Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

Паралельні та розподілені обчислення

Методичні рекомендації для виконання лабораторних робіт студентів денної форми навчання галузі 12 Інформаційні технології

ЗАТВЕРДЖЕНО
на засіданні кафедри кібербезпеки
та програмного забезпечення
(протокол №2 від 29.08.24)

Кропивницький
2024

Паралельні та розподілені обчислення: Методичні рекомендації до виконання лабораторних робіт для студентів денної форми навчання галузі 12 Інформаційні технології. / уклад. Минайленко Р.М. / М-во освіти і науки України, Центральноукр. нац. техн. ун-т; – Кропивницький: ЦНТУ– 2024. – 78 с.

Укладач: Минайленко Р.М

Рецензенти: Коваленко О.В., докт. техн. наук, доцент;

Улічев О.С., канд. техн. наук.

Зміст

Вступ

Курс «Паралельні та розподілені обчислення» призначений для набуття теоретичних і практичних знань в області паралельних та розподілених обчислень, оволодіння концепціями сучасного програмування в рамках парадигм паралельного та розподіленого програмування. Основу вивчення курсу складають підходи до програмування в багатопотокових системах, розподілених системах, системах синхроних паралельних розрахунків, розглядаються проблеми сумісної роботи процесів паралельної програми та їх синхронізації. Реалізація синхронних або асинхронних паралельних процесів з використанням бібліотек MPI / OpenMP, стандартними засобами мов програмування C++, Java для розподілених обчислень.

Метою викладання дисципліни « Паралельні та розподілені обчислення » є вивчення теоретичних основ та практичних аспектів використання паралельних обчислювальних систем для вирішення складних прикладних задач, оволодіння концепціями сучасного програмування в рамках парадигм паралельного та розподіленого програмування.

Основними **завданнями** вивчення дисципліни є:

– здатність використовувати сучасні методи і мови програмування для оволодіння концепціями паралельних та розподілених обчислень.

– здатність створювати програмне забезпечення для паралельних та розподілених обчислень.

Для денної форми навчання: Викладання курсу передбачає для засвоєння дисципліни традиційні лекційні заняття із застосуванням мультимедійних презентацій, у поєднанні з лабораторними заняттями.

Формат очний (Face to face)

Для заочної форми навчання: Під час сесії формат очний (Face to face), у міжсесійний період – дистанційний (online).

У результаті вивчення навчальної дисципліни «Паралельні та розподілені обчислення» студент буде:

– ПР17. Виконувати паралельні та розподілені обчислення, застосовувати чисельні методи та алгоритми для паралельних структур, мови паралельного програмування при розробці та експлуатації паралельного та розподіленого програмного забезпечення.

– ЗК2. Застосовувати знання у практичних ситуаціях для розв’язування технічних задач спеціальності, а саме керувати процесами, реалізовувати взаємодію процесів, моделювати паралельні обчислення, створювати та налагоджувати паралельні та розподілені програми, здійснювати побудову паралельного алгоритму і виконувати його аналіз, створювати програми із застосуванням процесів (потоків). Розв’язувати задачі аналізу та синтезу засобів паралельних та розподілених обчислень, використовуючи моделі організації паралельних обчислень, основні мови паралельного програмування, бібліотечні функції для паралельного програмування, засоби оптимізації обміну даними між

паралельно-виконуваними програмами, засоби опису, взаємодії, синхронізації та взаємного виключення паралельних процесів. Реалізовувати синхронні або асинхронні паралельні процеси з використанням бібліотек MPI / OpenMP, стандартними засобами мов програмування C++, Java для розподілених обчислень.

– СК16. Реалізовувати високопродуктивні обчислення на основі хмарних сервісів і технологій, паралельних і розподілених обчислень при розробці й експлуатації розподілених систем паралельної обробки інформації.

Політика дисципліни

Академічна доброчесність:

Очікується, що студенти будуть дотримуватися принципів академічної доброчесності, усвідомлювати наслідки її порушення. Детальніше за посиланням URL http://www.kntu.kr.ua/doc/dobro.pdf

Відвідування занять

Відвідування занять є важливою складовою навчання. Очікується, що всі студенти відвідають лекції і практичні заняття курсу.

Пропущені заняття повинні бути відпрацьовані не пізніше, ніж за тиждень до залікової сесії.

Поведінка на заняттях

Недопустимість: запізнень на заняття, списування та плагіат, несвоєчасне виконання поставленого завдання.

При організації освітнього процесу в Центральноукраїнському національному технічному університеті студенти, викладачі та адміністрація діють відповідно до: Положення про організацію освітнього процесу; Положення про організацію вивчення навчальних дисциплін вільного вибору; Положення про рубіжний контроль успішності і сесійну атестацію студентів ЦНТУ; Кодексу академічної доброчесності ЦНТУ.

Критерії оцінювання. Знання здобувачів вищої освіти оцінюється при проведенні екзаменаційного контролю як з теоретичної, так і з практичної підготовки за такими критеріями:

– "відмінно" – здобувач вищої освіти досконало засвоїв теоретичний матеріал, глибоко і всебічно знає зміст навчальної дисципліни, основні положення наукових першоджерел та рекомендованої літератури, логічно мислить і будує відповіді, вільно використовує набуті теоретичні знання при аналізі практичного матеріалу, висловлює своє ставлення до тих чи інших проблем, демонструє високий рівень засвоєння практичних навичок;

– "добре" – здобувач вищої освіти добре засвоїв теоретичний матеріал, аргументовано викладає його, володіє основними аспектами з першоджерел та рекомендованої літератури, має практичні навички, висловлює свої міркування з приводу тих чи інших проблем, але припускається певних неточностей і похибок у логіці викладу теоретичного змісту або при аналізі практичного матеріалу;

– "задовільно" – здобувач вищої освіти, в основному, володіє теоретичними знаннями з навчальної дисципліни, орієнтується в першоджерелах та рекомендованій літературі, але непереконливо відповідає, додаткові питання викликають невпевненість або відсутність стабільних знань; відповідаючи на запитання практичного характеру, виявляє неточності у знаннях, не вміє оцінювати факти та явища, пов'язувати їх із майбутньою діяльністю;

– "незадовільно" – здобувач вищої освіти не опанував навчальний матеріал дисципліни, не знає наукових фактів, визначень, майже не орієнтується в першоджерелах та рекомендованій літературі, відсутні наукове мислення, практичні навички не сформовані.

Шкала оцінювання: національна та ЄКТС

Оцінка за шкалою ECTS	Визначення	Оцінка		
		За національною системою (екзамен, диф. залік, курс. проект, курс. робота, практика)	За національною системою (залік)	За системою ЦНТУ
A	ВІДМІННО - відмінне виконання лише з незначною кількістю помилок	5 (відмінно)	Зараховано	90 – 100
B	ДУЖЕ ДОБРЕ - вище середнього рівня з кількома помилками	4 (добре)	Зараховано	82-89
C	ДОБРЕ - в загальному правильна робота з певною кількістю грубих помило			74-81
D	ЗАДОВІЛЬНО - непогано, але зі значною кількістю недоліків	3 (задовільно)	Зараховано	64-73
E	ОСТАТНЬО - виконання задовольняє мінімальні критерії			60-63
FX	НЕЗАДОВІЛЬНО - потрібно попрацювати перед тим, як перескласти	2 (незадовільно)	Не зараховано	35-59
F	НЕЗАДОВІЛЬНО - необхідна серйозна подальша робота			1-34

ЛАБОРАТОРНА РОБОТА №1

Тема: СЕРЕДОВИЩА РОЗРОБКИ ГРАФІЧНИХ ДОДАТКІВ NETBEANS IDE

Мета роботи: Одержання навичок роботи з NetBeans IDE на платформі JavaFx, створення проекту, введення та компіляція програми, створення файлу *.jar, що виконується, запуск програми.

Завдання:

1. Створити простий проект JavaFx в NetBeans IDE.
2. Ввести текст демонстраційної програми. Відкомпілювати програму, усунути ймовірні помилки. Створити файл *.jar, що виконується, запустити програму на виконання.

ТЕОРЕТИЧНІ ВІДОМОСТІ:

Перший приклад простої програми

Розглянемо просту програму, написану на Java. Почнемо з компіляції і запуску прикладу програми.

```
//=== Лабораторна робота № 1

public class Lab1 extends Application {
//=== Метод   launch() запускає програму викликом методу start()
@Override
public void start(Stage primaryStage) {
Button btn = new Button();
btn.setText("Say 'Hello World'");
btn.setOnAction((ActionEvent event) -> {
System.out.println("Hello World!");
});
StackPane root = new StackPane();
root.getChildren().add(btn);
Scene scene = new Scene(root, 300, 250);
primaryStage.setTitle("Hello World!");
primaryStage.setScene(scene);
primaryStage.show();
//=== вхід в програму
public static void main(String[] args) {
//Метод          launch() запускає програму викликаючи метод start()

launch(args);//
}
}
```

Введення коду програми

Синтаксис мови програмування Java аналогічний синтаксису мови програмування C, C++ та C#.

Java вихідний файл називається модулем компіляції. Він містить опис одного або декількох класів. Компілятор Java вимагає, щоб вихідний файл мав розширення *.java.

Java увесь код повинен розміщатися всередині класу.

Ім'я головного класу, який містить функцію main() повинне збігатися з іменем файлу, що містить програму. Необхідно також, щоб написання імені файлу відповідало імені класу, включаючи малі і прописні букви. Це обумовлене тим, що код Java чутливий до регістру символів.

Компіляція і виконання програми в NetBeans IDE

Компіляція програми здійснюється натисканням кнопки F9.
Компіляція і запуск програми здійснюється натисканням кнопки F6.
Файл, що виконується, має розширення (*.jar).

