitingActorQueue.remove(0);
         a.getSuspendIndicator().setValue(false);
         //Будемо чекати, поки не завершиться виконання правил дії
```

```
            a.getSuspendIndicator().waitForValue(true);
        }
}
```

Цей метод послідовно активізує правила дії «акторів», які чекають виконання умов, але вже не дочекаються цього. Передбачається, що після виходу з режиму очікування «актор» завершить свої правила дії, обробивши виключну ситуацію DispatcherFinishExeption.

Такий «дивний» спосіб пов'язаний з тим, що в Java неможливо переривати виконання потоку зовні. Потік повинен сам завершувати свою роботу.

Після виконання завершальних операцій диспетчер формує подію DispatcherFinishEvent, що сповіщає всіх зацікавлених слухачів про завершення роботи диспетчера.

4.2 Побудова активних компонентів моделі з використанням розглянутих класів

Можливості використання розглянутих класів для імітаційного моделювання продемонструємо на прикладі реалізації шару компонентів моделі простої СМО, що розглядалася у попередній лабораторній роботі.

Шар подання та клас моделі було вже створено у попередній лабораторній роботі.

Тут ми створимо класи для акторів моделі та клас транзакції.

4.2.1 *Аналіз правил дії активних компонентів*

4.2.1.1 **Клас транзакції**

Транзакція має стати у чергу на обслуговування та фіксувати інтервали часу, коли вона перебувала у черзі та повний час обслуговування.

Перелік даних, що необхідні транзакції для вирішення цих завдань, наведено у таблиці 4.1.

Таблиця 4.1 – Атрибути абстракції «Транзакція»

Назва поля	Клас(Тип)	Призначення поля	Джерело
createTime	double	Час появи транзакції у системі	Dispather
serviceDone	boolean	Ознака завершення обслуговування	Device
queue	QueueForTransaction	Посилання на чергу транзакцій	Model
histoQueue	Histo	Гістограма часу перебування у черзі	Model
histoService	Histo	Гістограма для часу обслуговування	Model

Атрибут createTime потрібен для визначення часу перебування у системі та у черзі. Його об’єкт може отримати на початку роботи через метод getCurrentTime від диспетчера.

Ознака serviceDone потрібна для визначення моменту завершення процесу обслуговування. Значення true для цієї ознаки має встановити обслуговуючий пристрій після завершення обслуговування. Для цього має бути відповідний метод set… .

Посилання на гістограми та чергу об1єкт може отримати через посилання на модель, яке може бути передано через конструктор.

Правила дії транзакції схематично можна представити у вигляді діаграми діяльності, рисунок 4.1.

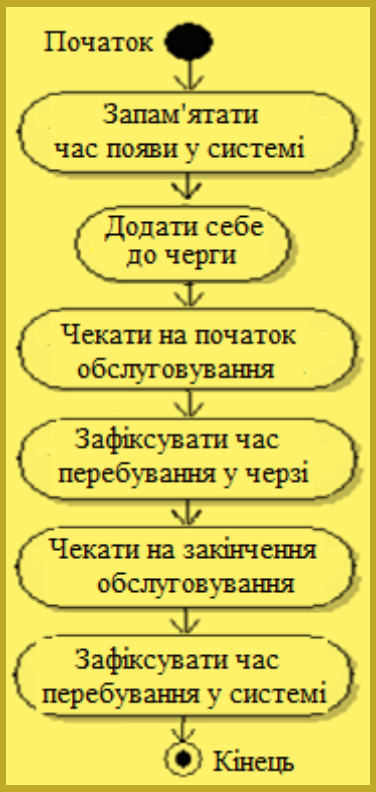

Рисунок 4.1 – Діаграма діяльності абстракції транзакція

4.2.1.2 **Генератор транзакцій**

Головне завдання цієї абстракції – через випадкові інтервали часу створювати транзакції та передавати їх диспетчеру. Свої дії об’єкт має виконувати впродовж усього часу моделювання.

Перелік даних, необхідних генератору, наведено у таблиці 4.2.

Таблиця 4.2 – Атрибути абстракції Генератор

Назва поля	Клас	Призначення поля	Джерело
model	Model	Посилання на модель системи	Model
rnd	Randomable	Генератор випадкових чисел	GUI
finishTime	double	Час моделювання	GUI

Посилання на модель системи необхідно зберігати, бо воно потрібно для подальшої передачі транзакціям, що будуть створюватися.

Атрибут rnd буде використовуватися для формування випадкової величини інтервалу часу між створеннями транзакцій. Посилання на цей

елемент можна отримати з візуальної частини.

Поле finishTime визначає час, впродовж якого має працювати генератор. Значення цього атрибуту можна також отримати з візуальної частини проекту.

Посилання на візуальну частину і на модель має бути передано через конструктор.

Правила дії генератора схематично можна представити у вигляді діаграми діяльності, рисунок 4.2.

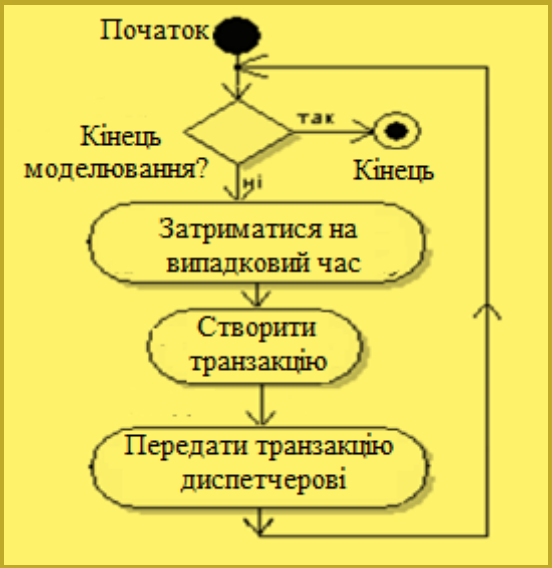

Рисунок 4.2 – Діаграма діяльності абстракції генератор

4.2.1.3 **Обслуговуючий пристрій**

Обслуговуючий пристрій працює із чергою транзакцій. Якщо черга пуста, обслуговуючий пристрій чекає на появу транзакції. Якщо транзакція є, пристрій забирає її з черги для обслуговування. Далі обслуговуючий пристрій імітує обробку транзакції шляхом затримки на деякий час. Після цього пристрій повідомляє транзакцію про завершення обслуговування.

Далі правила дії повторюються. Пристрій працює впродовж усього часу моделювання.

Для роботи обслуговуючому пристрою необхідні дані, перелік яких наведено у таблиці 4.3.

Таблиця 4.3 – Атрибути абстракції Обслуговуючий пристрій

Назва поля	Клас	Призначення поля	Джерело
queue	QueueForTransaction	Посилання на чергу транзакцій	Model
rnd	Randomable	Посилання на генератор випадкових чисел	Gui
finishTime	double	Час моделювання	Gui

Атрибут queue – це черга транзакцій. Посилання на неї можна отримати через модель.

Атрибут rnd буде використовуватися для формування випадкової величини інтервалу часу, що витрачається на обробку транзакцій. Посилання на цей елемент можна отримати з візуальної частини.

Поле finishTime визначає час, впродовж якого має працювати пристрій. Значення цього атрибуту можна також отримати з візуальної частини проекту.

Посилання на візуальну частину і на модель має бути передано через конструктор.

Правила дії обслуговуючого пристрою схематично можна представити у вигляді діаграми діяльності, рисунок 4.3.

Рисунок 4.3 – Діаграма діяльності абстракції обслуговуючий пристрій

4.2.2 *Реалізація класів активних компонент моделі*

Класи, що потребують реалізації у даному проекті – це класи для транзакцій, генератора транзакцій та обслуговуючого пристрою. Для створення об'єктів цих класів використовуються конструктори, через які передаються посилання на візуальну частину та модель. На підставі цих посилань у конструкторі ініціалізуються усі необхідні поля. Таким чином, атрибути акторів приймають свої значення у момент створення актора. У тих випадках, коли налаштування актора потрібно змінити після їх створення, слід для цього створити відповідні методи.

Поля, що містять умови у вигляді лямбда функцій у конструкторі ініціалізувати недоцільно. Справа у тому, що ці функції можуть містити посилання на створюваний об’єкт, якого в цей час ще не існує. Тому лямбда функції для умов, якщо вони є, створюються у методі rule, тобто в момент запуску правил дії акторів.

4.2.2.1 **Клас Transaction**

Об’єкти цього класу представляють абстракцію транзакція. У лістингу 4.14 наведено текст класу з коментарями.

Лістинг 4.14 – Клас Transaction

```
public class Transaction extends Actor {
    private double createTime;
    private QueueForTransactions<Transaction> queue;
    private Histo histoQueue;
    private Histo histoService;
    private boolean serviceDone;

    public Transaction(Model model) {
        this.queue = model.getQueue();
        this.histoQueue = model.getHistoTransactionWaitInQueue();
        this.histoService = model.getHistoTransactionServiceTime();
    }

    public double getCreateTime() {
        return createTime;
    }

    public void setServiceDone(boolean b) {
        this.serviceDone = b;
    }

    @Override
    public String toString() {
        return "Transaction " + createTime;
    }

    @Override
    protected void rule() throws DispatcherFinishException {
        createTime = dispatcher.getCurrentTime();
        nameForProtocol = "Транзакція " + createTime;
        queue.add(this);
        waitForCondition(() -> !queue.contains(this), "мають забрати на обслуговування");
        histoQueue.add(dispatcher.getCurrentTime() - createTime);
        waitForCondition(() -> serviceDone, "мають завершити обслуговування");
        histoService.add(dispatcher.getCurrentTime() - createTime);
    }
```

4.2.2.1 **Клас Generator**

Об’єкти цього класу представляють абстракцію генератор. У лістингу 4.15 наведено текст класу з коментарями.

Лістинг 4.15 – Клас Generator

```
public class Generator extends Actor {
    //Посилання на модель системи
    private Model model;
    // Генератор випадкового часу створення транзакції
    private Randomable rnd;
    // Тривалість роботи генератора
    private double finishTime;


    // Конструктор
    public Generator(String name, GUI gui, Model model) {
            setNameForProtocol(name);
            this.model = model;
            finishTime = gui.getChooseDataFinishTime().getDouble();
            rnd = gui.getChooseRandomGen();
    }

    // Правила дії
    public void rule() {
        while (getDispatcher().getCurrentTime() <= finishTime) {
            holdForTime(rnd.next());
            getDispatcher().printToProtocol(
                    "  " + getNameForProtocol() + " створює транзакцію.");
            Transaction transaction = new Transaction(model);
            dispatcher.addStartingActor(transaction);
        }
    }
}
```

4.2.2.2 **Клас Device**

Об’єкти цього класу представляють абстракцію обслуговуючий пристрій. У лістингу 4.16 наведено текст класу з коментарями.

Лістинг 4. 16 - Клас Device

```
// Клас для обслуговуючого пристрою
public class Device extends Actor {

    // Черга для тразакцій
    private QueueForTransactions<Transaction> queue;
    // Генератор часу, що витрачає прилад на обслуговування транзакції
    private Randomable rnd;
    // Час роботи генератора
    private double finishTime;

    // Конструктор, у якому ініціалізуються усі поля класу
    // через доступ до моделі та візуальної частини
```

```
    public Device(String name, GUI gui, Model model) {
        setNameForProtocol(name);
        finishTime = gui.getChooseDataFinishTime().getDouble();
        rnd = gui.getChooseRandomDev();
        queue = model.getQueue();
    }

    public void rule() throws DispatcherFinishException {
        // Створюємо умову, виконання якої буде чекати актор
        BooleanSupplier queueSize = () -> queue.size() > 0;
        // цикл виконання правил дії
        while (getDispatcher().getCurrentTime() <= finishTime) {
            // Перевірка наявності транзакції та чекання на її появу
            waitForCondition(queueSize, "у черзі має з'явиться транзакція");
            Transaction transaction = queue.removeFirst();
            // Імітація обробки транзакції
            holdForTime(rnd.next());
            transaction.setServiceDone(true);
        }
    }
}
```

4.3 Порядок виконання роботи

4.3.1 *Підготовка до роботи*

– Завантажте проекти, що знаходяться в архіві SimulationAllLab.zip до Eclipse скориставшись функцією меню Import→Existing Project into Workspace→ Select archive file.

4.3.2 *Знайомство з класами, які забезпечують псевдопаралельну роботу потоків*

Відкрийте проект Simulation2018.

– Ознайомтеся з класом process.BooleanSemaphore.

– Ознайомтеся з класом process.Actor.

– Ознайомтеся з класом process.Dispatcher.

4.3.3 *Знайомство з прикладом побудови моделі СМО*

– Ознайомтеся з класами акторів пакету testModel проекту SimulationAllLab. Саме ці класи розглядаються як приклад у методичних вказівках до лабораторної роботи. Активізуйте клас GUI і перевірте працездатність проекту.

4.3.4 *Знайомство з прикладом виконання РГР*

– Ознайомтеся з класами акторів пакету buldo2018 проекту SimulationAllLab.

– Почніть створювати акторів для своєї РГР.

4.4 Завдання для самостійної роботи

Створіть класи акторів для свого варіанту РГР і перевірте їх в режимі «Тест». Обов'язково аналізуйте протокол роботи моделі під час налагодження коду.

4.5 Вимоги до звіту

- Назва і мета роботи.
- Діаграма класів для створеного проекту.
- Діаграми діяльності акторів з РГР.
- Тексти створених класів.
- Протокол роботи створеної моделі (короткий).
- Копії екранів з результатами роботи моделі для різних налаштувань.
- Висновки по роботі.

4.6 Контрольні питання

1. Опис класу Dispatcher, його змінні та методи.
2. Опис класу Actor, його змінні та методи.
3. Клас BooleanSemaphore та його призначення.
4. Приклади запису умов для методу waitForCondition.
5. Правила дії об'єктів класів застосування, що було розглянуто у якості прикладу, та створеного застосування.

5 ЛАБОРАТОРНА РОБОТА № 5. ДОСЛІДЖЕННЯ НАЙПРОСТІШОЇ СМО

Мета роботи:

– Знайомство з методикою аналітичного дослідження характеристик найпростішої СМО.

– Експериментальне визначення характеристик найпростішої СМО шляхом моделювання.

5.1 Короткі теоретичні відомості

У попередній лабораторній роботі були розглянуті базові класи, що дозволяють будувати моделі СМО. Але при моделюванні нас цікавить не сама модель, а статистична інформація про її роботу.

Насамперед, робота моделі має бути візуалізована, тобто дослідник повинен мати можливість спостерігати за тим, як поводить себе модель. Звичайно для візуалізації на екран виводиться інформація про поточні значення часу моделювання, розміри черг та стана об'єктів. Ця інформація може виводитися на екран або в цифровому вигляді, або як переміщень різного роду покажчиків. Обидва способи мають свої плюси й мінуси, але варто враховувати, що аналогову інформацію людина сприймає краще. Можна використовувати і мультиплікацію. У попередній лабораторній роботі була використана аналогова динамічна індикації розмірів черг.

Крім візуалізації модель повинна видавати користувачеві статистичну інформацію про параметри, які його цікавлять. При моделюванні систем масового обслуговування найчастіше збирають наступну статистичну інформацію:

– статистичні характеристики для довжини черги;
– статистичні характеристики для часу очікування в черзі;
– статистичні характеристики для часу обслуговування;
– завантаження обслуговуючого приладу.

Реальна модель повинна створюватися із урахуванням цілей моделювання. При цьому варто використовувати базові класи пакетів process, qusystem, rnd, і на їхній основі, використовуючи механізм наслідування, створювати підкласи, які забезпечують вирішення поставлених задач. Найчастіше в цих підкласах перевизначають методи обробки подій, які ініціюються в базових класах, але іноді доводиться перевизначити і правила дії об'єктів. Для прискорення побудови моделей варто також використовувати допоміжні класи пакетів stat і paint.

Наявність стандартних класів, на базі яких будується модель, підвищує ймовірність того, що модель побудована вірно. Однак не виключено, що при програмуванні моделі були допущені логічні помилки, в результаті чого модель не буде адекватною системі, що досліджується. Тому досить важливо мати можливість порівняти результати моделювання з відомими результатами, хоча

б для деяких варіантів роботи моделі, щоб зменшити ймовірність одержання невірних результатів.

Найчастіше для порівняння використовують результати аналітичного дослідження СМО.

5.1.1 *Аналітичне дослідження СМО*

Аналітичні рішення знайдені тільки для деяких класів СМО. Як правило, це системи, у яких випадкові процеси підкоряються експоненціальному закону розподілу. Такі системи називаються марківськими. Цю назву системи одержали тому, що розрахунок їх можна проводити, використовуючи ланцюги Маркова. Цінність аналітичних рішень для імітаційного моделювання полягає в тому що, маючи теоретичні значення параметрів системи, ми можемо оцінити, наскільки точні результати будуть одержані шляхом моделювання.

5.1.1.1 **Ланцюг Маркова для простішої СМО**

Ланцюг Маркова для найпростішої СМО представлено на рисунку 5.1.

Станам системи відповідають вершини цього графа, номера яких відповідають кількості заявок у системі. Стан 0 - це стан, коли в системі немає заявок, і обслуговуючий прилад чекає. Імовірність того, що система перебуває в цьому стані, позначимо P0. Стан 1 - це стан, коли в системі одна заявка, і вона обслуговується. Довжина черги при цьому, також як і в попередньому стані, дорівнює 0. Імовірність цього стану позначимо P1. Стан 2 - це стан, коли в системі дві заявки, одна обслуговується а друга чекає в черзі. Довжина черги при цьому дорівнює 1. Імовірність цього стану позначимо P2, і так далі.

Ребра графа визначають можливі переходи системи з одного стану в інший. У кожний момент часу відбувається тільки одна подія. Якщо генератор додає заявку в чергу, то система переходить у наступний за номером стан. Якщо обслуговуючий прилад закінчує обробку заявки, то система переходить у попередній стан.

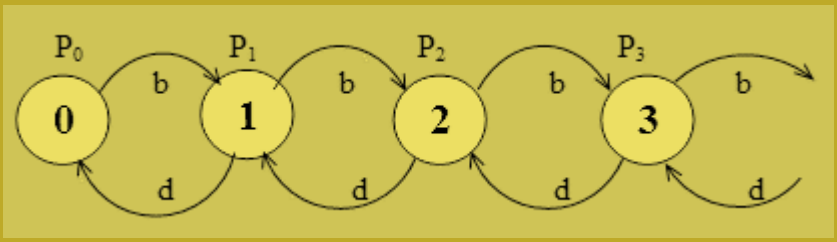

Рисунок 5.1 – Ланцюг Маркова для найпростішої СМО

Кожному ребру відповідає деяка ймовірність відповідного переходу. Процес переходів, представлений на рисунку 5.1, у науковій літературі іноді називають процесом розмноження й загибелі. Імовірність b (birth) залежить від інтенсивності генератора, а ймовірність d (death) - від продуктивності обслуговуючого приладу.

5.1.1.2 **Визначення ймовірностей станів системи в сталому режимі**

Стан ланцюга Маркова характеризується значеннями ймовірностей станів. У початковому стані системи ймовірність стану 0 дорівнює 1, а ймовірності інших станів нульові. У процесі роботи системи ці ймовірності змінюються, але згодом їх значення стабілізуються й приймають деякі постійні значення. Такий режим роботи системи називається сталим. У сталому режимі перехід з одного стану в інший відбуваються таким чином, що ймовірності станів не змінюються.

Здатність ланцюгів Маркова з часом переходити у сталий режим роботи називають «ергодичною властивістю».

Аналізуючи сталий режим у ланцюзі Маркова, можна визначити сталі значення ймовірності кожного стану системи. Для цього записуються рівняння балансу ймовірностей, засновані на такому твердженні: «якщо ймовірність стану не міняється, то ймовірність переходу системи в цей стан дорівнює ймовірності її виходу із цього стану».

Розглянемо рівняння балансу для нульового стану, виходячи із графа, представленого на рисунку 5.1.

Імовірність того, що система залишить стан 0, дорівнює $\mathbf{P_0 b}$, а ймовірність того, що система повернеться у стан 0, дорівнює $\mathbf{P_1 d}$.

Рівняння балансу ймовірностей для нульового стану запишеться так: $\mathbf{P_0 b = P_1 d}$, звідки одержимо $\mathbf{P_1 = b/d\ P_0}$.

Можна показати, що відношення **b/d** дорівнює відношенню інтенсивності надходження заявок у СМО до інтенсивності їхнього обслуговування. Це відношення в літературі зазвичай позначають грецькою буквою **ω** і називають коефіцієнтом завантаження системи. Тоді $\mathbf{P_1 = \omega P_0}$.

Розглянемо тепер перший стан.

Імовірність того, що система залишить цей стан, дорівнює $\mathbf{P_1 b + P_1 d}$, а ймовірність того, що вона в нього повернеться, буде $\mathbf{P_0 b + P_2 d}$.

Умова балансу ймовірностей буде така: $\mathbf{P_1 b + P_1 d = P_0 b + P_2 d}$. З огляду на те, що $\mathbf{P_1 d = P_0 b}$, одержимо: $\mathbf{P_1 b = P_2 d}$, **або** $\mathbf{P_2 = \omega P_1}$.

Підставивши в останню рівність $\mathbf{P1 = \omega P0}$, одержимо $\mathbf{P2 = \omega 2 P0}$

Записуючи умови балансу для наступних станів, одержимо для них аналогічні співвідношення. У загальному випадку можна записати

$$P_i = \omega^i \cdot P_0 \qquad (5.1)$$

Варто також враховувати, що в будь-який момент часу система повинна обов'язково перебувати в якому-небудь зі станів, отже, сума всіх ймовірностей дорівнює одиниці.

$$\sum_{i=0}^{\infty} P_i = 1 \qquad (5.2)$$

З огляду на співвідношення 5.1 і 5.2, можна одержати рівняння для визначення $\mathbf{P_0}$

$$\sum_{i=0}^{\infty} \omega^i \cdot P_0 = 1 \qquad (5.3)$$

Сума в лівій частині рівняння являє собою суму елементів геометричної прогресії і за умови **ω <1** дорівнює **P_0/(1-ω).** Звідси маємо:

$$P_0 = 1 - \omega \qquad (5.4)$$

З огляду на співвідношення 5.1 і 5.4, можна одержати рівняння для визначення ймовірності будь-якого стану системи в сталому режимі.

$$P_i = (1-\omega) \cdot \omega^i \qquad (5.5)$$

5.1.1.3 **Визначення ймовірностей появи черг різної довжини.**

Якщо система перебуває в станах 0 або 1, то черга відсутня, тому що в нульовому стані в системі зовсім немає заявок, а в першому - єдина заявка яка обслуговується, і черга порожня.

Отже, імовірність появи черги нульової довжини дорівнює сумі ймовірностей першого й другого стану.

$$P_{q=0} = P_0 + P_1 = (1-\omega) + (1-\omega) \cdot \omega = 1 - \omega^2 \qquad (5.6)$$

Якщо розглянути деякий і-ий стан, то для нього черга буде мати довжину і-**1**, тому ймовірність появи черги довжиною **k**, буде дорівнювати ймовірності **k+1** стану.

$$P_{q=k} = (1-\omega) \cdot \omega^{k+1} \qquad (5.7)$$

5.1.1.4 **Визначення середньої довжини черги**

Знаючи ймовірності появи черги будь-якої довжини, ми можемо визначити середню довжину черги по наступній формулі:

$$\bar{q} = \sum_{k=1}^{\infty} k \cdot P_{q=k} \qquad (5.8)$$

Підставивши в цю формулу ймовірність появи черги довжиною k, яка визначається по співвідношенню 5.7, одержимо

$$\bar{q} = \sum_{k=1}^{\infty} k(1-\omega) \cdot \omega^{k+1} = (1-\omega)\omega \sum_{k=1}^{\infty} k\omega^k \qquad (5.9)$$

З огляду на те, що

$$\sum_{k=0}^{\infty} k\cdot\omega^k = \omega + 2\omega^2 + 3\omega^3 + 4\omega^3 + \ \ldots \ =$$

$$=\omega+\omega^2+\omega^3+\omega^4+ \ \ldots \ +\omega^2+\omega^3+\omega^4+ \ \ldots \ +\omega^3+\omega^4+ \ \ldots \ =$$

$$=\omega(1+\omega+\omega^2+\omega^3+\ldots) \ +\omega^2(1+\omega+\omega^2+\omega^3+\ldots) \ +\omega^3(1+\omega+\omega^2+\omega^3+\ldots) \ +\ldots=$$

$$=(1+\omega+\omega^2+\omega^3+\ldots)\cdot(\omega+\omega^2+\omega^3+\ldots)=$$

$$=\omega(1+\omega+\omega^2+\omega^3+\omega^4+\ldots)^2=$$

$$=\omega/(1-\omega)^2,$$

одержимо

$$\bar{q} = (1-\omega)\cdot\omega\cdot\frac{\omega}{(1-\omega)^2} = \frac{\omega^2}{1-\omega} \qquad (5.10)$$

5.1.1.5 **Визначення середнього часу очікування в черзі**

Середній час очікування в черзі можна знайти, використовуючи так званий результат Літла.

Результат Літла говорить, що середню кількість заявок у черзі для марківської СМО можна знайти як добуток інтенсивності потоку на вході та середнього часу перебування заявок у черзі.

$$\bar{q} = \lambda\cdot\bar{\tau} \qquad (5.11)$$

Із цього співвідношення можна знайти вираз для визначення середнього часу очікування в черзі

$$\bar{\tau} = \bar{q}/\lambda \qquad (5.12)$$

Підставивши в співвідношення 5.12 вираз для середньої довжини черги 5.10, одержимо формулу 5.12 для визначення середнього часу очікування в черзі.

$$\bar{\tau} = \frac{\omega^2}{\lambda\cdot(1-\omega)} \qquad (5.12)$$

5.2 Дослідження СМО шляхом моделювання

Характеристики СМО, які оцінювалися вище, можуть бути знайдені й шляхом проведення експериментів з моделлю цієї системи.

5.2.1 *Опис проекту*

У лабораторній роботі досліджується проста марківська система масового обслуговування. До складу моделі входять генератор заявок, черга й обслуговуючий прилад. Система проектувалася виходячи з того, що вона повинна забезпечити:

– виведення діаграми зміни довжини черги у часі;

– накопичення, обробку й видачу (у вигляді гістограми й тексту) статистичної інформації про довжину черги;

– накопичення, обробку й видачу (у вигляді гістограми й тексту) статистичної інформації про час очікування заявок у черзі;

– забезпечувати налаштування параметрів моделі - тривалість моделювання, максимальний розмір черги, статистичні характеристики потоку заявок і часу обслуговування.

Проект було реалізовано з використанням технології проектування, що була розглянута у попередній лабораторній роботі та у РГР.

5.2.2 *Клас testTheory.TheoryGUI*

Клас testTheory.TheoryGUI реалізує візуальну частину застосування, що оформлена так само, як і попередньому проекті.

Ліва панель інтерфейсу користувача, яку видно на рисунку 5.2, забезпечує налаштування моделі. Праворуч розташовано компонент TabbedPane, який містить дві закладки. Перша закладка забезпечує запуск моделі у режимі тестування з динамічною індикацію довжини черги.

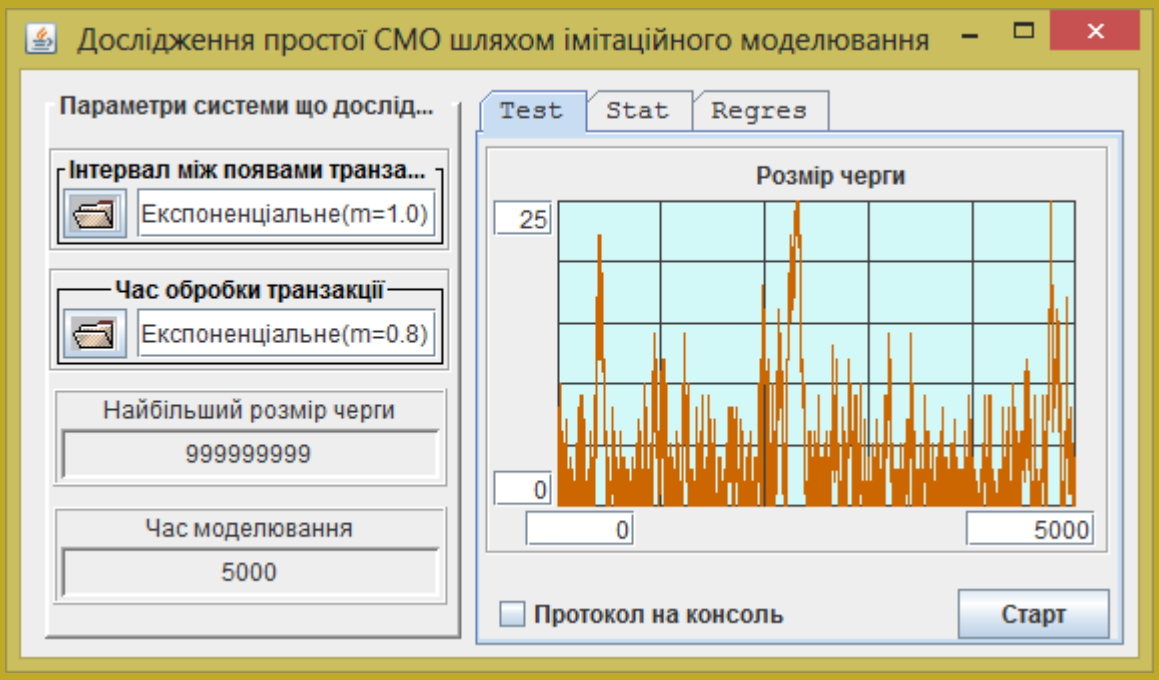

Рисунок 5.2 – Вигляд першої сторінки проекту

Запуск моделі відбувається при натисканні кнопки «Старт».

Структура моделі та її запуск у режимі тестування нічим не відрізняється від запуску моделі у попередніх лабораторних роботах.

Друга закладка використовується для запуску моделі у режимі накопичення та відображення статистичної інформації. Зовнішній вигляд візуальної частини проекту для цього режиму наведено на рисунку 5.3.

На закладці розташований компонент класу widjets.stat.StatisticsManager фреймворку Simulation. Цей компонент забезпечує запуск моделі у режимі отримання статистичних даних та відображає отримані статистичні дані.

Для того щоб компонент запрацював, йому потрібно передати посилання на фабрику моделей через метод setFactory(IModelFactory). У класі, що розглядається, це зроблено за допомогою лямбда функції:

statisticsManager.setFactory((d)-> new Model(d, this));

5.2.3 *Інші класи проекту*

Даний проект відрізняється від проекту, що розглядався у попередній роботі тільки тим, що тут не створюється активна транзакція. В якості транзакції тут використовується значення часу появи транзакції, яке генератор заносить до черги, а обслуговуючий пристрій вилучає ці дані з черги, визначає час перебування заявки у черзі і передає гістограмі.

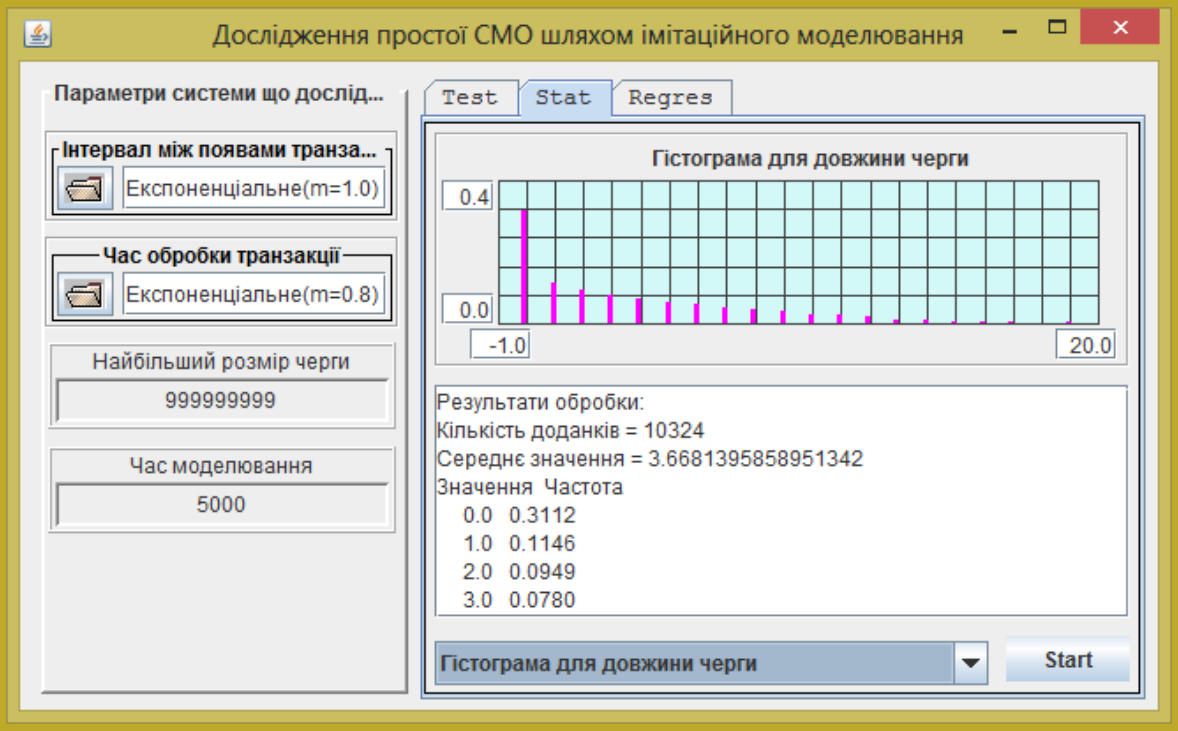

Рисунок 5.3 – Вигляд другої сторінки проекту

Компонент вимагає, окрім того, щоб модель реалізувала інтерфейс IStatisticsable, який буде розглянутий у наступному пункті.

5.2.4 *Статистична інформація про довжину черги*

Для збору статистичної інформації про довжину черги в даному проекті використовується об’єкт queue класу QueueForTransactions, який моделює чергу в системі. Цей об’єкт забезпечує і динамічну індикацію зміни своїх розмірів на діаграмі. Для накопичення статистичної інформації об’єктові queue було передано посилання на об'єкт discretHisto класу DiscretHisto. Об’єкт queue перед кожною зміною розміру черги передає до об'єкту discretHisto свій розмір та час, на протязі якого цей розмір не змінювався.

Програмно це реалізовано через методи beforeAdd() і beforeRemove(), які викликаються перед зміною розміру черги. Ці методи викликають приватний метод accum(), що передає дані у гістограму через метод

addFrequencyForValue(double frequency, double value), frequency – це різниця між поточним часом і моментом попередньої зміни черги, а value - це довжина черги.

Для зберігання значення моменту часу, пов'язаного з попередньою зміною черги, використовується приватна змінна екземпляра lastTime. Очищення черги й ініціалізація змінної lastTime відбувається в методі init().

Текст методу accum() наведений у лістингу 5.1.

Лістинг 5.1 - Метод накопичення інформації про довжину черги