Більш докладний розгляд першого прикладу програми

Програма Lab1 має кілька важливих особливостей, характерних для всіх програм Java.

Розглянемо кожну частину цієї програми більш докладно. Програма починається з наступного рядка:

```
// Лабораторна робота № 1
В Java підтримується три стилі коментарів.
/*Comment*/;
//Comment;
/** Comment*/.
```

Перші два являють собою звичайні коментарі, які застосовуються як в Java, так і в C++. Останній введений для автоматичного документування тексту програми. Після написання вихідного тексту утиліта автоматичної генерації документації збирає тексти таких коментарів в один файл.

Наступний рядок програми має такий вигляд:

```
public class Lab1 extends Application {
```

у цьому рядку ключове слово class використовується для оголошення про те, що виконується визначення нового класу. Lab1 – це ідентифікатор, що є іменем класу.

Усе визначення класу, у тому числі його членів, буде розміщатися між відкриваючої { і закриваючої } фігурними дужками.

Наступний рядок коду виглядає так:

```
public static void main(String[] args) {
```

Виконання всіх додатків Java починається з виклику методу main().

Ключове слово public – модифікатор доступу, який дозволяє програмісту управляти видимістю елементів класу. Коли елементу класу передує ключове слово public, він є доступним іншим об'єктам програми.

У цьому випадку метод main() повинен бути визначений як public, оскільки при запуску програми він повинен викликатися кодом, визначеним поза його класом.

Ключове слово static дозволяє викликати метод main() без явного створення екземпляра класу.

Ключове слово void повідомляє компілятору, що метод main() не повертає ніяких значень. Для передачі інформації, необхідної методу, служать змінні, які називають параметрами. Якщо для даного методу ніякі параметри не потрібні, слід указувати порожні дужки.

Метод main() містить тільки один параметр, але досить складний.

Частина String[] args оголошує параметр args, який являє собою масив екземплярів класу String.

У цьому випадку параметр args приймає будь-які аргументи командного рядка.

Увесь код, що утворює метод, розташований між відкриваючою і закриваючою фігурними дужками методу.

Ще один важливий момент: метод main() служить усього лише початком програми. Складна програма може включати десятки класів, тільки один з них повинен містити метод main(), щоб виконання було можливим.

Метод launch() запускає програму викликом методу start().

Модель програмування додатків платформи JavaFx

Один і той самий код JavaFx - додатків може запускатися в якості настільного додатка, який розвертається на клієнтському комп'ютері автономно, може розвертатися як додаток Java Web Start або відображатися в Web - браузері як JavaFx - апплет, вбудований в HTML - сторінку.

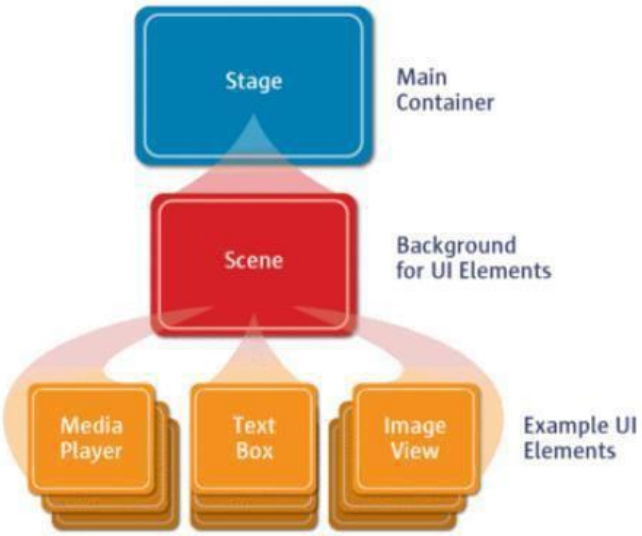
Крапкою входу в JavaFx-додатока служить Java - клас, що розширює абстрактний клас

javafx.application.Application і який містить метод main():

```
public class JavaFXApp extends Application { public static void main(String[] args) {
launch(args);

}
@Override
public void start(Stage primaryStage) {

//Встановлення параметрів сцени
. . .
primaryStage.setScene(scene);
primaryStage.setVisible(true);
}
```

Метод start() класу Application містить у якості параметра об'єкт Stage, що представляє графічний контейнер головного вікна JavaFx-додатка. Даний об'єкт Stage створюється середовищем виконання при запуску JavaFx-додатка і передається в метод start() головного класу JavaFx-додатка, що дозволяє використовувати методи об'єкта Stage для встановлення і відображення сцени JavaFx-додатка.

Перед встановленням і відображенням сцени в графічному контейнері Stage головного вікна JavaFx-додатка необхідно створити граф сцени, що складається з кореневого вузла і його дочірніх елементів, і на його основі створити об'єкт Scene сцени.



Дочірні вузли графа сцени, що представляють графіку, елементи контролю GUI-інтерфейсу, медіаконтент, додаються в кореневий вузол за допомогою методу getChildren().add() або методу getChildren().addAll(). При цьому дочірні вузли можуть мати візуальні ефекти, режими накладення, CSS-стилі, прозорість, трансформації, оброблювачі подій, брати участь в анімації по ключових кадрах, програмованої анімації та ін.

Розробка програмного забезпечення

Розробка програмного забезпечення здійснюється мовою програмування Java в інтегрованому середовищі розробки NetBeans IDE фірми Oracle.

ПОРЯДОК РОБОТИ:

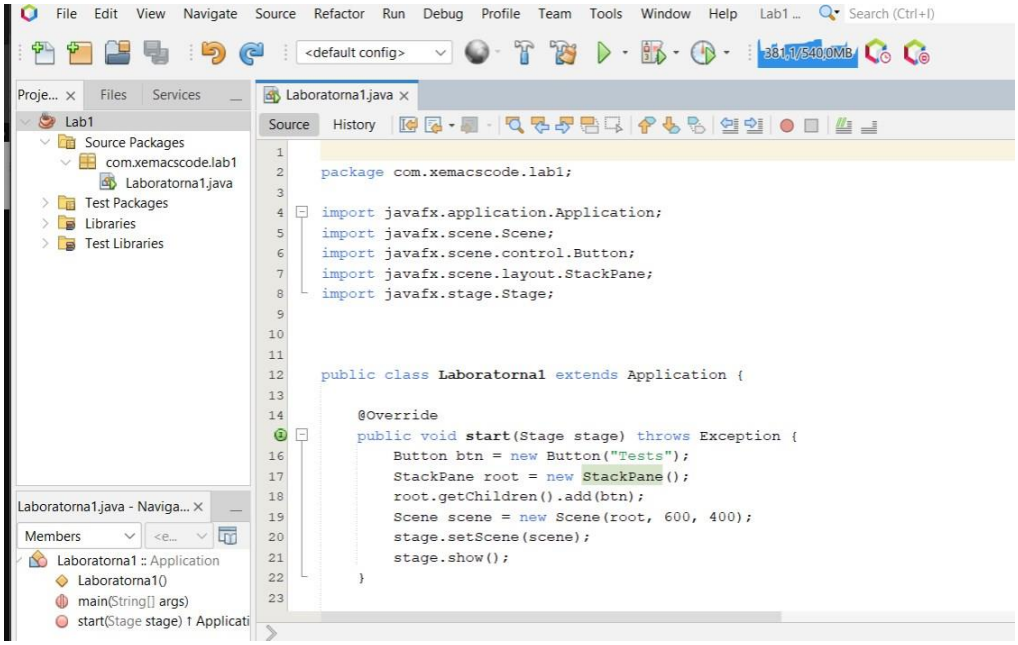
1. Створити свій директорій для файлів проекту.
2. Створити проект в інтегрованому середовищі розробки NetBeans IDE (c:\Temp\Ваша_Папка).
3. Написати програму мовою програмування Java (демонстраційна програма).
4. Скомпілювати програму.
5. Створити файл *.jar, який виконується.
6. Робота з NetBeans IDE

Створення проекту

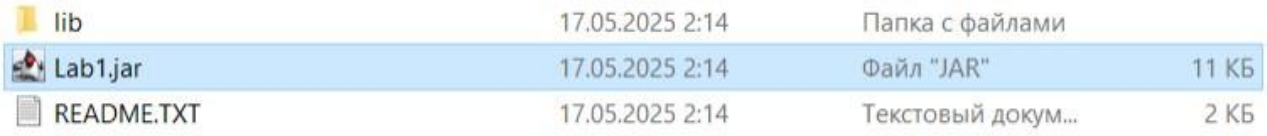
1. Запустити NetBeans IDE, у меню «Файл» вибрати пункт «Створити проект».
2. Вибрати пункт JavaFx, Додаток JavaFx і нажати кнопку «Далі».
3. Буде створений проект і шаблон програми:
4. Ввести код демонстраційної програми:

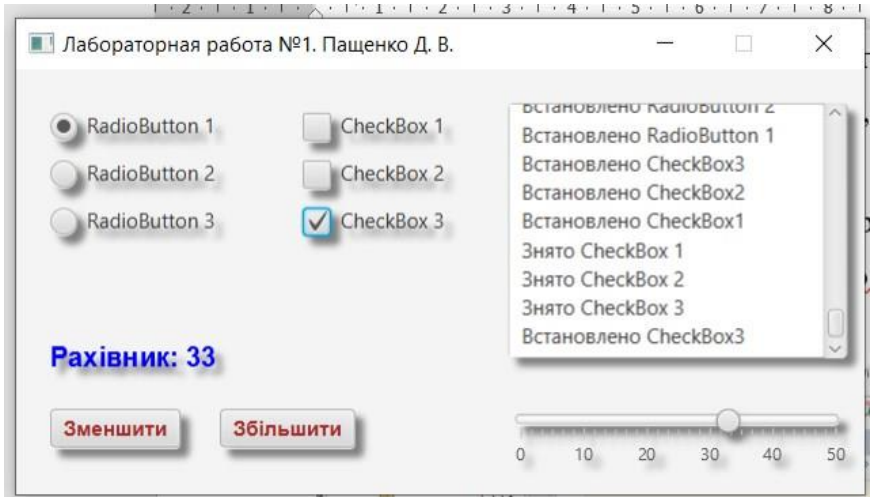
Приклад виконання програми

Створеня java-проекту за допомогою NetBeans IDE та перевірив, що все працює:



Створення демонстраційної програми з прикладу та *.jar – файл





Лістинг програми

```
//=============================================================
// Лабораторна робота № 1
// class: Laboratorna1
// Copyright (c) 2025 Pashchenko D. V.
//=============================================================
package com.xemacscode.lab1;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
```

```
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.CheckBox;
import javafx.scene.control.RadioButton;
import javafx.scene.control.Slider;
import javafx.scene.control.TextArea;
import javafx.scene.control.ToggleGroup;
import javafx.scene.effect.DropShadow;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;

/**
 *
 * @author Pashchenko D. V.
 */
//=================================================================
//Головний клас
//=================================================================
public class Laboratorna1 extends Application {
    Pane root_pane = new Pane();   //Група для вузлів сцени
    Group group = new Group();     //Група для вузлів сцени
    //=== елементи керування
    Slider slider;              //слайдер
    Button btn_inc;             //кнопка інкременту
    Button btn_dec;             //кнопка декременту
    //=== RadioButton
    RadioButton rb_1;           //1
    RadioButton rb_2;           //2
    RadioButton rb_3;           //3
    //=== CheckBox
    CheckBox cb_1;              //1
    CheckBox cb_2;              //2
    CheckBox cb_3;              //3
    //=== TextArea
    TextArea textArea;          //дісплей
    //=== графіка
    DropShadow dropShadow;      //тінь
    Text text;                  //текстовий напис
    //=== рахувач
    int counter = 0;
    int MAX_VAL = 50;           //максимальне значення рахунку

    //#################################################################
    //Створення вузлів (nodes)
```

```
//#################################################################
//=================================================================
//Тінь
//=================================================================
private void Shadow() {
   dropShadow = new DropShadow();      //створити об'єкт тіні
   dropShadow.setRadius(5.0);          //округлення кутів тіні
   dropShadow.setOffsetX(5.0);         //зміщення тіні по X та Y
   dropShadow.setOffsetY(5.0);
   dropShadow.setColor(Color.GRAY);    //колір тіні
}
//=================================================================
//Графіка
//=================================================================
private void CreateGraphNodes() {
   //=== створити тінь для вузлів сцени
   Shadow();
   //=== застосувати ефект для всіх вузлів групи
   //для класу Pane тінь не встановлюється, тому в Pane помістимо
   group = new Group();
   group.setEffect(dropShadow);
   //=== напис
   text = new Text();          //створити текст
   text.setFont(Font.font("Arial", FontWeight.BOLD, 16));
   text.setFill(Color.BLUE);   //колір тексту
}

//=================================================================
//Елементи керування
//=================================================================
private void CreateControlNodes() {
   //====== RadioButton
   ToggleGroup tg = new ToggleGroup();        //група для RadioButton
   rb_1 = new RadioButton("RadioButton 1");
   rb_1.setToggleGroup(tg);
   //
   rb_2 = new RadioButton("RadioButton 2");
   rb_2.setToggleGroup(tg);
   rb_2.setSelected(true);
   //
   rb_3 = new RadioButton("RadioButton 3");
   rb_3.setToggleGroup(tg);

   //====== CheckBox
   cb_1 = new CheckBox("CheckBox 1");
   cb_2 = new CheckBox("CheckBox 2");
   cb_3 = new CheckBox("CheckBox 3");

   //====== кнопка інкременту
   btn_inc = new Button();                  //створити кнопку
```

```
btn_inc.setText("Збільшити");              //напис на кнопці
btn_inc.setLayoutX(120);                   //розміщення кнопки на сцені
btn_inc.setLayoutY(200);
btn_inc.setTextFill(Color.BROWN);
btn_inc.setFont(Font.font("Arial", FontWeight.BOLD, 12));

//====== кнопка декременту
btn_dec = new Button();                    //створити кнопку
btn_dec.setText("Зменшити");               //напис на кнопці
btn_dec.setLayoutX(20);                    //розміщення кнопки на сцене
btn_dec.setLayoutY(200);
btn_dec.setTextFill(Color.BROWN);
btn_dec.setFont(Font.font("Arial", FontWeight.BOLD, 12));

//====== дісплей
textArea = new TextArea();
textArea.setLayoutX(290);
textArea.setLayoutY(20);
textArea.setPrefSize(200, 150);
textArea.setText("Лабораторная работа №1");
textArea.setWrapText(true);

//====== Слайдер
slider = new Slider();          //створити слайдер
slider.setLayoutX(290);         //координати
slider.setLayoutY(200);
slider.setPrefWidth(200);       //довжина слайдера
slider.setBlockIncrement(1.0);  //одиниця зміни
slider.setMajorTickUnit(10);    //великі ділення на шкалі
slider.setMinorTickCount(5);    //малі ділення на шкалі
slider.setMax(MAX_VAL);         //максимальні значення
slider.setMin(0);               //мінімальні значення
slider.setShowTickLabels(true); //показати значення шкали
slider.setShowTickMarks(true);  //показати ділення шкали
slider.setSnapToTicks(false);   //не прив'язувати к діленням
//=== ініціалізація слайдера
slider.setValue(counter);       //встановити повзунок слайдеру

//====== скомпонувати сцену
GridPane gp = new GridPane();   //створити компоновщик
gp.setHgap(50);                 //шаг по горизонталі
gp.setVgap(10);                 //шаг по вертикалі
gp.setPadding(new Insets(25, 0, 0, 20));    //відступи від краю вікна
//====== помістити об'єкти в компоновщик. (вузол, стовбець, рядок)
gp.add(rb_1, 0, 0);             //RadioButton
gp.add(rb_2, 0, 1);
gp.add(rb_3, 0, 2);
//
gp.add(cb_1, 1, 0);             //CheckBox
gp.add(cb_2, 1, 1);
```

```
    gp.add(cb_3, 1, 2);
    //
    gp.add(text, 0, 8);             //текст

    //====== помістити всі вузоли в корінь
    group.getChildren().addAll(gp, slider, btn_dec, btn_inc, textArea);
    root_pane.getChildren().addAll(group);
    //====== відобразити встановлені параметри
    toDisplay(counter);
}

//============================================================
//Обробник подій
//============================================================
private void onAction() {
    ////////////////////////////////////////////////////////
    //====== Button
    //=== Обробник події натискання кнопки
    btn_inc.setOnAction((ActionEvent event) -> {
        if (counter < MAX_VAL) {
            counter += 1;               //збільшити рахівник
        }
        toDisplay(counter);             //відобразити значення рахівника
    });

    //=== обробник події натискання кнопки
    btn_dec.setOnAction((ActionEvent event) -> {
        if (counter > 0) {
            counter -= 1;               //зменшити товщину лінії
        }
        toDisplay(counter);             //відобразити значення рахівника
    });

    ////////////////////////////////////////////////////////
    //====== Slider
    //=== обробник події повзунка слайдера
    slider.valueProperty().addListener((observable) -> {
        if (slider.isValueChanging()) {
            counter = (int) slider.getValue();  //отримати значення повзунка
            toDisplay(counter);                 //відобразити значення рахівника
        }
    });

    ////////////////////////////////////////////////////////
    //====== CheckBox
    //=== обробник події CheckBox_1
    cb_1.setOnAction((ActionEvent event) -> {
        if(cb_1.isSelected()){
            textArea.appendText("\nВстановлено CheckBox1");
        }else{
```

```
            textArea.appendText("\nЗнято CheckBox 1");
        }
    });

    //=== обробник події CheckBox_2
    cb_2.setOnAction((ActionEvent event) -> {
        if(cb_2.isSelected()){
            textArea.appendText("\nВстановлено CheckBox2");
        }else{
            textArea.appendText("\nЗнято CheckBox 2");
        }
    });

    //=== обробник події CheckBox_3
    cb_3.setOnAction((ActionEvent event) -> {
        if(cb_3.isSelected()){
            textArea.appendText("\nВстановлено CheckBox3");
        }else{
            textArea.appendText("\nЗнято CheckBox 3");
        }
    });

    ////////////////////////////////////////////////////////
    //====== RadioButton
    //=== обробник події RadioButton_1
    rb_1.setOnAction((ActionEvent t) -> {
        textArea.appendText("\nВстановлено RadioButton 1");
    });

    //=== обробник події RadioButton_2
    rb_2.setOnAction((ActionEvent t) -> {
        textArea.appendText("\nВстановлено RadioButton 2");
    });

    //=== обробник події RadioButton_3
    rb_3.setOnAction((ActionEvent t) -> {
        textArea.appendText("\nВстановлено RadioButton 3");
    });
}

//=============================================================
//Вивід значення рахівника
//=============================================================
public void toDisplay(double val) {
    String s = Integer.toString(counter);
    text.setText("Рахівник: " +s);          //значення рахівника
    textArea.appendText("\nРахівник: " + s); //значення рахівника
    slider.setValue(counter);              //встановити повзунок слайдера
}
```

```
    //=================================================================
    //Початок
    //=================================================================
    @Override
    public void start(Stage stage) throws Exception {
        //====== заголовок вікна
        stage.setTitle("Лабораторная работа №1. Пащенко Д. В.");
        stage.setResizable(false); //фіксований розмір
        //====== створити вузли сцени з графікою
        CreateGraphNodes();
        //====== створити вузли сцени з елементами керування
        CreateControlNodes();
        //====== створити обробник подій
        onAction();
        //root_pane.setOpacity(0.50);
        //====== створити сцену з розмірами та кольором фона
        Scene scene = new Scene(root_pane, 500, 250, Color.TRANSPARENT);
        //====== помістити сцену в вікно
        stage.setScene(scene);
        //====== відобразити вікно
        stage.show();
    }

    //=================================================================
    // вхід в програму (запускає метод start)
    //=================================================================
    public static void main(String[] args) {
        launch(args);
    }
}
```

Компіляція програми здійснюється натисканням кнопки F9.
Компіляція і запуск програми здійснюється натисканням кнопки F6.
Створення файлу, що виконується, здійснюється натисканням кнопки F11.

Контрольні питання:

1. Дати коротку характеристику мові програмування Java.
2. Призначення методу launch(args).
3. Призначення методу start(Stage primarystage).
4. Призначення об'єкта Stage.
5. Призначення об'єкта Scene.

ЛАБОРАТОРНА РОБОТА №2

Тема: СТВОРЕННЯ ДОДАТКА ДЛЯ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ

Мета роботи: Одержати навички створення додатків для паралельних обчислень.

Завдання:

1. Створити проект.
2. Вибрати колір, довжину і ширину прямокутника відповідно до варіанта.
3. Створити задачі і паралельні потоки обчислень.
4. У кожному потоці обчислити площу прямокутника в пікселах.
5. В окремому потоці обчислити сумарну площу прямокутників у пікселах.
6. Виконати програму

ТЕОРЕТИЧНІ ВІДОМОСТІ:

Java реалізує вбудовану підтримку багатопотокового програмування. Багатопотокова програма містить дві або більше частин, які можуть виконуватися одночасно.

Кожна частина такої програми називається потоком (Thread), і кожний потік задає окремий потік виконання. Інакше кажучи, багатопотоковість - це спеціалізована форма багатозадачності.

У середовищі потокової багатозадачності найменшим елементом керованого коду є потік.

Це означає, що одна програма може виконувати дві або більше задач одночасно.

Ще однією перевагою багатопоковості є зведення до мінімуму часу очікування. Це особливо важливо для інтерактивних мережних середовищ, у яких працює Java, тому що в них наявність очікування і простоїв звичайне явище. однопотокових середовищах програма змушена очікувати закінчення таких задач, перш ніж переходити до наступної, навіть якщо більшу частину часу програма простоює, очікуючи введення.

Багатопотоковість допомагає скоротити час простою, оскільки інші потоки можуть виконуватися, поки один очікує.

Клас Thread і інтерфейс Runnable Thread, що доповнює його. Багатопотокова система Java вбудована в клас unnable. Клас Thread інкапсулює потік виконання.

Щоб створити новий потік, програма повинна або розширити клас Thread, або реалізувати інтерфейс Runnable.

Клас Thread визначає кілька методів, які допомагають управляти потоками. Потік може знаходитися в одному зі станів, позначених наступними константами класу Thread.State:

NEW – потік створений, але ще не запущений;
RUNNABLE – потік виконується;
BLOCKED – потік блокований;
WAITING – потік чекає закінчення роботи іншого потоку; TERMINATED – потік закінчений.

Клас Thread

У класі Thread вісім конструкторів. Основний з них, Thread(Threadgroup group, Runnable target, String name, long stacksize); створює потік з іменем name, що належить групі

group і виконуючий метод run() об'єкта target. Останній параметр, stacksize, задає розмір стека і залежить від операційної системи.

Усі інші конструктори звертаються до нього з тим або іншим параметром, рівним null:

Thread() — створюваний потік буде виконувати свій метод run();
Thread(Runnable target);
Thread(Runnable target, String name);
Thread(String name);
Thread(Threadgroup group, Runnable target, String name);
Thread(Threadgroup group, Runnable target);
Thread(Threadgroup group, String name).

Ім'я потоку name не має ніякого значення, воно не використовується віртуальною машиною Java і застосовується тільки для розрізнення потоків програмі.

Після створення потоку його треба запустити методом start(). Віртуальна машина Java почне виконувати метод run() цього об'єкта-потоку. Потік завершить роботу після виконання методу run().

Потік можна призупинити статичним методом:

sleep(long ms);

на ms мілісекунд. Якщо обчислювальна система здатна відраховувати наносекунди, то можна призупинити потік з точністю до наносекунд методом:

sleep(long ms, int nanosec);

Коли програма Java стартує, негайно починає виконуватися один потік. Зазвичай його називають головним потоком (main thread) програми

Створення прямокутника

Геометрична фігура Rectangle представлена классом javafx.scene.shape.Rectangle, екземпляр якого можна створити за допомогою класу-фабрики RectangleBuilder або за допомогою одного з конструкторів:

Rectangle rectangle = new Rectangle();
Rectangle rectangle = new Rectangle(double width, double height);
Rectangle rectangle = new Rectangle(double width, double height, Paint fill);

Клас Rectangle представляє геометричний примітив - прямокутник і має, крім успадкованих від класу Shape властивостей, власні властивості:

A rcHeight, arcWidth, height, width, x, y.

Властивості x, y, height і width визначають координати і розміри прямокутника, а властивості arcHeight і arcWidth дозволяють створювати прямокутник із закругленими кутами, встановлюючи висоту і ширину дуги кута прямокутника.

Потік виконання створюється і запускається в такий спосіб

```
Thread th = new Thread(task);
th.setName(“Поток”);
th.start();
```

Перед запуском у потоці повинна бути розміщене задача, яка повинна бути виконана. Створюючи кілька потоків, у яких інкапсульовані виконувані задачі, створюється паралелізм обчислень.

Задачі створюються в такий спосіб:

```
Task<Void> task = new Task<Void>() {
@Override protected Void call() throws Exception { while (true) {
Platform.runLater(new Runnable() {
@Override public void run() {

//необхідні обчислення
}
});
//=== затримка

Thread.sleep(10);
}
}
};
```

В прикладі задача має тип <Void>, тому не повертає значень. Блок обчислень розміщується в чергу подій платформи.

Перед закриттям вікна необхідно зупинити всі запущені потоки в оброблювачі події закриття вікна:

Вказівки до виконання лабораторної роботи:

1. Створити свій директорій для файлів проекту.
2. Створити проект в інтегрованому середовищі розробки NetBeans IDE.
3. Вибрати колір, довжину і ширину прямокутника відповідно до варіанта.
4. Створити прямокутники.
5. Створити задачі для обчислень у потоках.
6. Створити паралельні потоки обчислень.
7. Помістити задачі для обчислень у потоки.
8. У кожному потоці обчислити площу прямокутника в пікселах.
9. В окремому потоці обчислити сумарну площу прямокутників у пікселах.
10. Скомпілювати і виконати програму.

Приклад виконання програми

1. Створив проект та обрав прямокутник за варіантом
(довжина - 190 пікс., ширина – 30 пікс., колір – рожевий).
2. Створив паралельні задачі та потоки, які обчислюють площу прямокутників, а також окрему задачу та потік для обчислення загальної площі.