```
private void accum() {
  if (discretHisto != null) {
    discretHisto.addFrequencyForValue(
      dispatcher.getCurrentTime() - lastTime, size());
    lastTime = dispatcher.getCurrentTime();
  }
}
```

У класі DiscretHisto ці дані накопичуються спочатку у колекції. Якщо розмір колекції перевищує 500, або процес накопичення закінчився, у класі створюється два масиви. Один для унікальних значень параметру value, другий для сум відповідних значень frequency. Таким чином дані value опиняються у елементах масиву arrayForValue, а відповідніі інтервали часу накопичуються в елементах масиву arrayForAbsolutFrequency.

Якщо після цього значення value перевищує максимальне із значень, заданих у масиві arrayForValue, то переданий інтервал накопичується в останньому елементі масиву. Для пошуку середньої довжини черги в класі DiscretHisto введені змінні totalInterval і totalIntegral. У першій з них накопичується загальна сума відрізків часу які заносяться в гістограму. Наприкінці моделювання тут буде величина, що дорівнює всьому періоду накопичення інформації. У другий змінній накопичується інтеграл від довжини черги за часом, значення якого визначається шляхом множення поточної довжини черги на ширину тимчасового інтервалу між поточним часом і часом попередньої події. Наприкінці моделювання, щоб оцінити ймовірність появи кожного із значень масиву arrayForValue кожен з елементів масиву arrayForAbsolutFrequency ділиться на значення змінної totalInterval. А розділивши значення змінної totalIntegral на значення змінної totalInterval, одержуємо середню довжину черги.

5.2.5 *Статистична інформація про час очікування в черзі*

У рішенні цієї задачі бере участь декілька компонентів моделі. Заявка створюється об'єктом класу theorStudy.TransactGenerator і являє собою час її створення. Отже, від генератора заявок, через чергу, до обслуговуючого приладу, передається час появи заявки, яка обслуговується.

У класі theorStudy.Device, який використовується для створення об'єкта, що моделює роботу обслуговуючого приладу, забезпечується обчислення часу очікування в черзі. Час очікування визначається як різниця між поточним часом і значенням змінної, що вилучена з черги. Ця інформація передається об'єкту класу Histo за допомогою методу add(double). У цьому об’єкті дані спочатку просто накопичуються у колекції. Коли їх кількість перевищує 500, ці дані аналізуються, визначається мінімальне та максимальне значення для даних. Після цього визначається кількість інтервалів і гістограма ініціалізується. Завершується цей етап перенесенням даних з колекції до гістограми. Надалі, нові дані заносяться безпосередньо до гістограми.

5.3 Порядок виконання роботи

5.3.1 *Підготовка до роботи*

– Активізуйте Java застосування, що знаходиться у архіві SimulationAllLab.jar, або завантажте проект SimulationAllLab.zip до Eclipse.

5.3.2 *Знайомство з теоретичними відомостями*

– Ознайомтесь з методикою аналітичного дослідження СМО.
– Ознайомтесь із програмою моделювання СМО.
– Ознайомтесь з методикою накопичення статистичних даних у моделі.

5.3.3 *Перевірка працездатності програмного комплексу*

– Активізувавши пункт меню «Лабораторне застосування» переконайтеся в працездатності програмного комплексу.

5.3.4 *Експерименти з моделлю*

– Налаштуйте для інтервалу між надходженнями заявок експоненціальний закон розподілу із середнім значенням інтервалу (m) між надходженнями заявок рівним останній цифрі номеру вашої залікової книжки +1.

– Налаштуйте тривалість моделювання в 10000 разів більше ніж інтервал між надходженнями заявок.

– Налаштуйте максимальний розмір черги 99999.

– Проведіть експерименти з моделлю, для коефіцієнта завантаження системи 0.8. Для цього встановлюйте для часу обслуговування заявки експоненціальний закон розподілення із середнім значенням рівним добутку коефіцієнта завантаження на середнє значення інтервалу між надходженнями заявок (m * 0.8). Повторіть експерименти не менш трьох разів. Результати фіксуйте у таблиці 5.1.

– Результати експериментів з таблиці 5.1 покажіть у вигляді стовпчастої діаграми, зображуючи теоретичні й середні експериментальні значення різними кольорами.

Таблиця 5.1 - Результати експериментів по оцінці ймовірностей появи черг

Коефіцієнт завантаження $\omega = \mu / \lambda = m_{device} / m_{generator} = 0{,}8$								
Час моделювання …….....								
Інтенсивність вхідного потоку $\lambda = 1/m_{generator}$ =…… Інтенсивність обслуговування $\mu = 1/m_{device}$ =.....								
Довжина черги		0	1	2	3	4	5	6
Теор. значення ймовірності								
Експериментальні значення відносної частоти	Дослід№1							
	Дослід№2							
	Дослід№3							
	Середнє							

– Проведіть експерименти по оцінці середньої довжини черги, для значень коефіцієнта завантаження системи 0.3, 0.4, 0.5, 0.6, 0.7, 08. Повторіть експерименти для кожного коефіцієнта завантаження не менш трьох разів. Результати фіксуйте у таблиці 5.2.

– Результати експериментів з таблиці 5.2 покажіть у вигляді графіків із зображенням експериментальних даних і теоретичних значень. Через точки із теоретичними значеннями проведіть лінію регресії.

Таблиця 5.2 - Результати експериментів по оцінці середньої довжини черги

Час моделювання ……							
Інтенсивність вхідного потоку $\lambda = 1/m_{generato}$=……							
Коефіцієнт завантаження		0.3	0.4	0.5	0.6	0.7	0.8
Теоретичне значення середньої довжини черги							
Експериментальні значення середньої довжини черги	Дослід№1						
	Дослід№2						
	Дослід№3						
	Середнє						

– Проведіть експерименти по оцінці середнього часу очікування в черзі, для значень коефіцієнта завантаження системи 0.3, 0.4, 0.5, 0.6, 0.7, 08. Повторіть експерименти для кожного коефіцієнта завантаження не менш трьох разів. Результати записуйте у таблицю 5.3.

– Результати експериментів з таблиці 5.3 покажіть у вигляді графіків із зображенням експериментальних даних і теоретичних значень. Через точки із теоретичними значеннями проведіть лінію регресії.

Таблиця 5.3 - Результати експериментів по оцінці середнього часу очікування в черзі

Час моделювання ……						
Інтенсивність вхідного потоку $\lambda = 1/m_{generator}$=……						
Коефіцієнт завантаження	0,3	0,4	0,5	0.6	0.7	0.8
Теоретичне значення середнього часу очікування в черзі						

Експериментальні значення середнього часу очікування в черзі	Дослід№1						
	Дослід№2						
	Дослід№3						
	Середнє						

– Порівняйте результати експериментів з теоретичними значеннями й зробіть висновок про можливість використання моделі для оцінки параметрів системи, яка моделюється.

5.4 Завдання для самостійної роботи

Реалізуйте в моделі для свого варіанту РГР інтерфейс IStatisicsable, а до візуальної частини додайте закладку з компонентом StatisticsManager і виконайте необхідні налаштування.

Перевірте роботу застосування у режимі отримання статистичних даних.

5.5 Зміст звіту

– Назва й ціль роботи.

– Результати експериментів з моделлю у вигляді копій екранів, таблиць, графіків і стовпчастих діаграм.

– Висновки за результатами порівняння теоретичних і експериментальних даних і можливості застосування моделі для дослідження СМО;

– Результати виконання завдання до самостійної роботи.

Контрольні питання

1. Намалювати діаграму класів проекту класів, що реалізовані у проекті і дати характеристику зв’язків між об’єктами класів .
2. Як створюється модель і які завдання вона виконує.
3. Як виводиться діаграма зміни довжини черги.
4. Як у моделі визначається середня довжина черги.
5. Як у моделі визначається час очікування в черзі.
6. Як у моделі визначається ймовірність появи черги якоїсь довжини.
7. Як у моделі визначається середній час обслуговування заявки.
8. Як у моделі визначити завантаження обслуговуючого приладу.
9. Як теоретично визначається ймовірність появи черги однієї або іншої довжини.
10. Як теоретично визначається середня довжина черги.
11. Як теоретично визначається час очікування в черзі.
12. Як теоретично визначити середній час обслуговування заявки.
13. Як теоретично визначити завантаження обслуговуючого приладу.

6 ЛАБОРАТОРНА РОБОТА № 6. АВТОМАТИЗАЦІЯ ПРОВЕДЕННЯ ОДНОФАКТОРНИХ ЕКСПЕРИМЕНТІВ З ІМІТАЦІЙНИМИ МОДЕЛЯМИ

Мета роботи:

– Знайомство з методикою планування та обробки результатів однофакторних багаторівневих експериментів на прикладі дослідження впливу коефіцієнту завантаження системи на показники роботи простої системи масового обслуговування.

6.1 Короткі теоретичні відомості про планування однофакторних експериментів та обробку їх результатів

Зазвичай при моделюванні проводиться не один, а багато експериментів. Це пов’язано, в першу чергу, з тим, що результат модельного експерименту величина випадкова і тому потребує оцінки точності, а точність отриманих даних можна оцінити лише в тому випадку, якщо ми маємо у розпорядженні вибірку випадкових величин. Крім того, моделювання часто проводиться з метою знайти залежність деякої характеристики досліджуваної системи від факторів, які впливають на неї. Для цього доводиться проводити серії експериментів при різних значеннях факторів. Чим більше експериментів в одній серії і чим більше різних значень факторів, які аналізуються, тим точніше дослідник може оцінити залежність, яка його цікавить. З іншого боку, чим більше експериментів, тим більше витрат на їх проведення. Тому при моделюванні доводиться вирішувати задачу планування експериментів для того, щоб досягти бажаного результату з мінімальними затратами.

За звичай, при проведенні експериментів потрібно вирішити такі задачі:

– визначення значень факторів, для яких проводяться серії експериментів;

– визначення кількості експериментів в одній серії;

– вибір функції регресії для отриманих результатів і визначення параметрів цієї функції;

– оцінка адекватності отриманої функції регресії результатам експерименту;

– оцінка точності отриманих результатів.

6.1.1 *Однофакторний експеримент на одному рівні*

Експерименти на одному рівні проводять з метою оцінки значення функції відгуку для даного значення фактору. Невідоме значення функції відгуку визначається за результатами серії із n експериментів шляхом знаходження середнього значення цих результатів.

Отримане таким чином значення функції відгуку називають крапковою оцінкою, від слова крапка, тобто графічно це значення відображається крапкою.

У випадку, якщо експериментів проведено дуже багато, то отримане середнє буде представляти значення функції відгуку досить точно. Це

пов’язано з тим, що дисперсія середнього у n раз менше дисперсії початкових випадкових величин, де n - кількість повторних експериментів..

При вибірках малого об’єму крапкова оцінка може значно відрізнятись від дійсного значення оцінюваної величини. З цієї причини при невеликих вибірках прийнято користуватися інтервальними оцінками. У цьому випадку значення функції відгуку визначається довірчим інтервалом, який задає діапазон, з якого може приймати значення функція відгуку, «від… до…».

Для того щоб правильно спланувати кількість експериментів одно факторного експерименту, потрібно знати, як це значення впливає на ширину довірчого інтервалу.

6.1.1.1 Довірчий інтервал

Довірчий інтервал визначається двома числами – кінцями інтервалу. Границі інтервалу обчислюються таким чином, щоб цей інтервал охоплював дійсне значення оцінюваної величини із заданою імовірністю.

Імовірність, з якою довірчий інтервал охоплює істинне значення величини, називають довірчою імовірністю, або надійністю.

Ширину довірчого інтервалу можна знайти, якщо відомий закон розподілення для випадкової величини середнього значення. Виходячи з того, що середнє значення визначається на підставі суми випадкових величин, що незначно відрізняються одна від одної, ця величина підпорядковується нормальному закону (теорема Ляпунова).

Дисперсію середнього значення можна знайти, обрахувавши дисперсію результатів експериментів $D_{експ}$.

$$D_{ср} = D_{експ} / n \tag{6.1}$$

Для нормального розподілення відомо, що з імовірністю γ випадкова величина попадає в діапазон

$$m \pm t_{\gamma} \cdot \sigma \tag{6.2}$$

де m – математичне очікування випадкової величини відгуку;

σ – середньоквадратичне відхилення;

γ – довірча імовірність;

t_{γ} – число, для якого подвійне значення функції Лапласа буде дорівнювати γ.

Для імовірності γ = 0.95, t_{γ} = 1.96. Для γ = 0.99, t_{γ} = 2.57. Для інших ймовірностей це число можна знайти в таблицях функції Лапласа. За звичай приймають γ = 0.95.

Таким чином, якщо знайдено середнє значення функції відгуку $y_{ср}$, і відома дисперсія результатів експерименту, то можна стверджувати, що з імовірністю γ істинне значення *y* знаходиться у діапазоні

$$y_{ср} - t_{\gamma} \cdot \sqrt{D_{эксп} / n} \;\; .. \;\; y_{ср} + t_{\gamma} \cdot \sqrt{D_{эксп} / n} \tag{6.3}$$

Границі цього діапазону і задають довірчий інтервал для y.

Формула для обчислення половини ширини довірчого інтервалу набуває такого вигляду

$$\Delta_{y_{ср}} = t_{\gamma} \cdot \sqrt{D_{эксп} / n} \qquad (6.4)$$

Із цієї формули випливає, що ширина довірчого інтервалу збільшується пропорційне квадратному кореню із дисперсії експерименту і звужується пропорційне квадратному кореню із кількості експериментів. Отже, якщо ми хочемо зменшити ширину інтервалу вдвічі, потрібно провести у 4 рази більше експериментів.

Слід мати на увазі, що величину t_{γ} можна знаходити у відповідності з функцією Лапласа тільки при об'ємах вибірки більших 30. Це пов'язано з тим, що оцінка дисперсії експерименту $D_{эксп}$ по вибірці малого розміру буде неточною. Тому, при малих вибірках, для визначення t_{γ} потрібно використовувати критичні значення розподілення Ст'юдента для n-1 числа степенів свободи, рівні значимості, який дорівнює 1-γ і двосторонній критичній області. Це призводить до розширення довірчого інтервалу. Так при 95% надійності і об'ємі вибірки 30 коефіцієнт t_{γ} дорівнює 2.05, а при об'ємі вибірки 10 стає рівним 2.23. Тобто при малих об'ємах вибірки, ширина довірчого інтервалу при зменшенні числа експериментів буде збільшуватись швидше, ніж корінь квадратний із об'єму вибірки.

Таким чином, число експериментів в серії може бути вибрано виходячи з потрібної точності отриманих результатів із урахуванням дисперсії експерименту.

6.1.2 *Однофакторний багаторівневий експеримент*

6.1.2.1 Планування багаторівневого експерименту

Експерименти на декількох рівнях фактору проводять з метою визначення залежності функції відгуку від значення фактору. Перша проблема, яку при цьому необхідно вирішувати, це визначення кількості рівнів фактору, на яких буде проводитися експеримент.

При вирішенні цієї проблеми потрібно керуватись такими міркуваннями:

– з точки зору оцінки точності результатів, краще провести багато експериментів на невеликій кількості рівнів, ніж небагато експериментів на багатьох рівнях;

– кількість рівнів повинна бути не менше, ніж число невідомих коефіцієнтів функції регресії;

– перевірку на адекватність можна провести тільки в тому випадку, якщо число рівнів фактору більше, ніж число коефіцієнтів функції регресії, які необхідно визначити;

– бажано, щоб число експериментів на рівнях було однаковим.

Враховуючи ці міркування, можна рекомендувати вибирати число рівнів на 1 більше, ніж число невідомих коефіцієнтів функції регресії. Якщо ж

припущень про вигляд цієї функції нема, слід починати проведення експериментів на трьох рівнях, а потім, у випадку необхідності, поступово збільшувати їх кількість.

Після вибору кількості рівнів слід вибирати значення фактору для кожного рівня. При вирішенні цієї проблеми можна керуватись наступним:

– експерименти обов'язково проводяться на краях області, що нас цікавить;

– значення фактору, що залишились, бажано рівномірно розподіляти в області, яка досліджується.

6.1.2.2 Обробка експериментальних даних

Після проведення експериментів слід визначити середнє значення функції відгуку y_{jcp} і дисперсії випадкової величини D_{yj} для кожного рівня.

Отримані середні значення функції відгуку для рівнів можуть бути використані в подальшому для отримання функції регресії, а значення дисперсії – для обчислення середньої дисперсії експерименту.

6.1.2.3 Перевірка однорідності дисперсії

Дисперсія функції відгуку на різних рівнях повинна бути приблизно однаковою. Лише в цьому випадку можна буде довіряти статистичній оцінці значимості впливу фактору на функцію відгуку і оцінці адекватності моделі.

Для того щоб оцінити, чи викликана різниця в отриманих значеннях дисперсії лише випадковими факторами, можна використовувати критерій Кочрена. Він використовується лише в тому випадку, коли число експериментів на кожному рівні однакове**. В якості оцінки в цьому критерії використовується відношення максимальної дисперсії до суми всіх дисперсій**. Очевидно, що це відношення не може бути більше одиниці і менше 1/p, де p – кількість рівнів. В останньому випадку всі дисперсії однакові.

Отримане значення відношення порівнюють із критичним значенням, яке можна знайти в статистичних таблицях розподілення Кочрена по кількості степенів свободи для оцінюваних дисперсій, яке дорівнює n-1 (n – кількість дослідів на рівні), кількості порівнюваних дисперсій і прийнятому рівневі значимості. Наприклад, при рівні значимості 0.05, кількості рівнів 4 і кількості експериментів на рівні 10 отримуємо критичне значення 0.5.

Якщо розраховане значення відношення менше критичного, дисперсії можна вважати однорідними.

Якщо дисперсії однорідні, то в якості оцінки точності експериментів, які проводяться, приймають їх середнє значення.

$$D_{эксп} = \sum_{j=1}^{p} D_{y_j} \Big/ p \qquad (6.5)$$

Якщо дисперсії неоднорідні, експериментатор ризикує загіпнотизувати себе числовими оцінками, які насправді помилкові. Тому потрібно спробувати зробити дисперсії однорідними. Для цього можна використати деяке

перетворення функції відгуку. Досить часто допомагає логарифмічне перетворення, тобто замість рівняння регресії виду y=f1(x), використовують рівняння ln(y)=f2(x).

6.1.2.4 Перевірка значимості фактору

Визначивши середнє значення функції відгуку для значень фактору, які аналізуються, і знаючи дисперсію експерименту, можна оцінити, чи впливає фактор на функцію відгуку. Якщо відповідь на це питання не очевидна, проводять статистичну оцінку значимості фактору за допомогою дисперсійного аналізу.

Основна ідея дисперсійного аналізу полягає у порівнянні відхилень, які викликані змінами фактору, з відхиленнями, які викликані похибками експерименту.

Для оцінки відхилень, викликаних змінами фактору, використовується дисперсія фактору ($D_ф$), яка представляє собою дисперсію випадкових величин відхилень середніх значень функції відгуку від загального середнього.

$$D_{ф} = \frac{\sum_{i=1}^{p} (y_i^{ср} - \bar{y})^2}{p-1} \qquad (6.6)$$

де $y_{iср}$ – середнє значення функції відгуку на і-му рівні;

$\bar{y}$ – загальне середнє функції відгуку на всіх рівнях.

Число степенів свободи для дисперсії, отриманої за цією формулою, дорівнює p-1.

Вплив фактора має значення в тому випадку, якщо дисперсія фактора набагато більше дисперсії середніх значень функції відгуку, яка може бути знайдена із дисперсії експерименту $D_{эксп}$, по формулі 6.7

$$D_{y^{ср}} = \frac{D_{эксп}}{n} \qquad (6.7)$$

де n – кількість експериментів на одному рівні.

Кількість степенів свободи дисперсії, отриманої за цією формулою, така ж, як і у дисперсії експерименту, і дорівнює p*(n-1).

Для порівняння дисперсій можна використовувати критерій Фішера.

Вплив фактора суттєвий, якщо відношення дисперсії фактора до дисперсії середніх значень функції відгуку більше критичного значення критерію Фішера. Очевидно, що таку перевірку слід проводити лише в тому випадку, якщо дисперсія фактора перевищує дисперсію середніх значень. Якщо ж дисперсія фактора менше дисперсії середніх значень, то впливу фактору на відгук нема.

6.1.2.5 Функція регресії

Якщо відомо, що вплив фактору є, можна спробувати представити залежність, яка нас цікавить, у вигляді деякої функції, близької до експериментальних даних. Графік цієї функції називають лінією регресії.

Вибір функції регресії досить нелегка справа і залежить від мистецтва експериментатора та його досвіду. З одного боку функція регресії не повинна бути складною – це зробить її використання більш трудомістким, з іншого боку вона повинна добре узгоджуватись з експериментальними даними.

Зручніше всього функцію регресії представляти у вигляді лінійної комбінації елементарних функцій, які залежать від величини фактору. В цьому випадку функція регресії буде мати вигляд 6.8.

$$y(x)=a_1\varphi_1(x)+ a_2\ \varphi_2(x)+...+ a_n\ \varphi_n(x) \qquad (6.8)$$

У наведеній залежності невідомими є коефіцієнти a_1, a_2 ,… a_n . Від вибору значень цих коефіцієнтів залежить, який вплив на лінію регресії мають відповідні їм функції φ(x). Наприклад, якщо передбачається, що залежність, яка нас цікавить, описується рівнянням y = ax + b, тоді φ1(w) = w, a φ2(w) =1, і задачею дослідника буде знайти найбільш підходящі значення a1= a і a2= b.

6.1.2.6 Метод найменших квадратів

Коли прийнято рішення про те, які складові будуть входити до складу функції регресії, постає задача визначення значень коефіцієнтів цієї функції. Значення коефіцієнтів повинні бути знайдені такими, щоб лінія регресії пройшла як можна ближче до експериментальних точок. Існують різні методи пошуку коефіцієнтів функції регресії.

У методі найменших квадратів в якості критерію близькості лінії регресії до результатів експерименту приймають суму квадратів відхилень експериментальних точок від відповідних значень лінії регресії. Найкращими значеннями коефіцієнтів вважаються такі, при яких сума квадратів відхилень буде мінімальною.

$$\sum_{i=1}^{n}\left(y_i^{регр}-y_i^{эксп}\right)^2 \to \min \qquad (6.9)$$

Для того, щоб знайти оптимальне значення параметрів, потрібно знайти часткові похідні від функції 6.9 по всім невідомим коефіцієнтам і прирівняти їх до 0. Після цього залишається розв’язати отриману систему лінійних рівнянь відносно невідомих значень параметрів.

В простішому випадку, при одному невідомому коефіцієнті, цільова функція 6.9 прийме вигляд 6.10.

$$\sum_{i=1}^{n}\left(a_1\cdot\varphi_1(x_i)-y_i^{эксп}\right)^2 \to \min \qquad (6.10)$$

Після диференціювання по невідомому коефіцієнту a1, отримаємо рівняння 6.11.

$$a_1 \sum_{i=1}^{p} \left(\varphi_1^2(x_i) \right) = \sum_{i=1}^{p} \left(\varphi_1(x_i) \cdot y_i^{эксп} \right) \tag{6.11}$$

Із цього рівняння неважко знайти вираз 6.12 для визначення коефіцієнта, який ми хочемо знайти.

$$a_1 = \sum_{i=1}^{p} \left(\varphi_1(x_i) \cdot y_i^{эксп} \right) \Big/ \sum_{i=1}^{p} \left(\varphi_1^2(x_i) \right) \tag{6.12}$$

У випадку двох невідомих коефіцієнтів цільова функція 6.9 прийме вигляд 6.14.

$$\sum_{i=1}^{n} \left(a_1 \cdot \varphi_1(x_i) + a_2 \cdot \varphi_2(x_i) - y_i^{эксп} \right)^2 \to \min \tag{6.13}$$

Після диференціювання по невідомим коефіцієнтам a1 і a2, отримаємо систему рівнянь 6.14.

$$\begin{aligned} a_1 \sum_{i=1}^{p} \left(\varphi_1^2(x_i) \right) + a_2 \sum_{i=1}^{p} \left(\varphi_1(x_i) \cdot \varphi_2(x_i) \right) &= \sum_{i=1}^{p} \left(\varphi_1(x_i) \cdot y_i^{эксп} \right) \\ a_1 \sum_{i=1}^{p} \left(\varphi_1(x_i) \cdot \varphi_2(x_i) \right) + a_2 \sum_{i=1}^{p} \left(\varphi_2^2(x_i) \right) &= \sum_{i=1}^{p} \left(\varphi_2(x_i) \cdot y_i^{эксп} \right) \end{aligned} \tag{6.14}$$

Розв’язавши цю систему рівнянь, отримаємо

$$\begin{aligned} a_1 &= \left(\sum_{i=1}^{p} \left(\varphi_1(x_i) \cdot y_i \right) \sum_{i=1}^{p} \left(\varphi_2^2(x_i) \right) - \sum_{i=1}^{p} \left(\varphi_1(x_i) \cdot \varphi_2(x_i) \right) \sum_{i=1}^{p} \left(\varphi_2(x_i) \cdot y_i \right) \right) \Big/ D \\ a_2 &= \left(\sum_{i=1}^{p} \left(\varphi_2(x_i) \cdot y_i \right) \sum_{i=1}^{p} \left(\varphi_1^2(x_i) \right) - \sum_{i=1}^{p} \left(\varphi_1(x_i) \cdot \varphi_2(x_i) \right) \sum_{i=1}^{p} \left(\varphi_1(x_i) \cdot y_i \right) \right) \Big/ D \end{aligned} \tag{6.15}$$

де

$$D = \sum_{i=1}^{p} \left(\varphi_1^2(x_i) \right) \sum_{i=1}^{p} \left(\varphi_2^2(x_i) \right) - \sum_{i=1}^{p} \left(\varphi_1(x_i) \cdot \varphi_2(x_i) \right)^2$$

Ці формули наведені тут тому, що вони реалізовані у класі RegresLinear, але не для того, щоб їх запам’ятовувати.

6.1.2.7 Перевірка адекватності лінії регресії

Параметри функції регресії визначаються виходячи із середніх значень функції відгуку для значень фактора, які аналізуються. Проте лінія регресії, як правило, проходить між точками, які відповідають цим середнім значенням. Відхилення лінії регресії від точок, які використовувались для її отримання, можуть бути любими. Великі відхилення свідчать про те, що функція регресії вибрана невдало і тому лінія регресії неадекватна експериментальним даним.

Для оцінки адекватності лінії регресії використовують дисперсію випадкових величин відхилень середніх значень функції відгуку від лінії регресії, яку можна називати дисперсією неадекватності (Dнад).

$$D_{над} = \frac{\sum_{i=1}^{p} (y_i^{ср} - y_i^{рґр})^2}{p-k} \qquad (6.16)$$

де $y_{iср}$ – середнє значення функції відгуку на і-му рівні;

$y_{iргр}$ – значення функції на лінії регресії для і-го рівня;

p – число рівнів зміни фактора;

k – число коефіцієнтів функції регресії, які були знайдені за значеннями $y_{ср}$.

Кількість степенів свободи для дисперсії, отриманої за цією формулою дорівнює p-k.

Лінію регресії вважають неадекватною в тому випадку, коли дисперсія неадекватності набагато більше дисперсії середніх значень функції відгуку, яка може біти знайдена із дисперсії експерименту $D_{эксп}$, за формулою 6.7.

Число степенів свободи дисперсії, отриманої за цією формулою, таке ж, як і у дисперсії експерименту, і дорівнює p*(n-1).

Для порівняння дисперсій можна використовувати критерій Фішера.

Лінія регресії може вважатися неадекватною, якщо відношення дисперсії неадекватності до дисперсії середніх значень функції відгуку більше критичного значення критерію Фішера. Очевидно, що таку перевірку слід проводити лише в тому випадку, коли дисперсія неадекватності перевищує дисперсію середніх значень. Якщо ж дисперсія неадекватності менше дисперсії середніх значень, то лінію регресії можна вважати адекватно.

6.2 Засоби для автоматизації проведення одно факторних експериментів та обробки їх результатів

6.2.1 *Компонент ExperimenManager*

В пакеті widgets.experiments міститься клас ExperimentManager, який надає розробнику імітаційних моделей візуальний компонент для автоматизації проведення однофакторних експериментів, накопичення результатів цих експериментів і подальшого дисперсійного та регресійного аналізу.

Компонент передбачає технологію, відповідно до якої модель створюється, ініціалізується та запускається самим компонентом.

Необхідною умовою для роботи компонента є передача йому посилання на фабрику моделей, що реалізує інтерфейс IModelFactory. Зв'язок компонента ExperimentManager з фабрикою моделей налаштовується через метод setFactory(IModelFactory).

Створивши модель, компонент ExperimentControl приводить її до типу widgets.experiments.IExperimentable. Це інтерфейс, через який компонент ExperimenManager буде працювати моделлю. Тобто модель має реалізовувати такий інтерфейс. Перелік методів інтерфейсу наведено у лістингу 6.1.

Лістинг 6.1 – Інтерфейс IExperimentable