Лістинг програми

```
//==============================================================
// Лабораторна робота № 2
// class: Laboratorna2
// Copyright (c) 2025 Pashchenko D. V.
//==============================================================
package com.xemacscode.lab2;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.scene.Group;
import javafx.concurrent.Task;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.effect.DropShadow;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.scene.input.MouseEvent;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.WindowEvent;
/**
 *
 * @author Pashchenko D. V.
 */
//==============================================================
//Головний клас
//==============================================================
```

```
public class Laboratorna2 extends Application {
    Pane root = new Pane();         //Група для вузлів сцени
    Group group = new Group();      //Група для вузлів сцени
    DropShadow dropShadow;          //тінь
    int MAX_HEIGHT = 190;           //максимальне значення довжини

    //=============================================================
    //Створення прямокутників
    //=============================================================
    private Rectangle createRectangle(double x, double y, double width, double height, Color color) {
        Rectangle rectangle = new Rectangle(x, y, width, height);
        rectangle.setFill(color);
        return rectangle;
    }
    //=============================================================
    //Зміна розміру прямокутників (тільки довжини)
    //=============================================================
    private void enableResize(Rectangle rectangle) {
        rectangle.setOnMouseDragged((MouseEvent event) -> {     //Зміна розміру при затисканні
клавіши миші
            double newHeight = event.getY() - rectangle.getY(); //Обчислення розміру
            if (newHeight > 1 && newHeight <= MAX_HEIGHT) {     //Перевіряємо розмір
                rectangle.setHeight(newHeight);                 //Задаємо новий розмір
            }
        });
    }


//=====================================================================
    //Задачи та потоки(1, 2, 3, 4) для обчислення площи кожного прямокутника

//=====================================================================
    private Task startCalculationTask(Rectangle rectangle, Text text) {
        Task<Void> task = new Task<>() {
            @Override
            protected Void call() throws Exception {
                while (true) {
                    Platform.runLater(() -> {
                        int area = (int) (rectangle.getWidth() * rectangle.getHeight()); //Обчислення площу
прямокутника
                        text.setText((Integer.toString(area)));                     //Вивід результату
                    });
                    Thread.sleep(100); // Затримка між обчисленнями
                }
            }
        };
        Thread thread = new Thread(task); //Створення потоку
        thread.setDaemon(true);         //Поток працює на фоні основного потоку програми
        thread.start();                 //Запуск потоку
```

```
        return task;
    }

    //=============================================================
    //Задача та поток(5) для обчислення площи прямокутників
    //=============================================================
    private Task startTotalTask(Text totalText, Rectangle... rectangles) {
        Task<Void> task = new Task<>() {
            @Override
            protected Void call() throws Exception {
                while (true) {
                    Platform.runLater(() -> {
                        int totalArea = 0;
                        for (Rectangle rect : rectangles) {
                            totalArea += (int)(rect.getWidth() * rect.getHeight());      //Обчислення загальної площі
                        }
                        totalText.setText("Сумарна площа: " + totalArea + " пікселів");  //Вивід результату
                    });
                    Thread.sleep(100); // Затримка між обчисленнями
                }
            }
        };
        Thread thread = new Thread(task); //Створення потоку
        thread.setDaemon(true);          //Поток працює на фоні основного потоку програми
        thread.start();                  //Запуск потоку

        return task;
    }

    //=============================================================
    //Створення текстів
    //=============================================================
    private Text createText(Rectangle rectangle, String initialText) {
        Text text = new Text(rectangle.getX(), rectangle.getY() - 25, initialText); //створення тексту
        text.setFont(Font.font("Arial", FontWeight.BOLD, 16)); //Розмір тексту та шрифт, жирність тексту
        text.setFill(Color.RED);  //Колір тексту
        return text;
    }
    //далі так само
    private Text createLabel(Rectangle rectangle, String initialText) {
        Text textLabel = new Text(rectangle.getX() - 10, rectangle.getY() - 5, initialText);
        textLabel.setFont(Font.font("Arial", FontWeight.BOLD, 16));
        textLabel.setFill(Color.GRAY);
        return textLabel;
    }

    private Text createTotalText(String initialText) {
```

```
    Text totalText = new Text(125, 320, initialText);
    totalText.setFont(Font.font("Arial", FontWeight.BOLD, 16));
    totalText.setFill(Color.RED);
    return totalText;
}

private Text createTotalTextThreadText(String initialText) {
    Text threadText = new Text(50, 320, initialText);
    threadText.setFont(Font.font("Arial", FontWeight.BOLD, 16));
    threadText.setFill(Color.GRAY);
    return threadText;
}

private Text createTopText(String initialText) {
    Text topText = new Text(95, 20 , initialText);
    topText.setFont(Font.font("Arial", FontWeight.BOLD, 16));
    topText.setFill(Color.RED);
    return topText;
}

//=============================================================
//Тінь
//=============================================================
private void Shadow() {
    dropShadow = new DropShadow();      //створити об'єкт тіні
    dropShadow.setRadius(10.0);         //округлення кутів тіні
    dropShadow.setOffsetX(7.0);         //зміщення тіні по X та Y
    dropShadow.setOffsetY(10.0);
    dropShadow.setColor(Color.GRAY);    //колір тіні
}

//=============================================================
//Початок
//=============================================================
@Override
public void start(Stage stage) throws Exception {
    //Створення тіні
    Shadow();

    // Створення прямокутників
    Rectangle rect1 = createRectangle(50, 70, 30, 190, Color.PINK);
    Rectangle rect2 = createRectangle(150, 70, 30, 190, Color.PINK);
    Rectangle rect3 = createRectangle(250, 70, 30, 190, Color.PINK);
    Rectangle rect4 = createRectangle(350, 70, 30, 190, Color.PINK);

    //Створенян текстів
    Text topText = createTopText("Площі прямокутників в пікселях");

    Text rect1Text = createText(rect1, "0");
    Text rect1Label = createLabel(rect1, "Поток 1");
```

```
Text rect2Text = createText(rect2, "0");
Text rect2Label = createLabel(rect2, "Поток 2");
Text rect3Text = createText(rect3, "0");
Text rect3Label = createLabel(rect3, "Поток 3");
Text rect4Text = createText(rect4, "0");
Text rect4Label = createLabel(rect4, "Поток 4");

Text totalText = createTotalText("Сумарна площа: 0 пікселів");
Text threadText = createTotalTextThreadText("Поток 5");

//застосування можливості зміни розміру прямокутників
enableResize(rect1);
enableResize(rect2);
enableResize(rect3);
enableResize(rect4);

//Додавання елементів в групу
group.getChildren().addAll(rect1, rect2, rect3, rect4,
        rect1Text, rect1Label, rect2Text, rect2Label,
        rect3Text, rect3Label, rect4Text, rect4Label,
        topText, totalText, threadText);

//Застосування тіні до елементів
group.setEffect(dropShadow);

// Додавання елементів на панель
root.getChildren().addAll(group);

// Створення сцени
Scene scene = new Scene(root, 420, 350, Color.TRANSPARENT);

// Встановлення сцени та відображення вікна
stage.setTitle("Лабораторная работа №2. Пащенко Д. В.");
stage.setScene(scene);
stage.show();

// Виконання обчислень у потоках
Task task1 = startCalculationTask(rect1, rect1Text);
Task task2 = startCalculationTask(rect2, rect2Text);
Task task3 = startCalculationTask(rect3, rect3Text);
Task task4 = startCalculationTask(rect4, rect4Text);
Task task5 = startTotalTask(totalText, rect1, rect2, rect3, rect4);

//Закриття задач при закритті вікна програми
stage.setOnCloseRequest(new EventHandler<WindowEvent>() {
    @Override
    public void handle(WindowEvent event) {
        task1.cancel();
        task2.cancel();
        task3.cancel();
```

```
            task4.cancel();
            task5.cancel();
            System.out.println("Threads closed"); //Повідомлення в консоль про закриття задач
        }
    });
  }

  //=============================================================
  // Точка входу в програму (запускає метод start)
  //=============================================================
  public static void main(String[] args) {
    launch(args);
  }
}
```

ВАРІАНТИ:

	Довжина, пікс	Ширина, пікс	Колір
0	110	20	червоний
1	120	30	сірий
2	130	40	синій
3	140	50	зелений
4	150	60	кавовий
5	160	60	чорний
6	170	50	фіолетовий
7	180	40	блакитний
8	190	30	рожевий
9	200	20	темно-сірий

Контрольні питання:

1. Порядок створення прямокутника і текстового об'єкта.
2. Порядок створення задачі для обчислень.
3. Порядок створення і запуску потоків обчислень.
4. Порядок закриття потоків при виході із програми.
5. Призначення затримки в циклі виконання задачі.

ЛАБОРАТОРНА РОБОТА №3

Тема: КЕРУВАННЯ ПРОЦЕСОМ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ

Мета роботи: Одержати навички керування процесом паралельних обчислень.

Завдання:

1. Створити проект.
2. Вибрати колір, довжину і ширину прямокутника відповідно до варіанта.
3. Створити задачу і паралельні потоки обчислень.
4. У кожному потоці обчислити площу прямокутника в пікселах.
5. В окремому потоці обчислити сумарну площу прямокутників у пікселах.
6. Виконати програму

ТЕОРЕТИЧНІ ВІДОМОСТІ:

Створення потоку
Java для цього визначено два образи.

1. За допомогою реалізації інтерфейсу Runnable.
2. За допомогою розширення класу Thread.

Використання інтерфейсу Runnable для створення потоку

Найпростіший спосіб створення потоку - це оголошення класу, який реалізує інтерфейс Runnable.

Можна створити потік з будь-якого об'єкта, який реалізує інтерфейс реалізувати інтерфейс Runnable, клас повинен оголосити єдиний метод run().

Усередині методу run() треба визначити код, який, буде виконуватися в потоці.

Метод run() встановлює крапку входу для іншого, паралельного потоку усередині програми. Цей потік завершиться, коли метод run() поверне керування.

Після того як буде оголошений клас, який реалізує інтерфейс Runnable, треба створити об'єкт типу Thread із цього класу. У класі Thread визначено кілька конструкторів.

Той, який повинен використовуватися в цьому випадку, виглядає таким чином.

Thread(Runnable об'єкт_потоку, String ім'я_потоку)

У цьому конструкторі об'єкт_потоку – це екземпляр класу, який реалізує інтерфейс Runnable. Він визначає, де почнеться виконання потоку. Ім'я нового потоку передається в параметрі імені_потоку.

Після того як новий потік буде створений, він не запускається доти, поки не буде викликаний метод start(), оголошений у класі Thread.

Метод start() виконує виклик методу run().

Приклад виконання програми

1. Створив проект та обрав прямокутник за варіантом
(довжина - 190 пікс., ширина – 30 пікс., колір – рожевий).
2. Створив паралельні задачі та потоки, які обчислюють площу прямокутників, а також керування ними, а також окрему задачу та поток для обчислення загальної площі.

Лістинг програми

```
//============================================================
// Лабораторна робота № 3
// class: Laboratorna3
// Copyright (c) 2025 Pashchenko D. V.
//============================================================
package com.xemacscode.lab3;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.scene.Group;
import javafx.concurrent.Task;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.RadioButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.effect.DropShadow;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
import javafx.scene.input.MouseEvent;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.WindowEvent;

/**
 *
 * @author Pashchenko D. V.
 */
//============================================================
//Головний клас
//============================================================
public class Laboratorna3 extends Application {
    Pane root = new Pane();         //Група для вузлів сцени
    Group group = new Group();      //Група для вузлів сцени
    DropShadow dropShadow;          //Тінь
    int MAX_HEIGHT = 190;           //максимальне значення довжини

    Thread[] threads = new Thread[4];    //Група потоків
    Task<Void>[] tasks = new Task[4];     //Група задач
    boolean[] taskActive = {false, false, false, false};    //Прапори активності задач/потоків
```