```
public interface IExperimentable {
        public void initForExperiment(double factor);
        public Map<String, Double> getResultOfExperiment();
}
```

Метод initForExperiment(double) викликається перед кожним запуском моделі і має забезпечити підготовку моделі до однократного запуску. В якості параметра у метод передається значення фактору, вплив якого вивчається.

Метод getResultOfExperiment() використовується для отримання результатів експерименту після його закінчення. Кожен елемент колекції, що повертає цей метод, в якості ключа містить текст, що ідентифікує результат експерименту, а значенням є сам результат. Таким чином метод може повертати будь яку кількість відгуків на задане значення фактору.

Для запуску моделі компонент буде використовувати метод диспетчера start().

Візуальна композиція компонента представлена на рисунку 6.1.

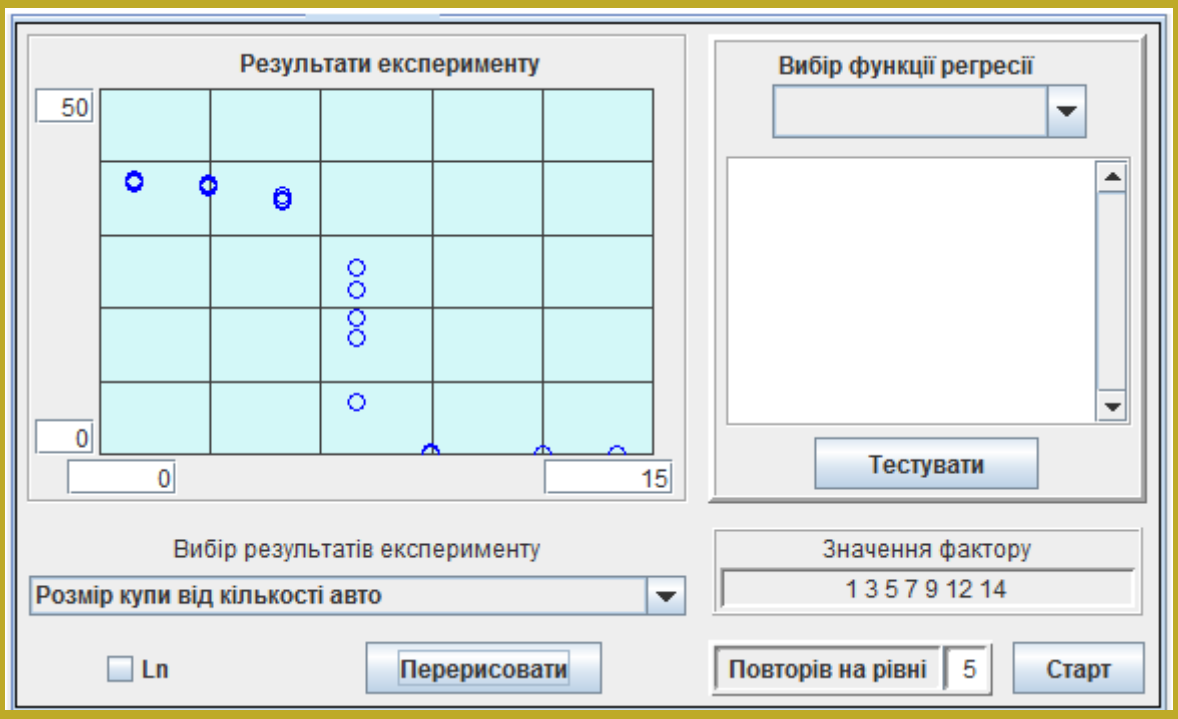

Рисунок 6.1 – Компонент для автоматизації проведення експериментів

Компонент можна використовувати як елемент інтерфейсу користувача у застосуваннях для імітаційного моделювання.

Значення фактору, для яких потрібно провести експерименти, задаються у вигляді рядка символів в полі «Значення фактору:». Числові значення факторів повинні бути розділені стандартними розділовими знаками.

Кількість експериментів, що мають бути проведені для кожного значення фактору, задається у полі «Повторів на рівні:».

Кнопка «Старт» викликає метод buttonStartClick(), який забезпечує проведення заданої кількості експериментів на кожному рівні. Під час проведення експериментів на діаграмі відображаються точки, які відповідають отриманим результатам, що забезпечує динамічну індикацію ходу експерименту.

На час проведення експерименту кнопка «Старт» блокується.

Результати експериментів можуть бути про логарифмовані. Для цього використовується перемикач «Ln».

Розглянемо, як компонент організує проведення експериментів з моделлю.

Для проведення експериментів створюється пул потоків, у кількості, що відповідає кількості ядер процесора.

Далі в методі організовано подвійний цикл, що забезпечує проведення потрібної кількості експериментів. Для кожного експерименту створюється окремий потік, який передається у пул для виконання.

Кожний з цих потоків перш за все, за допомогою фабрики, створює модель. Далі, через метод initForExperiment() моделі передається значення фактору і модель активізується за допомогою методу диспетчера start().

Про закінчення експерименту кожний потік дізнається з події DispatcherFinishEvent. Результати експерименту потік отримує за допомогою методу getResultOfExperiment(), який повертає колекцію результатів з їх назвами. Ці результати заносяться до іншої колекції, що зберігає результати усіх експериментів.

Кількість проведених експериментів контролюється за допомогою об’єкту класу CountDownLatch.

Коли усі експерименти проведено, формується перелік рядків для компоненту ComboBox і перша із отриманих залежностей виводиться на діаграму. Інші залежності можна переглянути, вибираючи відповідній рядок компоненту ComboBox.

До складу компоненту ExperimentManager включено також компонент класу RegresAnaliser за допомогою якого можна підібрати функцію регресії для отриманих результатів і протестувати її на адекватність

Компонент може працювати або у режимі однорівневого експерименту, або у режимі багаторівневого експерименту.

У першому випадку компонент видає інформацію про довірчий інтервал для результатів експерименту і графічно відображає його розміри на фоні експериментальних даних.

У режимі багаторівневого експерименту, компонент дозволяє вибрати функцію регресії і знайти її параметри і отримати графічне відображення. Окрім того надається інформація про результати перевірки на однорідність дисперсій, вплив фактору та адекватність функції регресії.

6.2.2 *Компонент класу RegresAnaliser*

В пакеті widjets.regres міститься клас RegresAnaliser, який надає розробникам імітаційних моделей візуальний компонент для регресійного та

дисперсійного аналізу результатів експериментів. Компонент можна використовувати при розробці інтерфейсу користувача у застосуваннях для імітаційного моделювання.

Передбачається, що даний компонент буде працювати з компонентами, які реалізують інтерфейс IRegresable. Зокрема, таким компонентом є компонент класу ExperimentControl. Зв'язок компонента з об'єктами, які реалізують інтерфейс IRegresable налаштовується через метод setIRegresable (IRegresable).

Опис інтерфейсу IRegresable представлений у лістингу 6.2. Методи цього інтерфейсу забезпечують доступ до експериментальних даних.

Лістинг 6.2 – Інтерфейс RegresExperimentable

```
public interface IRegresable {
    public double[] getFactorsArray();
    public double[][] getResultMatrix();
}
```

Для отримання результатів аналізу у графічному вигляді компоненту слід передати посилання на діаграму.

Компонент може працювати або у режимі однорівневого експерименту, або у режимі багаторівневого експерименту.

У першому випадку компонент видає інформацію про довірчий інтервал для результатів експерименту і графічно відображає його розміри на фоні експериментальних даних.

У режимі багаторівневого експерименту, компонент дозволяє вибрати функцію регресії і знайти її параметри і отримати графічне відображення. Окрім того надається інформація про результати перевірки на однорідність дисперсій, вплив фактору та адекватність функції регресії. Приклади використання компоненту зображені на рисунках 6.2, 6.3.

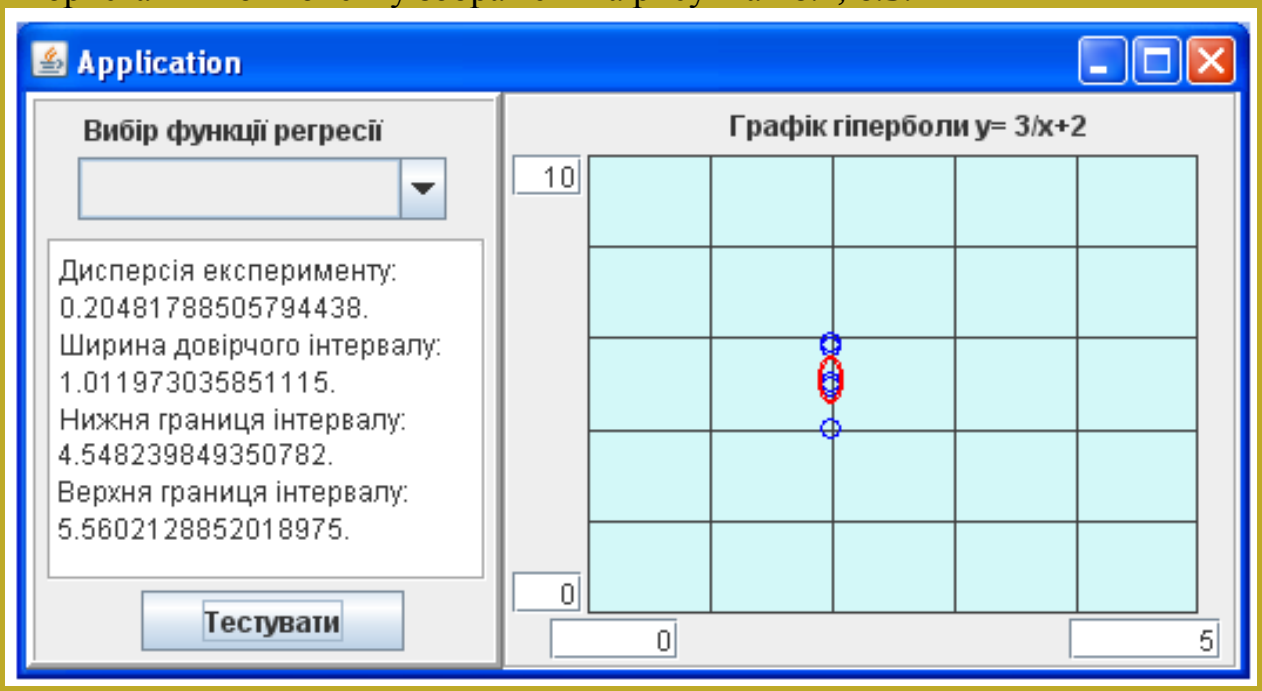

Рисунок 6.2 – Приклад використання RegresAnaliser для аналізу результатів однорівневого експерименту

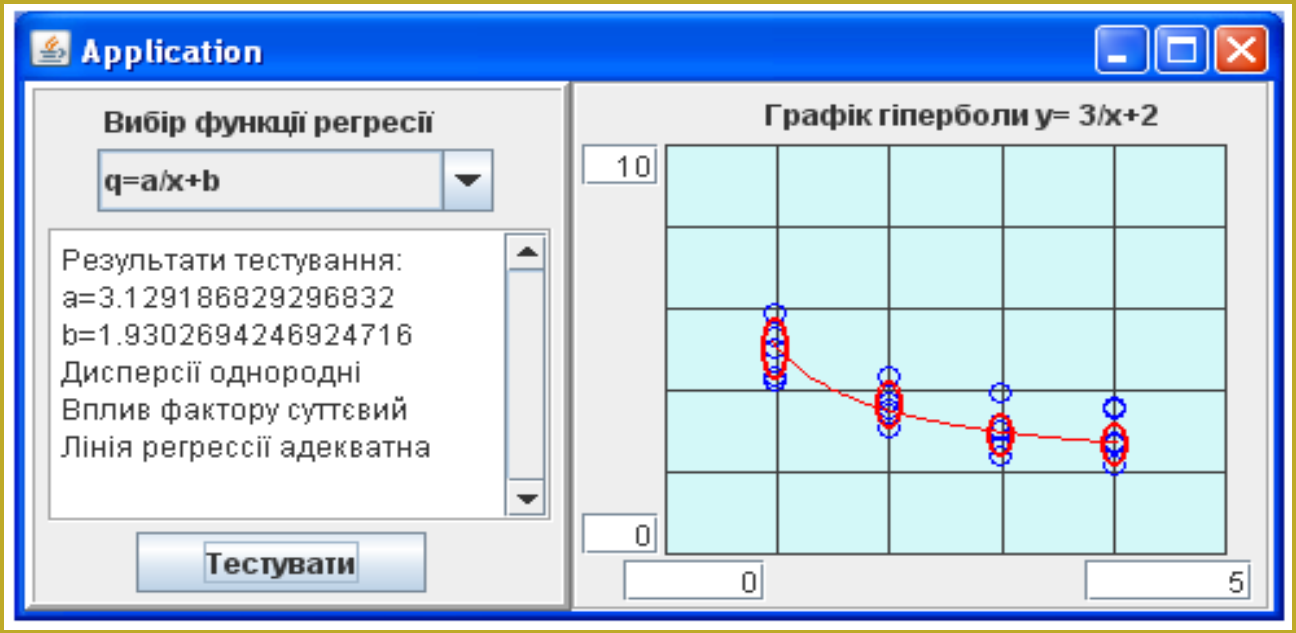

Рисунок 6.3 – Приклад використання RegresAnaliser для аналізу результатів багаторівневого експерименту

Компонент RegresAnaliser являє собою тільки візуальну оболонку. За обробку та аналіз даних відповідає ієрархія класів з базовим абстрактним класом RegresTesters, які також містяться в пакеті widjets.regres. Саме вони забезпечують дисперсійний та регресійний аналіз отриманих результатів.

6.2.2.1 Клас RegresTesters

Ієрархія класів, що забезпечують дисперсійний та регресійний аналіз результатів моделювання та публічні методи класу RegresTesters показані на рисунку 6.4.

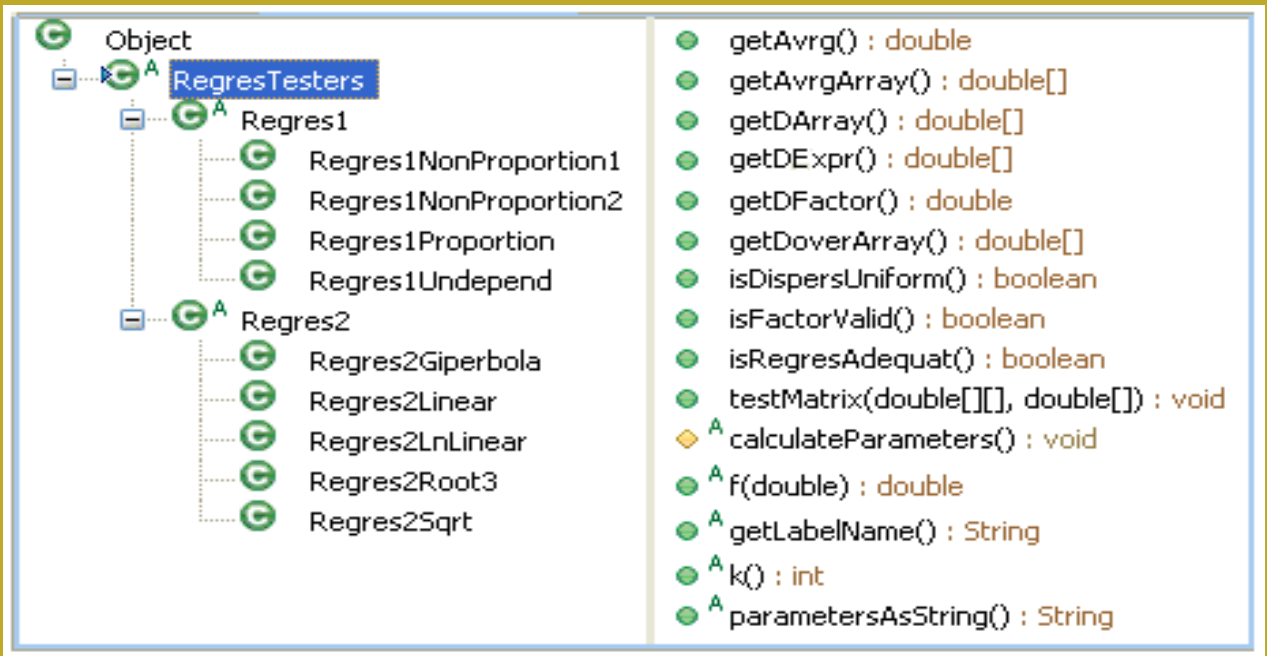

Рисунок 6.4 – Діаграма класу RegresTesters

Обробка даних ініціюється викликом методу TestMatrix класу RegresTesters. Через параметри цього методу на обробку надходять матриця результатів експериментів matrix та масив значень факторів factorArray.

Виклик методу призводить до розрахунку статистичних характеристик отриманих даних. Отримані характеристики запам'ятовуються у полях класу RegresTesters, які перераховані нижче.

p – кількість рівнів зміни фактора.

n – кількість експериментів на рівні.

avrgArray – масив середніх значень відгуку для кожного рівня.

dArray – масив дисперсій відгуку на кожному рівні.

doverArray – масив довірчих інтервалів на рівнях.

avrg – загальне середнє.

dExpr – дисперсія експерименту.

dFactor – дисперсія фактору.

dAdequat – дисперсія адекватності.

Доступ до цих даних можливий у підкласах, а також реалізується через методи get…() цього класу.

У методі використовуються поліморфні методи calculateParameters(), f() та k(). В базовому класі ці методи визначені як абстрактні, а конкретна їх реалізація знаходиться у підкласах.

Метод calculateParameters() забезпечує обчислення параметрів функції регресії для відповідного класу.

Метод f(double) використовується для розрахунку значень функції регресії.

Метод k() повертає кількість параметрів рівняння регресії.

Клас RegresTesters має двох абстрактних спадкоємців – класи Regres1 та Regres2.

6.2.2.2 Ієрархія класів Regres1

Клас Regres1 забезпечує отримання рівняння регресії у вигляді f=a*φ1(x), де φ1(x) – довільна функція, від якої залежить вигляд лінії регресії. Для забезпечення розрахунків значень функції регресії відповідно до наведеної формули у класі реалізовано метод f(double), що був визначений у суперкласі як абстрактний. Цей метод, у свою чергу, використовує абстрактний метод fi1(double), який має бути реалізований у підкласах, і забезпечувати обчислення значень функції φ1(x).

Реалізовано також метод k(), який повертає 1, виходячи з того, що функція регресії має лише один параметр – a.

Визначення невідомого коефіцієнту **a** в цьому класі забезпечено реалізацією метода calculateParameters(), у якому реалізовано розрахунок за формулою 6.13, яка була отримана вище.

Клас Regres1 має декілька спадкоємців, у яких реалізовано конкретний вигляд функції φ1(x).

Клас Regres1Undepend реалізує функцію φ1(x) у вигляді φ1(x) = 1. В результаті виходить, що клас RegresUndepend реалізує визначення параметрів функції регресії виду f=a. Це випадок, коли відгук не залежить від фактору.

Клас Regres1Proportion використовується для створення об'єктів, які забезпечують отримання рівняння регресії у вигляді f=a*x. Для цього в класі перевизначений метод fi1(double), який реалізує функцію φ1(x) у вигляді φ1(x) = x.

Клас Regres1NonProportion1 використовується для створення об'єктів, які забезпечують отримання рівняння регресії у вигляді f=a*x/(1-x). Для цього в класі перевизначений метод fi1(double), який реалізує функцію φ1(x) у вигляді φ1(x) = x/(1-x).

В класі Regres1NonProportion2 метод fi1(double), реалізує функцію φ1(x) у вигляді φ1(x) = x*x/(1-x). В результаті забезпечується отримання рівняння регресії у вигляді f=a*x*x/(1-x)

6.2.2.3 Ієрархія класів Regres2

Клас Regres2 наслідує клас RegressTesters і містить методи, які забезпечують отримання рівняння регресії у вигляді f=a*φ1(x)+b*φ2(x), де φ1(x) і φ2(x) – будь-які функції, які залежать від фактору і визначають вигляд лінії регресії. Для забезпечення розрахунків значень функції регресії відповідно до наведеної формули у класі реалізовано метод f(double), що був визначений у суперкласі як абстрактний. Цей метод, у свою чергу, використовує абстрактні методи fi1(double) та fi2(double), які мають бути реалізовані у підкласах, і забезпечувати обчислення значень функцій φ1(x) та φ2(x).

У класі реалізовано свій варіант методу k(), який повертає 2, виходячи з того, що функція регресії має два параметри – a і b.

Розрахунок коефіцієнтів a і b забезпечує реалізація методу calculateParameters(), який використовує формули 6.16, що були отримані вище.

Клас Regres2 має декілька спадкоємців, у яких реалізовано конкретний вигляд функцій φ1(x) та φ2(x).

Клас Regres2Linear реалізує визначення параметрів функції регресії виду f=a*x+b. В цьому класі метод fi1(double) реалізує функцію φ1(x) у вигляді φ1(x) = x, а функцію φ2(x) у вигляді φ2(x) = 1.

Клас Regres2Giperbola реалізує визначення параметрів функції регресії виду f=a/x+b. В цьому класі метод fi1(double) реалізує функцію φ1(x) у вигляді φ1(x) = 1/x, а функцію φ2(x) у вигляді φ2(x) = 1.

6.2.2.4 Розширення переліку функцій регресії компоненту RegresAnaliser

Щоб додати до компоненту RegresAnaliser ще якусь функцію регресії, необхідно створити клас, що успадковує клас RegresTesters, і реалізувати у ньому абстрактні методи суперкласу. Завдання спрощується, якщо нова функція регресії має вигляд, що передбачений класами Regres1 або Regres2. У цьому випадку достатньо успадкувати один з цих класів та реалізувати у ньому метод fi1(double), бо методи fi1(double) та fi2(double).

Створивши клас, що відповідає за нову функцію регресії, слід створити об'єкт цього класу і передати його компоненту RegresAnaliser за допомогою методу addFunction(RegresTesters).

6.3 Огляд проекту для лабораторної роботи

В лабораторній роботі досліджується найпростіша марківська система масового обслуговування. До складу моделі входять генератор заявок, черга і обслуговуючий прилад. Лабораторне застосування проектувалась виходячи з того, що воно повинно забезпечити:

- автоматичне проведення серій експериментів для заданого набору значень фактора;
- індикацію ходу проведення експериментів і їх результатів;
- накопичення інформації про результати експериментів;
- статистичну обробку результатів експериментів.

6.3.1 *Класи пакету testFactorExperiment*

Цей пакет містить класи, безпосередньо пов'язані з виконанням лабораторної роботи. До складу пакету входять три класи.

Клас FactorGUI реалізує інтерфейс користувача і виконує функції шару подання у застосуванні.

Клас Model реалізує інтерфейс IExperimentable є шаром моделі застосування.

Шар компонентів застосування представлений такими об'єктами:

- transactGenerator класу TransactGenerator, генератор трансакцій;
- queue классу QueueForTransactions, що моделює чергу транзакцій;
- device класу Device, обслуговуючий прилад
- discretHisto класу DiscretHisto, гістограма для довжини черги;
- histoTransactionWaitInQueue класу stat.Histo, гістограма для збору інформації про час очікування в черзі;

– histoWaitDevice класу stat.Histo, гістограма для збору інформації про час чекання для обслуговуючого пристрою.

Для об’єктів transactGenerator та device у пакеті створено класи, що реалізують поведінку генератора трансакцій та обслуговуючого пристрою.

Генератор трансакцій моделі створює заявки незалежно від максимального розміру черги. Заявки, що не приймаються чергою, втрачаються.

6.3.1.1 Клас FactorGUI

Клас FactorGUI реалізує інтерфейс користувача. Клас створений як модифікацію візуального класу із попередньої лабораторної роботи і відрізняється від нього наявністю додаткової закладки, на якій розташований компонент ExperimentManager. Візуальну композицію класу зображено на рисунку 6.5.

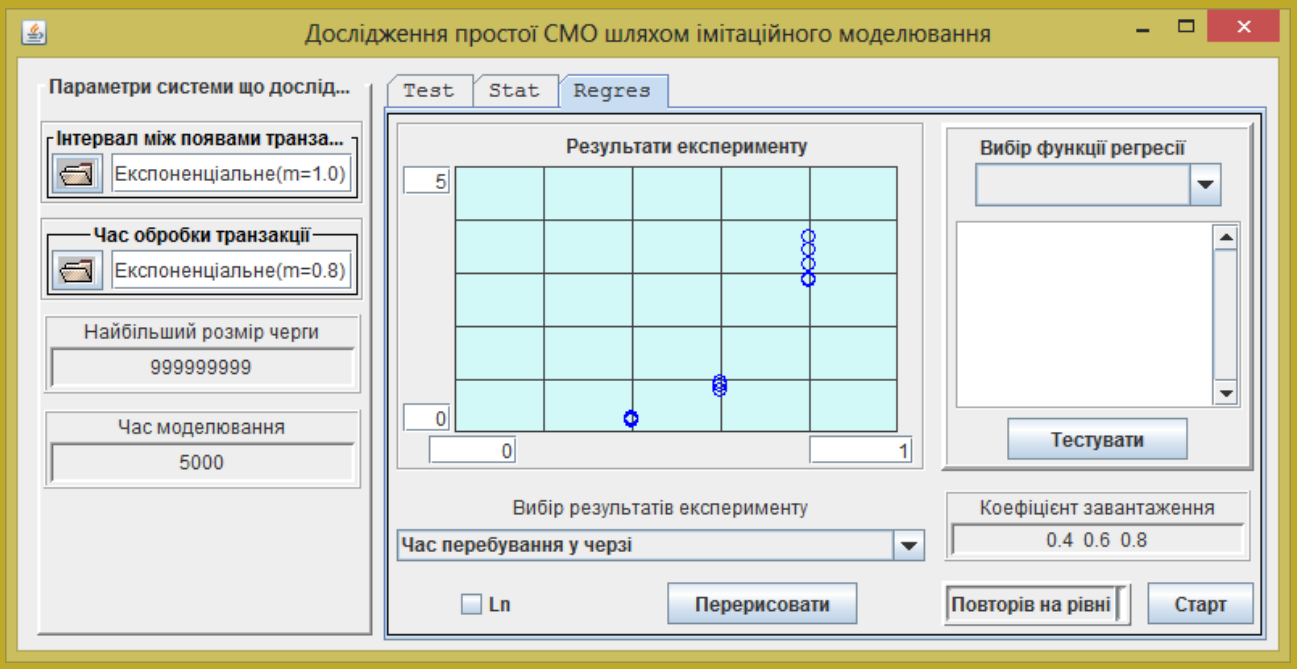

Рисунок 6.5 – Візуальна композиція класу ExperimentUI

Клас створено майже повністю за допомогою візуального редактора. Єдине, що потребувало програмування – це передача посилань на діаграму та фабрику моделей компоненту ExperimentControl та посилання на діаграму компоненту RegresAnaliser.

6.3.1.2 Клас Model

Клас Model задає структуру моделі і забезпечує створення та ініціалізацію її компонентів. За основу класу взято клас моделі із попередньої лабораторної роботи. Додатково у класі реалізовано інтерфейс IExperimentable, що дозволяє підключати її до компонента ExperimentManager. В лістингу 6.3 наведено реалізації методів інтерфейсу IExperimentable в даному класі.

Лістинг 6.3 – Реалізація інтерфейсу IExperimentable у класі Model

```
public void initForExperiment(double factor) {
        double mGen = gui.getChooseRandomGen().average();
        getDevice().setRnd(new Negexp(factor * mGen));
}