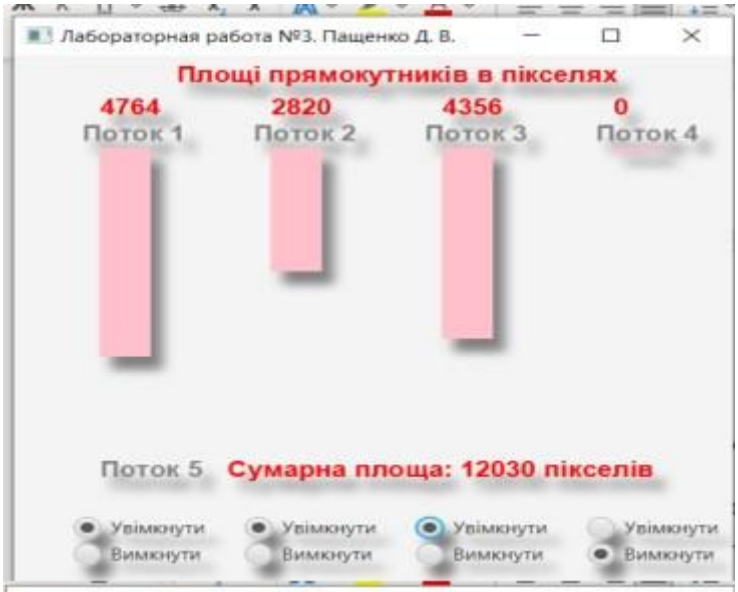
```
    //=================================================================
    //Створення прямокутників
    //=================================================================
    private Rectangle createRectangle(double x, double y, double width, double height, Color color) {
        Rectangle rectangle = new Rectangle(x, y, width, height);
        rectangle.setFill(color);
        return rectangle;
    }

    //=================================================================
    //Зміна розміру прямокутників (тільки довжини)
    //=================================================================
    private void enableResize(Rectangle rectangle, int index) {
        rectangle.setOnMouseDragged((MouseEvent event) -> { //Зміна розміру при затисканні
клавіши миші
            if(!taskActive[index]) {  //Перевірка чи увікнена задача
                return;
            }

            double newHeight = event.getY() - rectangle.getY(); //Обчислення розміру
            if (newHeight > 1 && newHeight <= MAX_HEIGHT) {     //Перевіряємо розмір
                rectangle.setHeight(newHeight);
        });
    }


//=====================================================================
    //Задачи та потоки(1, 2, 3, 4) для обчислення площи кожного прямокутника

//=====================================================================
    private void startCalculationTask(Rectangle rectangle, Text text, int index) {
        tasks[index] = new Task<>() {
            @Override
            protected Void call() throws Exception {
                while (true) {
                    if (taskActive[index]) {   //перевірка, якщо задача активна
                        Platform.runLater(() -> {
                            enableResize(rectangle, index); //застосування можливості зміни розміру
прямокутників
                            int area = (int) (rectangle.getWidth() * rectangle.getHeight()); //Обчислення
площу прямокутника
                            text.setText(Integer.toString(area));  //Вивід результату
});
}
                    else {
                        rectangle.setHeight(3);  //Встановлення мінімальної висоти прямокутника, якщо
задача вимкнена
                        text.setText(Integer.toString(0)); //Встановлення площини прямокутника нуль,
якщо задача вимкнена
```

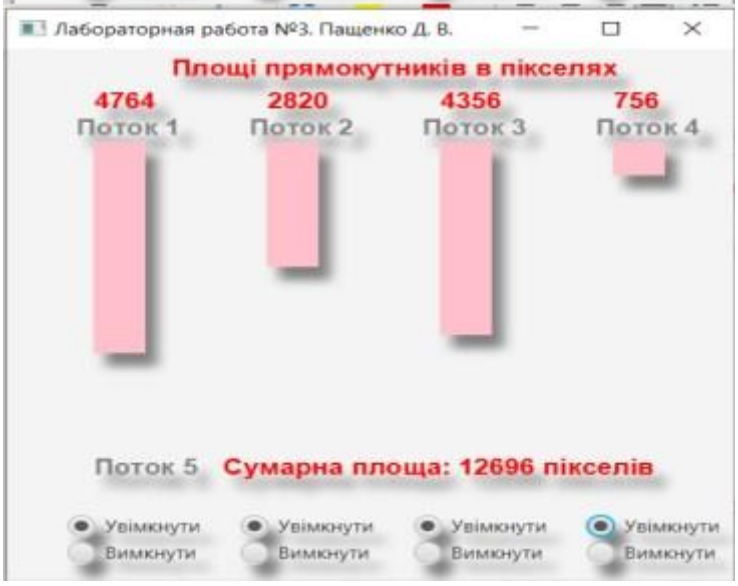
```
                }
                Thread.sleep(100); // Затримка між обчисленнями
            }
        }
    };
    threads[index] = new Thread(tasks[index]);  //Створення потоку
    threads[index].setDaemon(true); //Поток працює на фоні основного потоку програми
    threads[index].start();//Запуск потоку
}

//=================================================================
//Задача та поток(5) для обчислення площи прямокутників
//=================================================================
private Task startTotalTask(Text totalText, Rectangle... rectangles) {
    Task<Void> task = new Task<>() {
        @Override
        protected Void call() throws Exception {
            while (true) {
                Platform.runLater(() -> {
                    int totalArea = 0;
                    for (Rectangle rect : rectangles) {
                        totalArea += (int)(rect.getWidth() * rect.getHeight());//Обчислення загальної
площи
}
totalText.setText("Сумарна площа: " + totalArea + " пікселів");  //Вивід результату
                });
                Thread.sleep(100); // Затримка між обчисленнями
            }
        }
    };
    Thread thread = new Thread(task); //Створення потоку
    thread.setDaemon(true);          //Поток працює на фоні основного потоку програми
    thread.start();                  //Запуск потоку

    return task;
}

//=================================================================
//Створення текстів
//=================================================================
private Text createText(Rectangle rectangle, String initialText) {
    Text text = new Text(rectangle.getX(), rectangle.getY() - 25, initialText); //створення тексту
    text.setFont(Font.font("Arial", FontWeight.BOLD, 16 //Розмір тексту та шрифт, жирність
тексту
    text.setFill(Color.RED);  //Колір тексту
    return text;
}
//далі так само
private Text createLabel(Rectangle rectangle, String initialText) {
    Text textLabel = new Text(rectangle.getX() - 10, rectangle.getY() - 5, initialText);
```

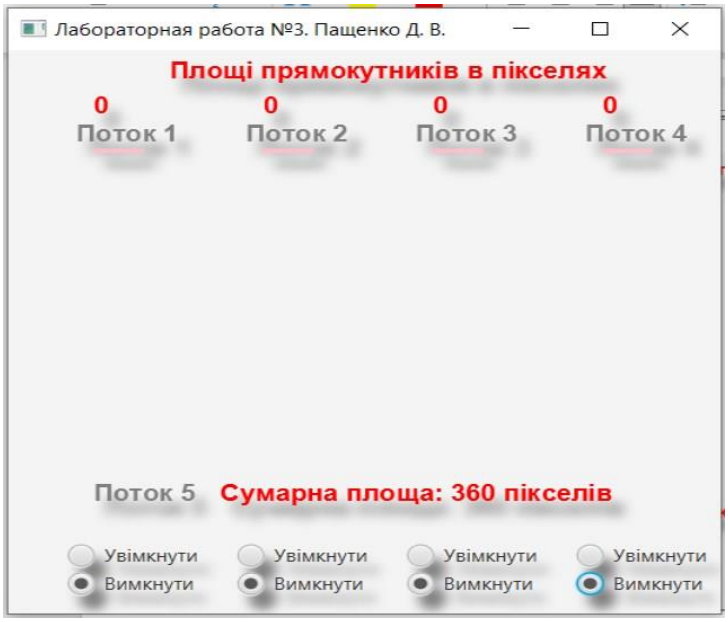
```
        textLabel.setFont(Font.font("Arial", FontWeight.BOLD, 16));
        textLabel.setFill(Color.GRAY);
        return textLabel;
    }

    private Text createTotalText(String initialText) {
        Text totalText = new Text(125, 320, initialText);
        totalText.setFont(Font.font("Arial", FontWeight.BOLD, 16));
        totalText.setFill(Color.RED);
        return totalText;
    }

    private Text createTotalTextThreadText(String initialText) {
        Text threadText = new Text(50, 320, initialText);
        threadText.setFont(Font.font("Arial", FontWeight.BOLD, 16));
        threadText.setFill(Color.GRAY);
        return threadText;
    }

    private Text createTopText(String initialText) {
        Text topText = new Text(95, 20 , initialText);
        topText.setFont(Font.font("Arial", FontWeight.BOLD, 16));
        topText.setFill(Color.RED);
        return topText;
    }

    //=============================================================
    //Створення RadioButtons
    //=============================================================
    private void createRadioButtons(Rectangle rectangle, int index) {
        ToggleGroup toggleGroup = new ToggleGroup();//Створення об'єкта група перемикачів

        //Група перемикачів, які вмикають задачі/потоки
        RadioButton enableButton = new RadioButton("Увімкнути");
        enableButton.setToggleGroup(toggleGroup);    //Додавання до групи
        enableButton.setLayoutX(rectangle.getX() - 15); //Позиція по X та Y
        enableButton.setLayoutY(350);
        enableButton.setOnAction(e -> taskActive[index] = true);  //Додавання події активності
перемикача

        //Група перемикачів, які вимикають задачі/потоки
        RadioButton disableButton = new RadioButton("Вимкнути");
        disableButton.setToggleGroup(toggleGroup);
        disableButton.setSelected(true); //Увімкнутий за замовчуванням
        disableButton.setLayoutX(rectangle.getX() - 15);
        disableButton.setLayoutY(370);
        disableButton.setOnAction(e -> taskActive[index] = false);

        group.getChildren().addAll(enableButton, disableButton); //Додавання перемикачів на
панель
```

```
}

//=================================================================
//Тінь
//=================================================================
private void Shadow() {
   dropShadow = new DropShadow();      //створити об'єкт тіні
   dropShadow.setRadius(10.0);         //округлення кутів тіні
   dropShadow.setOffsetX(7.0);         //зміщення тіні по X та Y
   dropShadow.setOffsetY(10.0);
   dropShadow.setColor(Color.GRAY);    //колір тіні
}

//=================================================================
//Початок
//=================================================================
@Override
public void start(Stage stage) throws Exception {
   //Створення тіні
   Shadow();

   // Створення прямокутників
   Rectangle rect1 = createRectangle(50, 70, 30, 3, Color.PINK);
   Rectangle rect2 = createRectangle(150, 70, 30, 3, Color.PINK);
   Rectangle rect3 = createRectangle(250, 70, 30, 3, Color.PINK);
   Rectangle rect4 = createRectangle(350, 70, 30, 3, Color.PINK);

   //Створенян текстів
   Text topText = createTopText("Площі прямокутників в пікселях");

   Text rect1Text = createText(rect1, "0");
   Text rect1Label = createLabel(rect1, "Поток 1");
   Text rect2Text = createText(rect2, "0");
   Text rect2Label = createLabel(rect2, "Поток 2");
   Text rect3Text = createText(rect3, "0");
   Text rect3Label = createLabel(rect3, "Поток 3");
   Text rect4Text = createText(rect4, "0");
   Text rect4Label = createLabel(rect4, "Поток 4");

   Text totalText = createTotalText("Сумарна площа: 0 пікселів");
   Text threadText = createTotalTextThreadText("Поток 5");

   //Створення RadioButtons
   createRadioButtons(rect1, 0);
   createRadioButtons(rect2, 1);
   createRadioButtons(rect3, 2);
   createRadioButtons(rect4, 3);

   //Додавання елементів в групу
   group.getChildren().addAll(rect1, rect2, rect3, rect4,
```

```
                rect1Text, rect1Label, rect2Text, rect2Label,
                rect3Text, rect3Label, rect4Text, rect4Label,
                topText, totalText, threadText);

        //Застосування тіні до елементів
        group.setEffect(dropShadow);

        // Додавання елементів на панель
        root.getChildren().addAll(group);

        // Створення сцени
        Scene scene = new Scene(root, 420, 400, Color.TRANSPARENT);

        // Встановлення сцени та відображення вікна
        stage.setTitle("Лабораторная работа №3. Пащенко Д. В.");
        stage.setScene(scene);
        stage.show();

        // Виконання обчислень у потоках
        startCalculationTask(rect1, rect1Text, 0);
        startCalculationTask(rect2, rect2Text, 1);
        startCalculationTask(rect3, rect3Text, 2);
        startCalculationTask(rect4, rect4Text, 3);
        Task task5 = startTotalTask(totalText, rect1, rect2, rect3, rect4);

        //Закриття задач при закритті вікна програми
        stage.setOnCloseRequest(new EventHandler<WindowEvent>() {
            @Override
            public void handle(WindowEvent event) {
                for (Task task : tasks) {
                    task.cancel();
                }
                task5.cancel();
                System.out.println("Threads closed"); //Повідомлення в консоль про закриття задач
            }
        });
    }

    //============================================================
    // Точка входу в програму (запускає метод start)
    //============================================================
    public static void main(String[] args) {
        launch(args);
    }
}
```