public Map<String, Double> getResultOfExperiment() {
        Map<String, Double> results = new HashMap<>();
        results.put("Середня довжина черги", getDiscretHisto().getAverage());
        results.put("Час перебування у черзі",
                                getHistoTransactionWaitInQueue().getAverage());
        results.put("Доля простою обслуговуючого пристрою",
                                getHistoWaitDevice().getAverage());
        return results;
}
```

6.3.1.3 Клас TransactionGenerator

Цей клас нічим не відрізняється від такого ж класу, що розглядався у попередній роботі.

6.3.1.4 Клас Device

Цей клас відрізняється від такого ж класу, що розглядався у попередній роботі, тільки тим, що у ньому визначено метод setRnd(Randomable), через який передається посилання на генератор випадкових чисел, що визначає тривалість обробки транзакції. Цей генератор створюється у моделі, відповідно до значення коефіцієнту завантаження, який є фактором експерименту, що проводиться.

6.4 Порядок виконання роботи

6.4.1 *Підготовка до роботи*

– Активізуйте Java застосування, що знаходиться у архіві SimulationAllLab.jar, або завантажте проект SimulationAllLab.zip до Eclipse.

6.4.2 *Перевірка працездатності програмного комплексу*

– Активізувавши застосування «Лабораторна робота 6» переконайтеся в працездатності програмного комплексу.

6.4.3 *Дослідження структури програмного комплексу*

Ознайомтесь із класами проекту TestFactorExperiment. Подивіться, як побудована модель системи. Прослідкуйте за послідовністю запуску моделі. Ознайомтесь із класами пакетів widgets.experiment та widgets.regres.

6.4.4 *Експерименти з моделлю на одному рівні фактору*

При виконанні цього пункту у полі «Значення фактору:» потрібно задавати лише одне значення фактору. В цьому випадку, при натисненні на кнопку «Тестувати» в текстовому вікні виводиться інформація про величину 95% довірчого інтервалу, розрахованого за результатами експериментів.

Значення фактору обирайте в діапазоні від 0.40 до 0.99, у відповідності з двома останніми цифрами номера залікової книжки. Якщо отримане число менше 0.40, додаємо до нього 0.4.

6.4.4.1 Дослідження впливу об'єму вибірки на ширину довірчого інтервалу

Проведіть експерименти з моделлю для свого значення фактору при кількості експериментів на одному рівні 5, 30 і 100, і часі моделювання рівному 1000. Результати занесіть до таблиці 6.1.

Визначте середнє значення ширини інтервалу для кожного стовпчика.

Зробіть копію екрану з результатами одного з експериментів для звіту.

Побудуйте графік залежності середньої ширини довірчого інтервалу від числа експериментів.

Зробіть висновок про вплив об'єму вибірки на ширину інтервалу.

Таблиця 6.1 – Границі довірчих інтервалів при часі моделювання рівному 1000 і коефіцієнті завантаження системи …..

№ Досліду	Число експериментів на рівні								
	5			30			100		
	Нижня границя	Верхня границя	Ширина	Нижня границя	Верхня границя	Ширина	Нижня границя	Верхня границя	Ширина
1									
2									
3									
Середня ширина									

6.4.4.2 Дослідження впливу часу моделювання на ширину довірчого інтервалу

Проведіть серію аналогічних експериментів при кількості експериментів на одному рівні, яка дорівнює 30, і часі моделювання рівному 5000 і 10000. Результати занесіть до таблиці 6.2. Порівняйте результати і зробіть висновок про вплив часу моделювання на точність отримуваних результатів.

Таблиця 6.2 – Границі довірчих інтервалів для 30 експериментів на рівні, і коефіцієнті завантаження системи …..

№ Досліду	Час моделювання								
	1000			5000			10000		
	Нижня границя	Верхня границя	Ширина	Нижня границя	Верхня границя	Ширина	Нижня границя	Верхня границя	Ширина
1									
2									

3									
Середня ширина									

Побудуйте графік залежності середньої ширини довірчого інтервалу від часу моделювання.

Зробіть висновок про вплив часу моделювання на точність отримуваних результатів.

6.4.5 *Експерименти з моделлю на декількох рівнях фактору*

– Проведіть експеримент з моделлю для набору коефіцієнтів завантаження системи 0.5 0.7 0.9 і коефіцієнта, який відповідає номеру залікової книжки, при кількості експериментів на одному рівні 10, і часі моделювання рівному 500.

– Знайдіть адекватну лінію регресії для отриманих експериментальних даних.

– Зверніть увагу на результат аналізу дисперсій. Дисперсії, як правило, виходять неоднорідними через велику різницю середніх значень на рівнях.

– Результат, у вигляді копії вікна, зафіксуйте у звіті.

6.4.6 *Вирівнювання дисперсій*

– Налаштуйте модель таким чином, щоб при регресійному аналізі оброблялися логарифми від середньої довжини черги і повторіть обробку. Знайдіть адекватну лінію регресії для отриманих експериментальних даних. Майте на увазі, що значення логарифмів можуть бути від'ємними, тому зробіть відповідні налаштування діаграми.

– Зверніть увагу на результат аналізу дисперсій. Дисперсії, як правило, виходять однорідними, бо різниця середніх значень на рівнях істотно зменшується при логарифмуванні.

– Результат, у вигляді копії вікна, зафіксуйте у звіті.

6.4.7 *Експерименти з моделлю при обмеженій довжині черги*

Перед проведенням цих експериментів змініть значення максимальної довжини черги у об'єкта з максимально можливого до 4.

– Проведіть експеримент з моделлю для набора коефіцієнтів завантаження системи 0.1 0.4 0.8 1.2 1.6 2.0 при кількості експериментів на одному рівні 10, часі моделювання рівному 500.

– Спробуйте знайти найбільш підходящу лінію регресії для отриманих експериментальних даних.

– Вигляд отриманої залежності досить складний. Але можна розбити весь діапазон від 0.1 до 2 на три ділянки і для кожної підібрати окрему лінію регресії. Але для цього потрібно створити додаткові класи.

6.4.8 *Розширення переліку ліній регресії*

– Створіть класи, які забезпечать пошук рівняння регресії для початкової і кінцевої ділянки залежності, отриманої в попередньому експерименті. Для початкової ділянки реалізуйте рівняння виду q=a*exp(w)/(1-w)+b, а для кінцевої у вигляді q=a*ln(w)/√(w)+b.

– Додайте ці класи до переліку функцій компоненту RegresAnaliser класу GUI. Щоб додати нові функції до компоненту модифікуйте метод getRegresAnaliser() класу GUI, використовуючи для розширення набору функцій метод addFunction(newRegresTesters).

– Проведіть експерименти для кожної ділянки окремо і перевірте можливість використання нових функцій регресії для крайніх ділянок. Середню частину можна представити прямою.

– Результати у вигляді копій вікон для кожної ділянки зафіксуйте у звіті.

6.4.9 *Поради до виконання завдання по розширенню набору ліній регресії*

– створити новий Java проект та пакет, наприклад, lab6.

– скопіювати класи із пакету testFactorExperiment у пакет lab6.

– пов’язати проект із фреймворком Simulation

– створити у пакеті ще два класи RegresExp та RegresLn, що успадковують клас Regres2. Ці класи забезпечать пошук параметрів “a” і “b” для нових функцій регресії. У класах реалізувати методи fi1 та fi2

– доопрацювати метод getRegresAnaliser() класу GUI, використовуючи для розширення набору функцій метод addFunction(newRegresTesters).

6.5 Завдання для самостійної роботи

Реалізуйте в моделі для свого варіанту РГР інтерфейс IExperimentable, а до візуальної частини додайте закладку з компонентом ExperimentManager і виконайте необхідні налаштування.

Перевірте роботу застосування у режимі проведення однофакторних багаторівневих експериментів.

6.6 Зміст звіту

– Назва і ціль роботи.
– Копії екранів з результатами експериментів.
– Таблиця із результатами експериментів.
– Графічне відображення результатів експериментів.
– Тексти класів, що за рахунок яких було розширено перелік функцій регресії.
– Результати виконання завдання для самостійної роботи.
– Висновки по результатам виконання лабораторної роботи.

Контрольні запитання

1. Як знайти довірчий інтервал.
2. Як перевірити дисперсію на однорідність.
3. Як можна досягти однорідності дисперсій.
4. Як обчислити дисперсію експерименту.
5. Як обчислити дисперсію фактору.
6. Як оцінити значимість впливу фактору на функцію відгуку.
7. Метод найменших квадратів.
8. Як обчислити дисперсію адекватності.
9. Описати роботу об’єкта ExperimentControl.
10. Описати роботу об’єкта RegresAnaliser.
11. Як розширити перелік функцій регресії об’єкта RegresAnaliser.

7 ЛАБОРАТОРНА РОБОТА № 7. ДОСЛІДЖЕННЯ ПЕРЕХІДНИХ ПРОЦЕСІВ У СИСТЕМАХ МАСОВОГО ОБСЛУГОВУВАННЯ

Мета роботи

- Ознайомитися з методикою аналізу перехідних процесів у СМО шляхом імітаційного моделювання та методами обробки отриманих експериментальних даних.
- Ознайомитися з методами пошуку екстремумів функцій однієї змінної.

7.1 Короткі теоретичні відомості

7.1.1 *Перехідний процес і сталий режим у СМО*

СМО відносяться до динамічних систем і, отже, при вивченні цих систем слід розрізняти сталий режим роботи (для реальних СМО він зазвичай рано чи пізно настає) і перехідний процес. Так, наприклад, черга в простішій СМО спочатку дорівнює 0, а через деякий час починає змінюватись навколо деякого середнього значення. Перші заявки у системі чекають на обслуговування недовго, навіть при великому завантаженні системи, а для заявок, які з'явилися пізніше, середній час очікування може бути досить великим.

Досліджувати перехідні процеси аналітично, як правило, не вдається через складність диференціальних рівнянь, що описують поведінку системи в перехідному режимі. Проте, ці режими є інтересними для практики. Перш за все, вони цікаві для систем, які працюють недовго, і не встигають досягти сталого режиму. Але навіть при вивченні сталих режимів, інформація про тривалість перехідного процесу корисна, оскільки дозволяє визначити момент часу, з якого слід почати збирати інформацію про сталий режим. Таким чином, проблема побудови моделей, що дозволяють оцінити характер перехідних процесів шляхом моделювання, є досить актуальною.

7.1.2 *Методика отримання експериментальних даних про перехідний процес*

Під час спостереження за роботою однієї СМО важко встановити закономірності, характерні для перехідних процесів. Це пов'язано із сильними флуктуаціями змінної, що нас цікавить, наприклад, довжини черги. Але якщо одночасно запустити багато паралельно працюючих моделей, і спостерігати за змінами усередненого по всіх моделях значення параметру, то вплив флуктуацій буде зменшено, і будуть виявлені закономірності характерні для перехідного режиму.

Маючи засоби для моделювання псевдопаралельних процесів, вирішити задачу виявлення перехідного процесу достатньо легко. Для цього можна одночасно моделювати роботу декількох однакових систем і проводити усереднення змінної, яка нас цікавить. Чим більше таких систем буде

паралельно працювати, тим менш помітними будуть флуктуації, і тим виразніше будуть виявлятися закономірності перехідного процесу. Надійність результатів, одержаних при цьому, залежить тільки від потужності комп'ютера і часу, виділеного на проведення експериментів.

Таким чином можна запропонувати наступну методику оцінки параметрів перехідного процесу для середньої довжини черги. Створюється велика кількість однакових моделей, які одночасно стартують і паралельно працюють у віртуальному модельному часі, єдиному для всіх моделей. Кожна з моделей на невеликому відрізку часу (інтервал накопичення або дискретизації) збирає поточну інформацію про довжину черги і наприкінці цього відрізку часу визначає середнє значення черги на цьому інтервалі, реалізуючи таким чином усереднення у часі (рисунок 7.1).

Рисунок 7.1- Усереднювання значень довжини черги за часом у межах однієї реалізації

Послідовність середніх значень, отриманих у ході експерименту, не дає уявлення про характер перехідного процесу, але дозволяє провести дискретизацію процесу. Усереднення ж результатів, отриманих на кожному інтервалу накопичення, по всіх реалізаціях дозволяє отримати послідовність середніх значень довжини черги для кожного інтервалу, що дасть уявлення про характер перехідного процесу (рисунок 7.2).

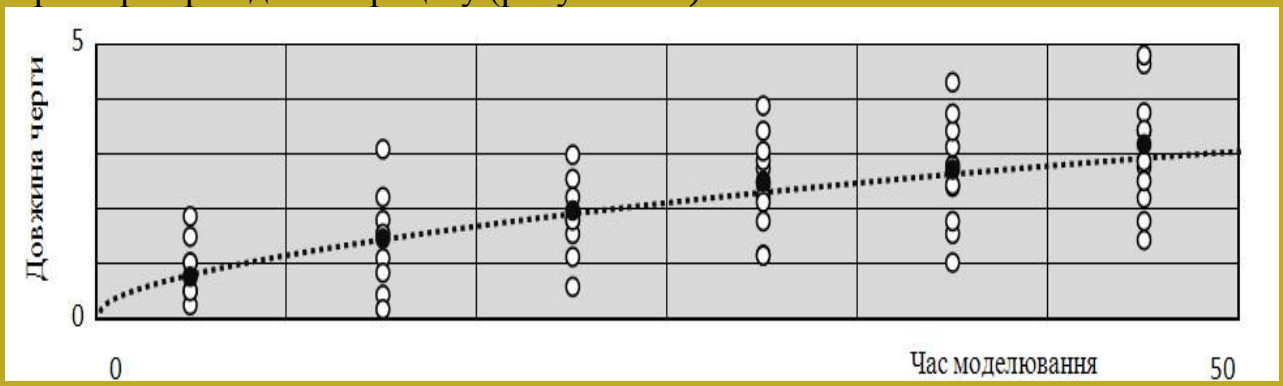

Рис. 7.2. Результати усереднювання середньої довжини черги по реалізаціях

Інтервал накопичення може бути як-завгодно малим, але в його суттєвому зменшенні нема потреби, якщо у перехідному процесі нема високочастотних складових.

В прикладі, який розглядався, зміна характеристики СМО (довжина черги) була пов’язана із часом. Проте таке можливо не завжди.

Наприклад, якщо досліджувати перехідний процес для часу очікування в черзі, то його потрібно пов'язувати не з часом, а з номером заявки. Тоді, в результаті усереднення по реалізаціях, буде отримана залежність середнього часу очікування в черзі від номера заявки. Але такий варіант тут не розглядається.

7.2 Засоби для дослідження перехідних процесів

Класи та інтерфейси, які можуть бути використані під час дослідження перехідних процесів у різних моделях, в тому числі й під час самостійної роботи над РГР, знаходяться у пакеті widgets.trans бібліотеки Simulation.

7.2.1 *Інтерфейс ITransMonitoring*

У цьому інтерфейсі оголошено методи, що використовує компонент TransProcessManager для роботи з чергами, для яких досліджується перехідний процес.

Інтерфейс має два методи.

Метод resetAccum() використовується для ініціалізації накопичувача інформації і викликається на початку інтервалу накопичення.

getAccumAverage() повертає середнє значення черги на інтервалі накопичення і викликається по завершенню інтервалу накопичення.

Інтерфейс реалізовано у класах QueueForTransactions та Store.

7.2.2 *Компонент TransProcessManager*

Цей компонент забезпечує дослідження перехідних процесів у чергах. Зовнішній вигляд компоненту наведено на рисунку 7.3.

Інформація про перехідний процес формується шляхом усереднення розміру черг за часом та по реалізаціям для великої кількості паралельно працюючих моделей.

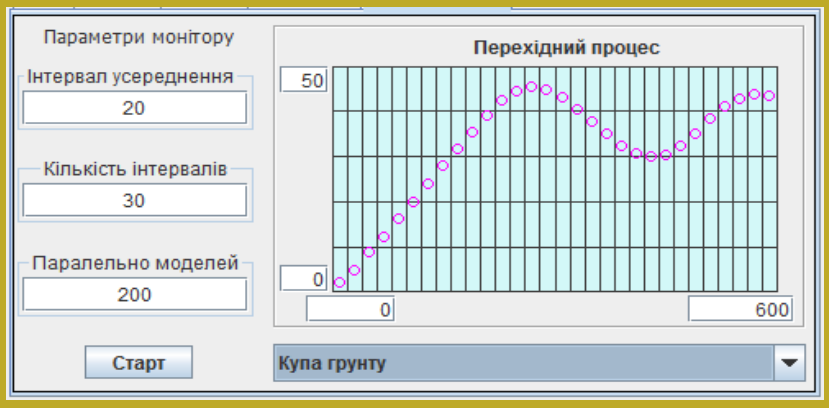

Рисунок 7.3 – Вигляд компоненту класу TransProcessManager

Так само як у випадку компоненту ExperimentManager тут використовується технологія, відповідно до якої моделі створюються, ініціалізуються та запускаються самим компонентом.

Необхідною умовою для роботи компонента є передача йому посилання на фабрику моделей, що реалізує інтерфейс IModelFactory. Зв'язок компонента ExperimentManager з фабрикою моделей налаштовується через метод setFactory(IModelFactory).

Створивши модель, компонент TransProcessManager приводить її до типу widgets.trans.ITransProcesable. Це інтерфейс, через який компонент TransProcessManager буде працювати з моделлю і модель має реалізовувати такий інтерфейс. Інтерфейс ITransProcesable передбачає реалізацію двох методів.

Метод void initForTrans(double finishTime) використовується для ініціалізації компонентів моделі. Як параметр до методу передається тривалість моделювання.

Метод Map<String, ITransMonitoring> getMonitoringObjects() надає компонентові перелік черги, що досліджуються.

Кожен елемент колекції, що повертає цей метод, в якості ключа містить текст, що ідентифікує чергу, а значенням є сама черга. Таким чином компонент може працювати з будь якою кількістю черг.

На підставі переліку ключових значень елементів отриманої колекції результатів створюється модель для компоненту JComboBox, який дозволяє викликати для перегляду потрібний перехідний процес.

Візуальна частина компоненту, що була показана на рисунку 7.3, забезпечує його налаштування, а також запуск процесу моделювання.

Поле "Інтервал усереднення" визначає довжину інтервалу накопичення інформації.

Поле "Кількість інтервалів" визначає кількість інтервалів накопичення.

У полі "Паралельно моделей" задається кількість паралельно працюючих систем.

Крім візуальної частини до складу компонента входять поля, які містять посилання на такі об'єкти:

monitor – монітор, об'єкт класу TransMonitor, що наслідує клас Actor, і в ньому визначений метод rule(), де описано правила дії, що забезпечують

проведення експерименту. Об’єкт створюється у компоненті.

factory – посилання на фабрику моделей, що реалізує інтерфейс IModelFactory, за допомогою якої будуть створюватися моделі.

diagram – посилання на компонент класу Diagram, що забезпечує відображення результатів моделювання і регресійного аналізу отриманих даних.

dispatcher – посилання на об’єкт класу Dispatcher, що забезпечує взаємодію у часі монітора та моделей.

Для запуску процесу моделювання використовується кнопка «Старт», з якою пов'язаний метод startMonitoring(), лістинг 7.1. Завдання цього методу налаштувати діаграму, створити монітор і передати йому параметри експерименту та посилання на фабрику моделей, а після цього передати монітор диспетчеру та запустити процес моделювання, викликавши метод диспетчера start().

Лістинг 7.1 - Текст методу для запуску процесу моделювання

```
private void startMonitoring() {
    getJButtonStart().setEnabled(false);
    diagram.setHorizontalMaxText(String.valueOf((int) getFinishTime()));
    diagram.setGridByX(getNIntervals());
    diagram.clear();
    monitor = new TransMonitor();
    monitor.setNameForProtocol("Monitor");
    monitor.setDiagram(getDiagram());
    monitor.setComboBox(getComboBox());

    monitor.setNParallel(getNParallel());
    monitor.setNIntervals(getNIntervals());
    monitor.setInterval(getInterval());
    monitor.setFactory(getFactory());

    getDispatcher().addStartingActor(getMonitor());
    getDispatcher().start();
    //Створення потоку, що розблокує кнопку «Старт»
    new Thread() {
        public void run() {
            try {
                dispatcher.getThread().join();
            } catch (InterruptedException e) {
                e.printStackTrace();
```

```
            }
            getJButtonStart().setEnabled(true);
        };
    }.start();
}
```

7.2.2.1 Клас TransMonitor

Об’єкт цього класу використовується для проведення експериментів по виявленню перехідного процесу в досліджуваній моделі. Об’єкт, за допомогою фабрики, створює необхідну кількість моделей та через метод initForTrans передає моделям час моделювання. Створені моделі передає до стартового списку диспетчера. Через задані проміжки часу, отримує від моделей інформацію про середню довжину черг на інтервалі накопичення і визначає загальне середнє. Отримані дані накопичуються, а потім можуть бути передані на обробку. Крім того, ці дані в процесі роботи компонента можуть візуалізуватися.

Клас TransMonitor наслідує клас Actor, і в ньому визначений метод rule(), який описує правила дії його об'єктів, що забезпечують реалізацію функцій по проведенню експерименту.

Текст методу rule() наводиться в лістингу 7.2.

Лістинг 7.2 - Правила дії об’єкту monitor

```
public void rule() {
    // Масив для моделей
    ITransProcesable[] modelArray = new ITransProcesable[nParallel];
    for (int i = 0; i < nParallel; i++) {
        // Створюємо моделі і готуємо їх до старту
        modelArray[i] = (ITransProcesable) (getFactory()
                .createModel(getDispatcher()));
        modelArray[i].initForTrans(interval * nIntervals);
    }

    // Готуємося до моніторингу моделей
    intervalAverageAsocArray = new HashMap[nIntervals];

    // Готуємо модель для комбобокс
    Map<String, ITransMonitoring> resMap = modelArray[0].getMonitoringObjects();
    String[] keyStrings = (String[]) resMap.keySet().toArray(
            new String[resMap.keySet().size()]);
    comboBox.setModel(new DefaultComboBoxModel<String>(keyStrings));

    // Цикл по інтервалам усереднення
    for (int i = 0; i < nIntervals; i++) {
        // Ініціалізація накопичувачів інформації
        for (int j = 0; j < nParallel; j++)
            for(ITransMonitoring q : modelArray[j].getMonitoringObjects().values()){
                q.resetAccum();
            };
```

```
            // Затримка на довжину інтервалу
            this.holdForTime(interval);

            // Ініціалізація карти середніх значень для інтервалу
            Map<String, Double> averageMap = new HashMap<>();
            for (String s : keyStrings)
                averageMap.put(s, 0.0);

            // Накопичення даних
            for (int j = 0; j < nParallel; j++) {
                for(Entry<String, ITransMonitoring> kv :
                    modelArray[j].getMonitoringObjects().entrySet()){
                    double result = kv.getValue().getAccumAverage();
                    double sum = averageMap.get(kv.getKey());
                    sum += result;
                    averageMap.put(kv.getKey(), sum);
                };
            }

            // Усереднення  даних
            for (String s : keyStrings) {
                double sum = averageMap.get(s);
                sum /= nParallel;
                averageMap.put(s, sum);
            }
            // Запис та індикація результату усереднення
            intervalAverageAsocArray[i] = averageMap;
            if (diagram != null) {
                diagram.getPainter().drawOvalAtXY(
                        (float) (interval * (i + 0.5)),
                        averageMap.get(keyStrings[0]).floatValue(), 3, 3);
                diagram.getPainter().drawOvalAtXY(
                        (float) (interval * (i + 0.5)),
                        averageMap.get(keyStrings[0]).floatValue(), 2, 2);
            }
        }
    }
}
```

У методі rule()за допомогою фабрики створюється масив моделей, розмірність якого дорівнює кількості паралельно працюючих моделей, і кожна модель ініціалізується.

Далі створюється масив колекцій для результатів моніторингу intervalAverageAsocArray.

Після цього монітор починає головний цикл роботи. У цьому циклі монітор ініціалізує накопичувачі інформації в моделях шляхом виклику методу resetAccum(), а потім зупиняється на проміжок часу, який дорівнює довжині інтервалу накопичення інформації.

Після зупинки монітора диспетчер запускає всі моделі, і вони працюють, накопичуючи інформацію протягом інтервалу накопичення.

Після завершення інтервалу накопичення відновляються правила дії

монітора. Він одержує інформацію від кожної з черг, використовуючи метод getAccumAverage(), і визначає середнє значення цих результатів для кожної черги по усім моделям, що працюють паралельно. Отримані середні значення монітор заносить у масив колекцій результатів моніторингу, а також відображає на діаграмі точку, яка відповідає отриманому результату для однієї з черг.

7.3 Пошук лінії регресії для перехідного процесу

Для простої системі масового обслуговування M/M/1/∞ перехідний процес для середньої довжини черги виглядає приблизно так, як показано на рисунку 7.4.

Рисунок 7.3 – Перехідний процес для СМО M/M/1/∞

Для отриманої послідовності точок можна знайти функцію регресії і таким чином отримати аналітичний опис перехідного процесу.

7.3.1 *Функція регресії для перехідного процесу у СМО M/M/1/∞*

Для даних, представлених на рисунку 7.3 і характерних для найпростішої СМО, можна припустити, що залежність, яка нас цікавить, має експоненціальний характер. Така лінія регресії зображена на рисунку 7.2. Відповідна аналітична залежність має вигляд формули 7.1.

$$q(t)=\left(1-e^{-\frac{t}{T}}\right)\cdot q_{cm} \qquad (7.1)$$

де q_{cm} – величина, до якої прямує значення середньої довжини черги з плином часу (стале значення). В залежності від вибору цього параметру рівень, який установився , буде вище, або нижче;

T – параметр, який називають «постійна часу». Чим більше його значення, тим повільніше росте значення функції.

Теоретично, сталий режим для такого процесу наступає лише в нескінченності, але на практиці звичайно вважають, що перехідний процес

закінчується за час рівний 3Т. За цей час значення функції досягає величини рівної 95% від сталого значення

7.3.2 *Цільова функція*

Вибір функції регресії – це лише половина справи. Потрібно ще визначити значення її параметрів. В прикладі, що розглядався вище, таких параметрів два – постійна часу і стале значення. Від вибору значень цих параметрів залежить, як близько до експериментальних точок пройде лінія регресії.

Для прийняття рішення про вибір значень параметрів, потрібно мати критерій оцінки близькості лінії регресії до експериментальних точок. Формальне описання такого критерію називають цільовою функцією. Значення цільової функції залежить від вибору параметрів і повинне збільшуватися, якщо лінія регресії віддаляється від експериментальних точок, і зменшуватися при її наближенні. Найкращими значеннями параметрів будуть такі, при яких цільова функція має мінімальне значення.

У якості цільової функції можна прийняти суму абсолютних значень ординат відхилень експериментальних точок від відповідних значень лінії регресії.

$$\sum_{i=1}^{n}\left|q_i^{регр}-q_i^{експ}\right|\to \min \qquad (7.2)$$

Недолік цього полягає в тому, що при аналітичному визначенні значень параметрів, що забезпечують мінімум цільової функції, доведеться знаходити часткові похідні по цих параметрах, а модуль робить функцію такою, що не можна диференціювати. З цієї причини найчастіше цільова функція представляється у вигляді суми квадратів відхилень

$$\sum_{i=1}^{n}\left(q_i^{регр}-q_i^{експ}\right)^2\to \min \qquad (7.3)$$

7.3.3 *Метод найменших квадратів*

Метод найменших квадратів припускає представлення цільової функції у вигляді суми квадратів відхилень. Для того, щоб знайти оптимальні значення параметрів, потрібно взяти похідні від цільової функції по цих параметрах і прирівняти їх до 0. Після цього залишається розв'язати отриману систему рівнянь щодо невідомих значень параметрів. В розглянутому вище прикладі цільова функція виглядатиме так

$$\sum_{i=1}^{n}\left(q_{cm}\left(1-e^{-\frac{t_i}{T}}\right)-q_i\right)^2\to \min \qquad (7.4)$$

Взявши похідні від цільової функції по $q_{уст}$ і Т, та прирівнявши їх до

нуля 0, отримаємо систему рівнянь для визначення цих параметрів.

$$\begin{cases} q_{cm}\sum_{i=1}^{n}\left(1-e^{-\frac{t_i}{T}}\right)^2-\sum_{i=1}^{n}q_i\left(1-e^{-\frac{t_i}{T}}\right)=0 \\ q_{cm}\sum_{i=1}^{n}t_i e^{-\frac{t_i}{T}}\left(1-e^{-\frac{t_i}{T}}\right)-\sum_{i=1}^{n}t_i q_i e^{-\frac{t_i}{T}}=0 \end{cases} \quad (7.5)$$

На жаль, вирішити аналітично цю систему рівнянь не вдається. Але знайти оптимальні значення q_{cm} і T можна за допомогою чисельних методів пошуку екстремуму. В цьому випадку цільова функція не обов'язково повинна бути такою, що диференціюється і можна мінімізувати як суму квадратів відхилень, так і суму модулів відхилень.

7.3.4 *Пошук екстремуму для функції однієї змінної за допомогою чисел Фібоначчі*

Планування експериментів для пошуку оптимального значення цільової функції, яка залежить від однієї змінної, полягає в наступному.

Перш за все, задається точність, з якою потрібно визначити невідомий параметр. Цю величину називатимемо кроком пошуку.

Потім визначаються границі області пошуку, всередині якої знаходиться екстремум, причому екстремум повинен бути тільки один. Такі функції називають унімодальними.

7.3.4.1 Визначення границь області пошуку

Для визначення границь області пошуку слід максимально використовувати всю наявну апріорну інформацію про досліджувану систему. В тому випадку, якщо границі області пошуку відразу визначити не вдається, зазвичай використовують наступний алгоритм (передбачається, що ми шукаємо мінімум цільової функції).

Вибирається будь-яке значення параметра і для нього обчислюється значення цільової функції.

Значення параметра збільшується на величину кроку, обчислюється значення цільової функції в новій точці, і порівнюється з її значенням в початковій точці.

Якщо нове значення менше, значить напрямок пошуку вибрано правильно і перший етап завершено.

Якщо нове значення більше попереднього, то знак кроку зміни змінної потрібно змінити на протилежний і повторити попередні дії.

Якщо ж і в цьому випадку значення цільової функції більше, ніж в початковій точці, то це означає, що мінімум знаходиться в початковій точці і його шукати вже не потрібно.

В результаті виконання першого етапу, стає відомим напрямок пошуку,

який задається знаком кроку зміни параметра.

Далі починається пошук другої границі області пошуку.

В розглянутому нижче алгоритмі пошуку, відстань від початкової точки до точки, де проводиться експеримент, поступово збільшується з кожним новим кроком, а ліву границю переносять у точку, де був попередній експеримент. Для формування відстаней до точок, де буде проводитися новий експеримент, зручно використовувати числа Фібоначчі.

Особливість чисел Фібоначчі полягає в тому, що кожне наступне, починаючи з 3-го, дорівнює сумі двох попередніх:

1 1 2 3 5 8 13 21 34 55 89 . . .

Таким чином, визначивши напрямок пошуку, ми маємо результати експериментів у точках 0 та 1. Відстань між ними дорівнює 1. наступне число Фібоніччі 2, тому наступний експеримент ми проводимо на відстані 2 від крайньої точки, тобто у точці 3.

Якщо значення цільової функції у цій точці більше за попереднє, фіксуємо границі пошуку – це точки 0 та 3 і переходимо до пошуку всередині інтервалу.

Якщо ж значення функції у цій точці менше за попереднє, продовжуємо пошук правої границі. Визначаємо відстань між точками, де були проведені останні експерименти (точки 1 та 3). Вона дорівнює 2. Наступне число фібоначчі 3. Початок координат переносимо у точку 1. Тоді точка 3 приймає значення 2 (2=3-1), а новий експеримент проводять в точці 2+3=5.

Якщо функція продовжує зменшуватися, то точка 2 перетворюється на 0, точка 5 на 3 і новий експеримент проводять в точці 3+5=8.

Наступний експеримент буде проводитись у точці, віддаленій на 13 кроків від початкової, причому, слід враховувати, що абсолютна координата початкової точки постійно зміщується, тобто в процесі пошуку правої границі ми постійно змінюємо і ліву границю.

Так продовжується до тих пір, поки значення цільової функції не почне зростати, рисунок 7.4.

Зверніть увагу, послідовність розмірів інтервалів між точками, де проводилися експерименти, відповідає послідовності чисел Фібоначчі

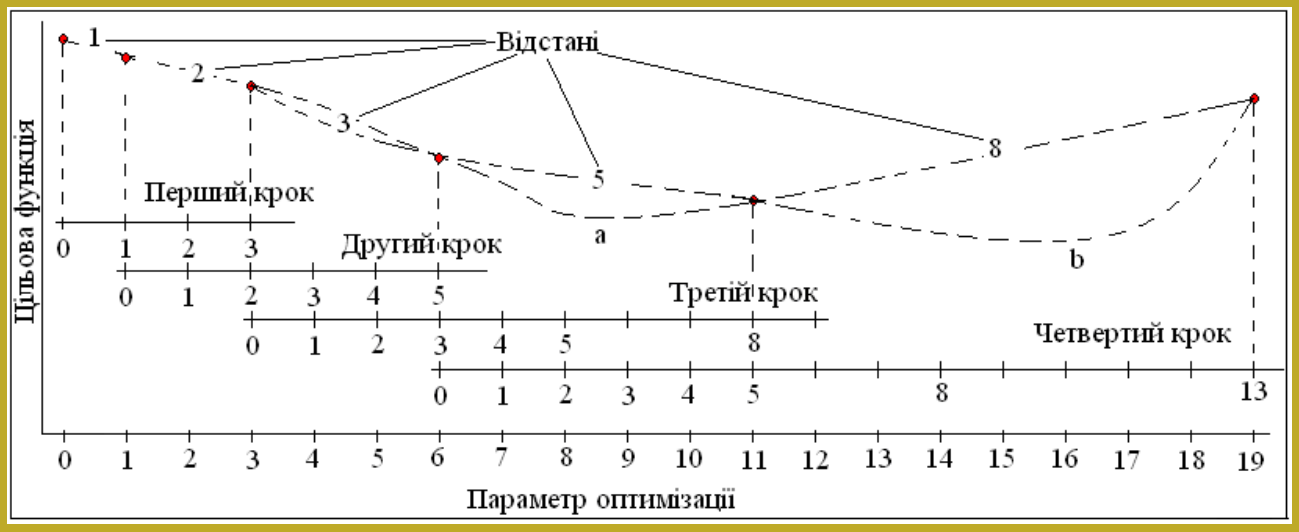

Рисунок 7.4 – Результати визначення меж області пошуку

На рисунку 7.4 зображена ситуація, коли в точці 13 цільова функція почала зростати. В результаті, на краях інтервалу значення функції стали більше, ніж посередині. Проте мінімум може бути розташований як у лівій частині інтервалу – варіант кривої “a” на рисунку 7.3, так і в правій – варіант кривої “b” на тому ж рисунку.

7.3.4.2 Пошук екстремуму у середині області

Після визначення границь області пошуку, можна починати пошук точки екстремуму всередині цієї області. Пошук полягає в поступовому звуженні знайденої області пошуку. Тут також дуже зручно використовувати числа Фібоначчі. Алгоритм фактично є продовженням попереднього і полягає в наступному.

Усередині області пошуку проводять додаткові експерименти так, щоб значення цільової функції були відомі в обох точках, які відповідають числам Фібоначчі, з яких складається число, що визначає праву границю пошуку. Для рисунка 7.3 це точки 5 і 8. Як правило, для однієї з цих точок результат вже визначений, залишається протестувати іншу точку. В результаті, область пошуку розбивається на 3 інтервали, ширина кожного з яких пропорційна деякому числу Фібоначчі.

Після цього порівнюють значення функції відгуку для двох точок всередині області пошуку, і приймають рішення про звуження цієї області, приймаючи в якості нової границі області пошуку точку з більшим значенням функції відгуку. Якщо значення цільової функції для точок всередині області пошуку однакові, то відкидається і лівий і правий інтервали. Ширина нової області пошуку у будь-якому випадку знову виходить рівною числу Фібоначчі.

У випадку, якщо змінюється ліва границя області пошуку, слід відкоригувати значення чисел Фібоначчі для границь області і точок всередині неї. Всі ці числа повинні бути зменшені на старе значення лівої границі. В результаті, лівій границі знову відповідатиме число 0.

Якщо ширина хоча б одного з інтервалів більше одиниці, процес повторюється.

В іншому випадку обирається точка з мінімальним значенням цільової функції, яка і буде точкою оптимуму.

Розглянемо приклад пошуку у середині області, як продовження попереднього. Нехай функція відгуку має вигляд показаний на рисунку 7.5, варіант b.

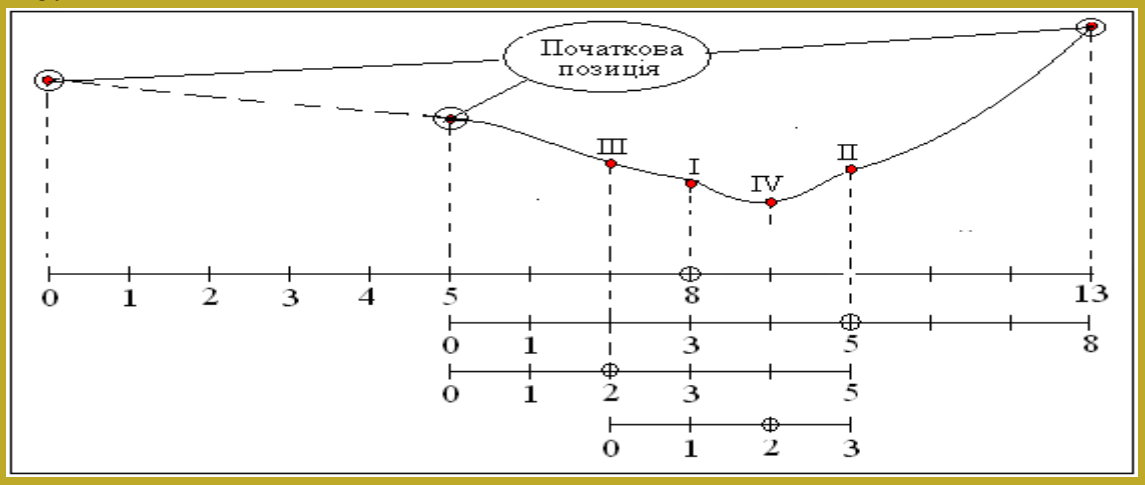

Рисунок 7.5 – Пошук в межах області

Після визначення меж області пошуку ми маємо результати експериментів у точках 0, 5 та 13. Ширина області дорівнює числу Фібоначчі 13, що складається з 5 та 8, тому наступні експерименти слід проводити в точках 5 та 8. Але в точці 5 експеримент вже проведено. Залишається провести експеримент в точці 8, результат якого має позначку I.

Цей результат менший ніж у точці 5, тому ліву границю пошуку переносимо у точку 5 и змінюємо систему координат.

Ширина нової області дорівнює числу 8, що складається з 3 та 5. У точці 3 (до зміни координат це була точка 8) експеримент вже проведено. Залишається провести експеримент в точці 5, результат якого має позначку II.

Цей результат більший, ніж у точці 3, тому змінюємо праву границю і переносимо її до точки 5. Систему координат при цьому змінювати не потрібно.

Ширина звуженої області дорівнює числу 5, що складається з 2 та 3. У точці 3 експеримент вже проведено. Залишається провести експеримент в точці 2, результат якого має позначку III.

Цей результат більший, ніж у точці 3, тому змінюємо ліву границю і переносимо її до точки 3 і знову змінюємо систему координат.

Ширина нової області дорівнює числу 3 і в усіх точках області, окрім точки 2, експерименти вже проведено. Залишається провести останній експеримент і визначитися з мінімумом. На рисунку результат експерименту має позначку IV і тут відгук має мінімальне значення, але могло статися й так, що відгук у цій точці мав значення більше ніж у точці 2, тоді б саме це значення параметру було б оптимальним.

Як бачимо, основна перевага методу Фібоначчі полягає в тому, що на кожному кроці використовуються дані попереднього кроку, що скорочує число експериментів.

7.3.5 *Метод золотого перетину*

Цей метод відрізняється від попереднього лише тим, що розмір області пошуку не пов'язаний з числом Фібоначчі, але співвідношення між інтервалами, на які ділиться область пошуку, повинне зберігатися постійним. На рисунку 7.5 представлено розділення області пошуку при застосуванні методу золотого перетину.

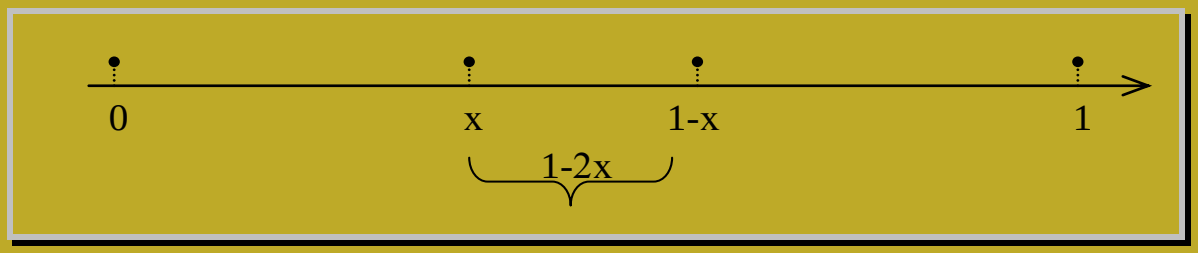

Рисунок 7.5 – Розділення області пошуку при застосуванні методу золотого перетину

Ширина всієї області на рисунку представлена одиницею. Точки всередині області, в яких потрібно провести експерименти, віддалені від границь області на відстань x. Після оцінки результатів експериментів в цих точках, один із крайніх інтервалів області буде відкинуто і точки, в яких потрібно буде проводити експерименти на наступному кроці пошуку, будуть вже віддалені від границі інтервалу на відстань 1-2x. Але відношення довжини цього інтервалу до ширини нової області пошуку повинно залишатися незмінним. Тобто

$$\frac{1-2x}{1-x} = \frac{x}{1} \qquad (7.6)$$

Із цього відношення отримуємо квадратне рівняння для пошуку x.

$$x^2 - 3x + 1 = 0 \qquad (7.7)$$

Розв'язавши це рівняння, отримуємо

$$x \approx 0.382,\ 1\text{-}x \approx 0.618,\ 1\text{-}2x \approx 0{,}236 \qquad (7.8)$$

7.3.6 *Метод дихотомії*

Цей метод один із найвідоміших, але не найкращий, оскільки він вимагає проведення більшої кількості експериментів в порівнянні із попередніми методами. Суть методу полягає в наступному, рисунок 7.5.

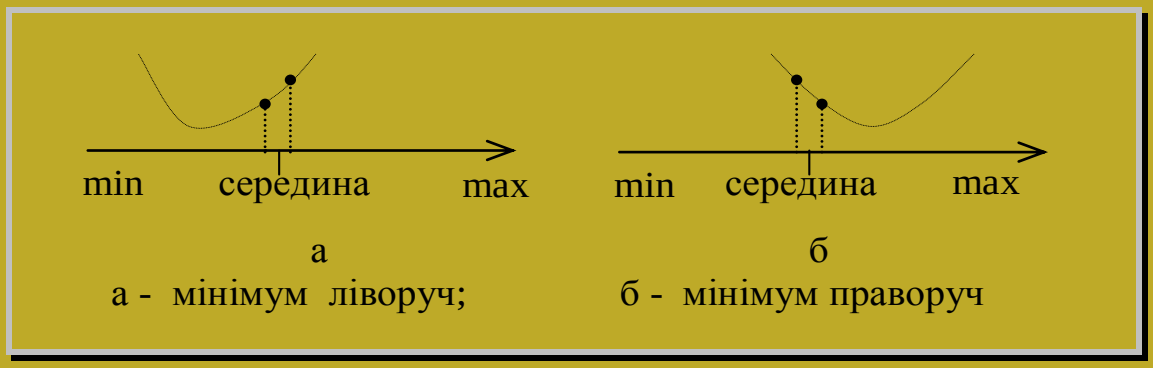

Рисунок 7.5 – Пошук екстремуму методом дихотомії

Область пошуку ділиться навпіл і з двох сторін від середньої точки, на невеликій відстані, яка відповідає необхідній точності визначення значення параметра, проводиться два експерименти. На підставі результатів цих дослідів робиться висновок про те, де знаходиться екстремум - в лівій або правій половині області пошуку, що дозволяє звузити область пошуку в 2 рази.

В середині нової області пошуку знову проводять два експерименти і так доти поки ширина області пошуку не стане менше заданої точності.

7.3.7 *Пошук екстремуму функції кількох змінних*

Існує декілька способів вирішення цієї задачі. Найбільш простим із них вважається метод координатного спуску. Суть його полягає в наступному.

Обирається початкова точка для пошуку і, починаючи від цієї точки, знаходиться мінімум цільової функції, шляхом зміни лише однієї змінної. Вся решта змінних не міняється.

Після визначення мінімуму по вибраній координаті починається пошук мінімуму по наступній координаті і так далі. Після того, як значення цільової функції перестануть змінюватися при переході від однієї змінної до іншої, пошук закінчується.

Саме цей метод використовується в лабораторній роботі. Пошук проводиться поперемінно, спочатку по сталому рівню перехідного процесу, потім по його тривалості.

7.3.8 *Компонент ParmFinderView*

Цей компонент забезпечує пошук параметрів рівняння регресії для перехідного процесу в досліджуваній моделі. Візуальна композиція класу наведена на рисунку 7.6.

Візуальна частина використовується для налаштування й виводу результатів пошуку у вигляді чисел.

Пошук параметрів рівняння регресії може виконуватися як в автоматичному, так і в ручному режимі.

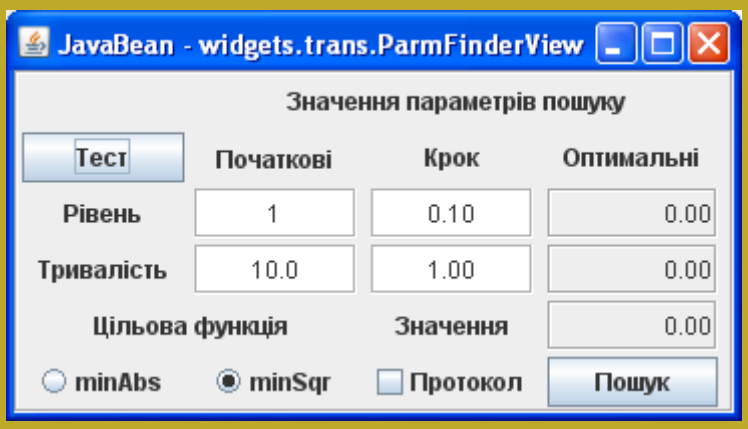

Рисунок 7.6 - Візуальна композиція класу ParmFinderView

При ручному пошуку користувач оцінює по експериментальних точках значення сталого рівня й час перехідного процесу, вводить ці значення в колонку «Початкові» і натискає кнопку «Тест». Уведені значення передаються компоненту ParamFinder, який відображає на діаграмі лінію регресії, що відповідає введеним параметрам, і розраховує значення цільової функції, що є показником відхилення експериментальних точок від лінії регресії.

В якості цільової функції можна вибирати або суму абсолютних відхилень, або суму квадратів відхилень експериментальних точок від лінії регресії. Для перемикання використовується група радіо кнопок «minAbs» - «minSqr».

Якщо тестувати різні варіанти параметрів лінії регресії можна підібрати більше менш підходящі значення.

При автоматичному пошуку задаються стартові значення параметрів лінії регресії й крок їхньої зміни при пошуку. При натисканні кнопки «Пошук» ці значення передаються компоненту ParmFinder, який методом координатного спуску робить пошук оптимальних значень параметрів, після чого лінія регресії й значення цільової функції виводяться для знайдених значень параметрів.

За бажанням можна отримати протокол пошуку.

7.3.8.1 Клас TransParmFinder

Цей клас забезпечує реалізацію алгоритму координатного спуску при пошуку оптимальних значень параметрів лінії регресії. Для пошук екстремуму по кожній з координат використовується метод чисел Фібоначчі.

Об'єкти розглянутого класу знаходять параметри лінії регресії, рівняння якої визначається співвідношенням 7.9

$$Y(t) = \left(1 - e^{-\frac{3t}{tau}}\right) \cdot level \qquad (7.9)$$

Ця формула ідентична формулі 7.1. Різниця полягає лише в тому, що замість постійної часу T, тут використовується тривалість перехідного процесу tau=3T.

Розташування лінії регресії щодо експериментальних точок залежить від значень параметрів залежності 7.9 - тривалості перехідного процесу tau, і сталого значення черги, що представлено у формулі змінної level. Оптимальні значення саме цих параметрів і знаходять об’єкти класу TransParmFinder.

7.4 Опис програмного комплексу для лабораторної роботи

Програмний комплекс для даної лабораторної роботи створювався у вигляді пакету testTrans виходячи з того, що він повинен забезпечувати:

- налаштування параметрів моделі;
- накопичення, обробку і видачу в графічному вигляді середніх значень черги по інтервалам накопичення;
- можливість пошуку параметрів рівняння регресії;

Класи, які відповідають за роботу моделі, що досліджується, знаходяться в пакеті testTrans проекту SimulationAllLab.

Клас TransGUI реалізує шар подання програми. До складу шару подання, окрім компонентів, що забезпечують налаштування параметрів моделі, входять компонент TransProcessManager та компонент ParmFinderView.

Клас Model реалізує інтерфейс IExperimentable є шаром моделі застосування.

Шар компонентів застосування представлений такими об’єктами:

- transactGenerator класу TransactGenerator, генератор трансакцій;
- queue классу QueueForTransactions, що моделює чергу транзакцій;
- device класу Device, обслуговуючий прилад.

Для об’єктів transactGenerator та device у пакеті створено класи, що реалізують поведінку генератора трансакцій та обслуговуючого пристрою.

Генератор трансакцій моделі створює заявки незалежно від максимального розміру черги. Заявки, що не приймаються чергою, втрачаються.

7.4.1 *Клас TransGUI*

Клас TransGUI реалізує шар подання програми. До складу шару подання, окрім компонентів, що забезпечують налаштування параметрів моделі, входять компонент TransProcessMonitor та компонент ParmFinderView.

Компонент TransProcessManager забезпечує настройку параметрів та проведення експерименту по визначенню перехідного процесу.

Компонент ParmFinderView забезпечує пошук параметрів рівняння регресії для перехідного процесу в системі. Початкові дані об'єкт одержує від компоненту TransProcessManager, а результати обробки виводить на його діаграму і в протокол.

Візуальна композиція класу представлена на рисунку 7.7. Компонента ParmFinderView на рисунку не видно. Він з’являється за викликом.

Особливість програмної реалізації класу полягає у тому, що тут створюється фабрика моделей і посилання на неї передається компоненту TransProcessManager.

У разі виклику компонента ParmFinderView, він створюється і йому передається посилання на компонент TransProcessManager через метод setIRegresable() та на діаграму, через метод setDiagram().

Для візуалізації компонента ParmFinderView у класі створюється об'єкт типу JFrame.

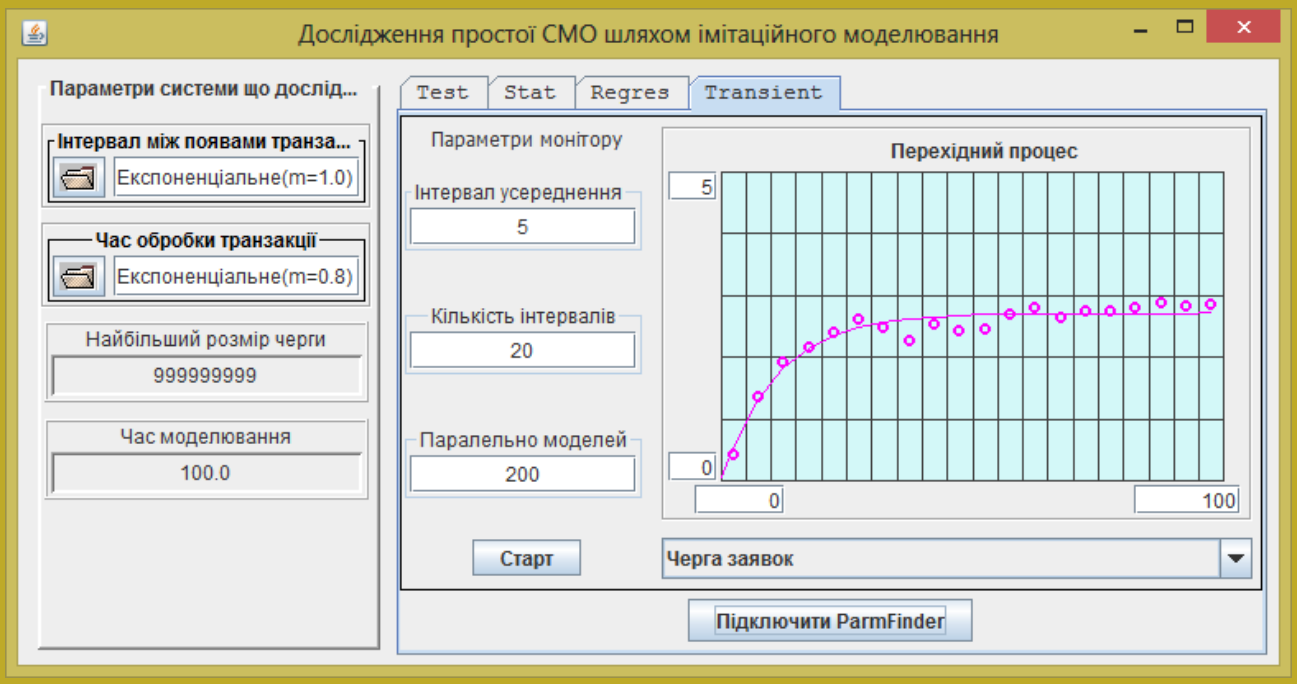

Рисунок 7.7 - Візуальна композиція проекту для виконання лабораторної роботи

7.4.2 *Реалізація шару моделі*

7.4.2.1 Клас Model

Клас Model у проекті відрізняється від аналогічного класу, що використовувався у попередній лабораторній роботі тим, що у ньому реалізовано інтерфейс ITransProcessable. В лістингу 7.3 наведено реалізації методів інтерфейсу ITransProcessable в даному класі.

Лістинг 7.7 – Реалізація інтерфейсу ITransProcessable у класі Model