Лабораторная работа №3. Пащенко Д. В.
Площі прямокутників в пікселях
4764
Поток 1
2820
Поток 2
4356
Поток 3
0
Поток 4
Поток 5
Сумарна площа: 12030 пікселів
Увімкнути
Вимкнути
Увімкнути
Вимкнути
Увімкнути
Вимкнути
Увімкнути
Вимкнути


Лабораторная работа №3. Пащенко Д. В.
Площі прямокутників в пікселях
4764
Поток 1
2820
Поток 2
4356
Поток 3
756
Поток 4
Поток 5
Сумарна площа: 12696 пікселів
Увімкнути
Вимкнути
Увімкнути
Вимкнути
Увімкнути
Вимкнути
Увімкнути
Вимкнути


Лабораторная работа №3. Пащенко Д. В.
Площі прямокутників в пікселях
0
Поток 1
0
Поток 2
0
Поток 3
0
Поток 4
Поток 5
Сумарна площа: 360 пікселів
Увімкнути
Вимкнути
Увімкнути
Вимкнути
Увімкнути
Вимкнути
Увімкнути
Вимкнути

Контрольні питання:

1. Способи створення потоків.
2. Призначення інтерфейсу Runnable.
3. Призначення класу Task.
4. Керування потоками в додатку.
5. Видалення потоків при закритті вікна додатка.
6. .

ВАРІАНТИ:

	Довжина, пікс	Ширина, пікс	Колір
0	110	20	червоний
1	120	30	сірий
2	130	40	синій
3	140	50	зелений
4	150	60	кавовий
5	160	60	чорний
6	170	50	фіолетовий
7	180	40	блакитний
8	190	30	рожевий
9	200	20	темно-сірий

ЛАБОРАТОРНА РОБОТА № 4

Тема: ПАРАЛЕЛЬНІ ОБЧИСЛЕННЯ В БАГАТОПРОЦЕСОРНИХ СИСТЕМАХ. ТЕХНОЛОГІЯ FORK-JOIN

Мета роботи: Одержати навички створення паралельних обчислень у багатопроцесорних системах.

Завдання:

1. Створити додаток, що реалізує паралельні обчислення в багатопроцесорних системах.
2. Реалізувати паралелізм на рівні окремих процесорів.
3. Зробити аналіз результатів виконання паралельних обчислень для одного і декількох процесорів.
4. Зробити аналіз ефективності використання великої кількості паралельних потоків обчислень.
5. Зробити висновок про оптимальну кількість процесорів в обчислювальній системі з погляду співвідношення ефективності/вартості.
6. Зробити висновок про оптимальну кількість потоків в обчислювальній системі для оптимальної кількості процесорів.

ТЕОРЕТИЧНІ ВІДОМОСТІ:

Fork-Join - метод, застосовуваний у комунікаційних і комп'ютерних системах і служить для прогнозування продуктивності виконання великої кількості робочих задач.

Метод полягає в тому, що кожна задача розбивається на безліч синхронізованих задач, які обробляються паралельно на різних серверах.

Суть методу проста: велика задача розбивається на задачі поменше, ті, у свою чергу, на ще більш дрібні задачі, і так доти, поки це має сенс.

У самому кінці отримана тривіальна задача виконується послідовно. Даний етап називається Fork

Результат виконання послідовних задач об'єднується вгору по ланцюжку, поки не вийде рішення самої верхньої задачі.

Даний етап називається Join. Виконання всіх задач відбувається паралельно.

Приклад виконання програми

Створив проект, який обчислює середнє арифметичне.

Лістинг програми

```
//============================================================
// Лабораторна робота № 4
// class: Laboratorna4
// Copyright (c) 2025 Pashchenko D. V.
//============================================================
package com.xemacscode.lab4;

import javafx.application.Application;
import javafx.scene.Scene;
```

```
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.stage.Stage;

import java.util.Random;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import javafx.application.Platform;
import javafx.scene.Group;
import javafx.scene.effect.DropShadow;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;

/**
 *
 * @author Pashchenko D. V.
 */
public class Laboratorna4 extends Application{

    Pane root = new Pane();         //Група для вузлів сцени
    Group group = new Group();      //Група для вузлів сцени
    DropShadow dropShadow;          //Тінь

    GridPane processorGrid = new GridPane();    //Група для CheckBox
    Text arraySizeText;                         //Розмір масива текст
    Text threadCountText;                       //Кількість потоків текст

    //============================================================
    //Тінь
    //============================================================
    private void Shadow() {
        dropShadow = new DropShadow();      //створити об'єкт тіні
        dropShadow.setRadius(10.0);         //округлення кутів тіні
        dropShadow.setOffsetX(7.0);         //зміщення тіні по X та Y
        dropShadow.setOffsetY(10.0);
        dropShadow.setColor(Color.GRAY);    //колір тіні
    }

    //============================================================
    //Середньє арифметичне
    //============================================================
    public void calculateAverage(TextField arraySizeField, TextField threadCountField,
                                 TextArea resultArea, CheckBox[] processorCheckboxes) {
        try {
            int arraySize = Integer.parseInt(arraySizeField.getText());//Отримує int значення з тексту
            int threadCount = Integer.parseInt(threadCountField.getText());
```