```
//Реалізація інтерфейсу ItransProcesable
@Override
public void initForTrans(double finishTime) {
    gui.getChooseDataFinishTime().setDouble(finishTime);
    getDevice().setFinishTime(finishTime);
    getTransactGenerator().setFinishTime(finishTime);
}

@Override
public Map<String, ITransMonitoring> getMonitoringObjects() {
    Map<String, ITransMonitoring> map = new HashMap<>();
    map.put("Черга заявок", (ITransMonitoring)getQueue());
    return map;
}
```

7.4.3 *Реалізація шару компонентів моделі*

7.4.3.1 Клас TransactionGenerator

Цей клас відрізняється від такого ж класу, що розглядався у попередній лабораторній роботі, тим, що до нього додано метод setFinishTime(), що викликається з методу initForTrans().

7.4.3.2 Клас Device

Цей клас відрізняється від такого ж класу, що розглядався у попередній роботі, тільки тим, що до нього додано метод setFinishTime(), що викликається з методу initForTrans().

7.5 Порядок виконання роботи

7.5.1 *Підготовка до роботи*

– Активізуйте Java застосування, що знаходиться у архіві SimulationAllLab.jar, або завантажте проект SimulationAllLab.zip до Eclipse.

7.5.2 *Перевірка працездатності програмного комплексу*

– Активізувавши застосування «Лабораторна робота 7» переконайтеся в працездатності програмного комплексу.

7.5.3 *Дослідження впливу настроювань моделі на результати моделювання*

– Дослідить вплив кількості паралельно працюючих систем на характер залежності середнього розміру черги від часу. Для цього проведіть два експерименти при малій і великій кількості паралельно працюючих систем і порівняйте їх. Поняття «велике» і «мале» залежить від можливостей комп'ютера. Для кожного настроювання робіть кілька дослідів, і у звіті фіксуйте найбільш характерний результат. Проаналізувавши результати, зробіть висновок.

– Дослідить вплив довжини інтервалу накопичення на характер залежності середнього розміру черги від часу при середній кількості паралельно працюючих системах. Для цього порівняйте графіки цієї залежності для інтервалів довжиною 20 і 5, а кількості інтервалів накопичення відповідно 10 і 40, щоб час моделювання в цих дослідах залишався рівним 200. Для кожного налаштування робіть кілька дослідів, і у звіті фіксуйте найбільш характерний результат. Проаналізувавши результати, зробіть висновок.

7.5.4 *Автоматичний пошук оптимальної лінії регресії*

– занесіть стартові значення, далекі від оптимальних, у відповідні поля на панелі «Значення параметрів пошуку»;

– увімкніть перемикач «Протокол»;

– натисніть на кнопку «Пошук». В результаті буде виведений графік рівняння регресії, а у стовпчику «Оптимальні» будуть показані оптимальні значення параметрів і значення цільової функції;

– Досліджуйте працездатність алгоритму при різних стартових значеннях параметрів і величинах кроку. Фіксуйте стартові значення і результати пошуку;

– Досліджуйте працездатність алгоритму при різних кількостях експериментальних точок і інтервалів накопичення. Фіксуйте кількість точок і інтервалів і результат пошуку.

7.5.5 *Ручний пошук оптимальної лінії регресії*

– установіть значення довжини інтервалу накопичення й кількість паралельно працюючих систем такими, щоб отримувати найбільш прийнятні по швидкодії і якості результати;

– кількість інтервалів накопичення виберіть такою, щоб за час моделювання перехідний процес закінчувався;

– натисніть кнопку «Старт» і отримайте послідовність експериментальних точок для перехідного процесу.

– занесіть як стартові значення у відповідні поля на панелі «Значення параметрів пошуку» числа, які далекі від оптимальних значень.

– натисніть на кнопку «Тест». У результаті буде виведений графік рівняння регресії, що відповідає введеним значенням, а в полі «Значення» показане значення цільової функції;

– використовуючи метод координатного спуска, реалізуйте вручну пошук коефіцієнтів рівняння регресії, виконавши по одному циклу пошуку для кожної з двох координат.

7.5.6 *Дослідження залежності тривалості перехідного процесу від коефіцієнта завантаження системи*

– використовуючи модель, отримайте залежність тривалості перехідного процесу, та сталого значення середньої довжини черги від коефіцієнта завантаження системи. Проведіть на 3 рівнях по 3 експерименти на кожному рівні. Отримані результати покажіть у вигляді таблиці й графіка. Сталі значення середньої довжини черги порівняйте з теоретичними.

7.6 Завдання для самостійної роботи

Реалізуйте в моделі для свого варіанту РГР інтерфейс ITransprocesable, а до візуальної частини додайте закладку з компонентом TransProcessManager і виконайте необхідні налаштування.

Перевірте роботу застосування у режимі дослідження перехідних процесів.

7.7 Зміст звіту

– Найменування й ціль роботи.

– Результати дослідження впливу налаштування моделі на результати моделювання у вигляді графіків перехідних процесів для різних параметрів моделі й висновки про вплив налаштувань на вид досліджуваної залежності.

– Результати експериментів по оцінці працездатності алгоритму автоматичного пошуку.

– Протокол автоматичного пошуку.

– Результати дослідження залежності тривалості перехідного процесу та сталого значення середньої довжини черги від коефіцієнта завантаження системи.

– Результати виконання завдання для самостійної роботи.

– Висновки по роботі.

Контрольні питання

1. Методика одержання експериментальних даних про перехідний процес.
2. Що таке лінія регресії і як її знайти.
3. Що таке цільова функція, як її вибрати.
4. Методика пошуку екстремуму методом дихотомії.
5. Методика пошуку екстремуму методом золотого перетину.
6. Методика пошуку екстремуму з використанням чисел Фібоначчі.
7. Метод координатного спуску.
8. Опис структури проекту по діаграмі класів.
9. Опис моделі, яка використовується в проекті.
10. Опис класів, які описують властивості й поводження основних компонентів моделі.
11. Як організований збір інформації про довжину черги.
12. Як організоване усереднення інформації про довжину черги.
13. Як працює монітор.
14. Як реалізований у проекті автоматичний пошук параметрів лінії регресії, пояснити, використовуючи протокол пошуку.
15. Як працює застосування, починаючи від натискання кнопки “Старт”.

Рекомендована література

1. Томашевський В.И.Моделювання систем. – К.: Видавнича група BHV, 2005. –352с.:іл.

2. Советов Б.Я., Яковлев С.А. Моделирование систем: Учебник для вузов. – М.: Высш.шк., 1998. –320с.