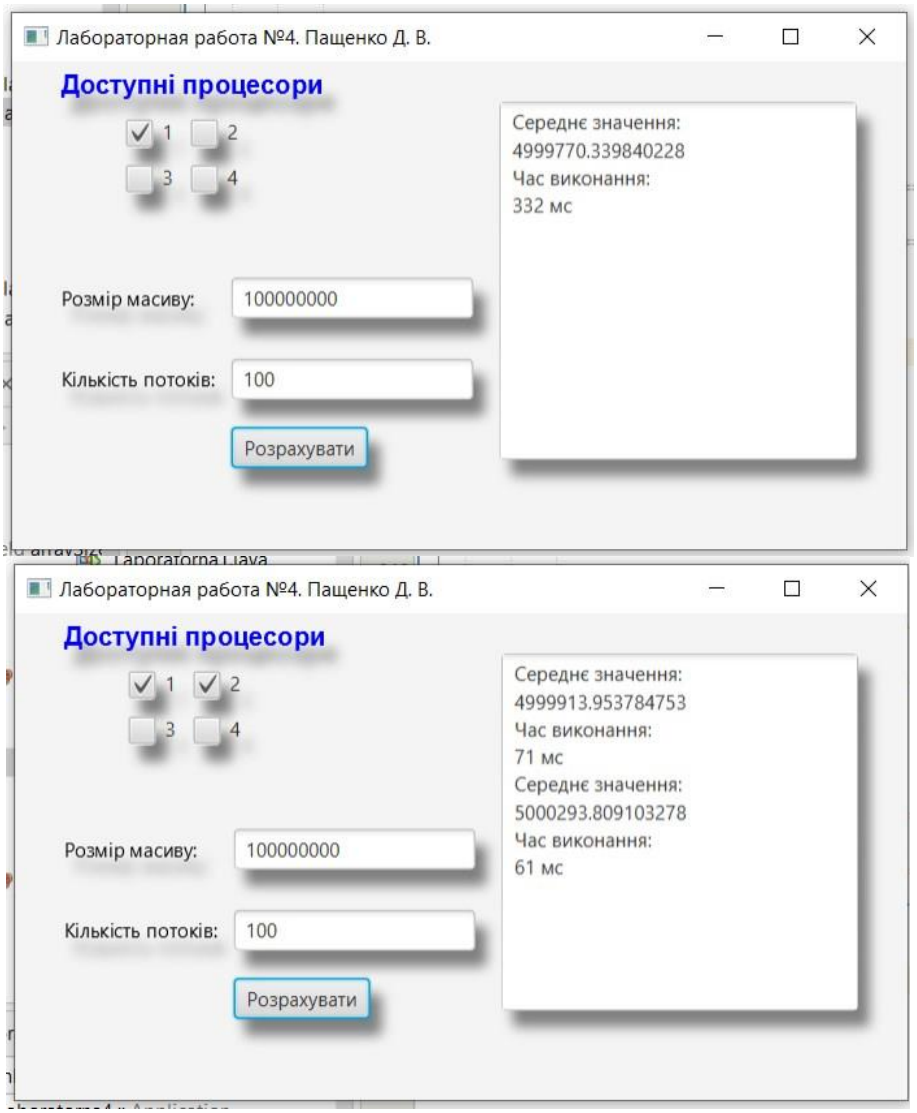
```
        //Перевірка розміру масиву та кількості потоків
        if (arraySize <= 0 || threadCount <= 0) {
            resultArea.appendText("Розмір масиву та кількість потоків мають бути більшими за
0\n");
            return;
        }

        //Перевірка, які процесори вибрані
        boolean[] processors = new boolean[processorCheckboxes.length];
        int activeProcessorCount = 0;  // Лічильник обраних процесорів
        for (int i = 0; i < processorCheckboxes.length; i++) {
            processors[i] = processorCheckboxes[i].isSelected();
            if (processors[i]) {
                activeProcessorCount++;
            }
        }

        //Якщо жоден процесор не вибраний
        if (processors.length == 0) {
            resultArea.appendText("Будь ласка, оберіть хоча б один процесор.\n");
            return;
        }

        //Генерація масиву випадкових чисел
        double[] array = new double[arraySize];
        Random random = new Random();
        for (int i = 0; i < arraySize; i++) {
            array[i] = random.nextDouble() * 10000000; //Випадкові числа від 0 до 10000000
        }

        long startTime = System.nanoTime();  //Початковий час

        //Розподіл обчислень між потоками
        int effectiveThreadCount = threadCount * activeProcessorCount; // Збільшення кількості
потоків залежно від процесорів
        ExecutorService executor = Executors.newFixedThreadPool(effectiveThreadCount);
        double[] partialSums = new double[effectiveThreadCount];
        int chunkSize = arraySize / effectiveThreadCount;

        for (int th = 0; th < effectiveThreadCount; th++) {
            final int start = th * chunkSize;
            final int end = (th == effectiveThreadCount - 1) ? arraySize : start + chunkSize;
            final int index = th;

            executor.submit(() -> { //Завдання для кожного потоку
                double sum = 0;
                for (int i = start; i < end; i++) {
                    sum += array[i];
                }
                partialSums[index] = sum;
```

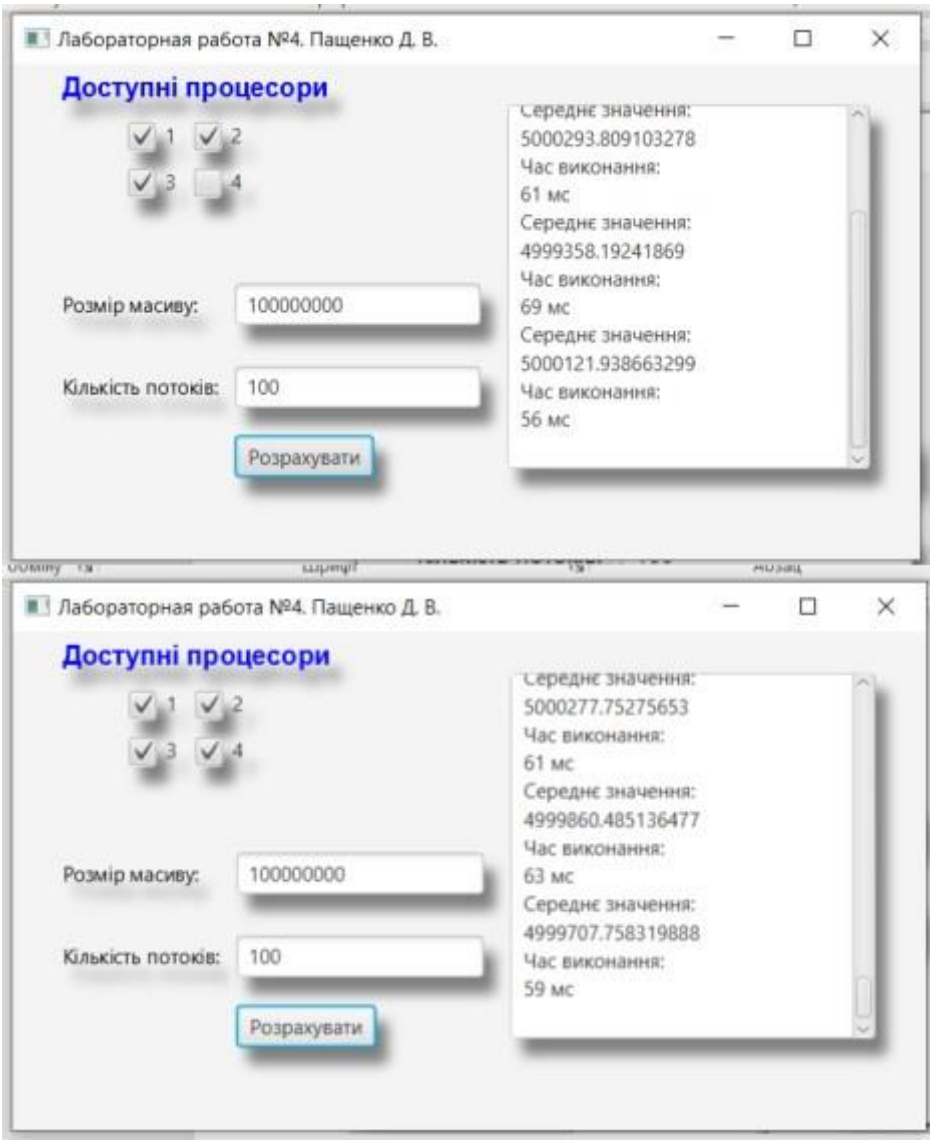
```
            });
        }

        executor.shutdown();//Завершення виконання потоків
        executor.awaitTermination(1, TimeUnit.MINUTES);

        //Обчислення загальної суми та середнього значення
        double totalSum = 0;
        for (double sum : partialSums) {
            totalSum += sum;
        }
        double average = totalSum / arraySize;

        long endTime = System.nanoTime();  //Кінцевий час виконання
        long duration = TimeUnit.NANOSECONDS.toMillis(endTime - startTime); //Час
виконання в мілісекундах

        //Виведення результатів
        Platform.runLater(() -> {
            resultArea.appendText("Середнє значення:\n" + average + "\n");
            resultArea.appendText("Час виконання:\n" + duration + " мс\n");
        });

        //Обробка помилок введення
    } catch (NumberFormatException e) {
        resultArea.appendText("Будь ласка, введіть коректні числові значення для розміру
масиву та кількості потоків.\n");
        //Обробка переривання виконання
    } catch (InterruptedException e) {
        resultArea.appendText("Обчислення було перервано.\n");
    }
}

//============================================================
//Створення елементів інтерфейсу
//============================================================

//Створення підпису для процесорів
private Text createProcessorsText(String initialText) {
    Text processorsText = new Text(30, 20, initialText);
    processorsText.setFont(Font.font("Arial", FontWeight.BOLD, 16));
    processorsText.setFill(Color.BLUE);

    return processorsText;
}

//Створення CheckBox для процесорів
private CheckBox[] createProcessorsPane() {
    CheckBox[] processorCheckboxes = new CheckBox[4];
    for (int i = 0; i < 4; i++) {
```

```
            processorCheckboxes[i] = new CheckBox(String.valueOf(i + 1));
            processorGrid.add(processorCheckboxes[i], i % 2, i / 2);
        }

        processorGrid.setHgap(10);
        processorGrid.setVgap(10);
        processorGrid.setLayoutX(70);
        processorGrid.setLayoutY(35);

        return processorCheckboxes;
    }

    //Створення поля для розміру масиву
    private TextField createArraySizePane(String initialText) {
        arraySizeText = new Text(30, 150, initialText);
        TextField arraySizeField = new TextField();
        arraySizeField.setLayoutX(135);
        arraySizeField.setLayoutY(133);

        return arraySizeField;
    }

    //Створення поля для кількості потоків
    private TextField createThreadCountPane(String initialText) {
        threadCountText = new Text(30, 200, initialText);
        TextField threadCountField = new TextField();
        threadCountField.setLayoutX(135);
        threadCountField.setLayoutY(183);

        return threadCountField;
    }

    //Створення текстової області для результатів
    private TextArea createResultArea() {
        TextArea resultArea = new TextArea();
        resultArea.setEditable(false);
        resultArea.setLayoutX(300);
        resultArea.setLayoutY(25);
        resultArea.setPrefWidth(220);
        resultArea.setPrefHeight(220);

        return resultArea;
    }

    //Створення кнопки розрахунку
    private Button createComputeButton(String initialString, TextField arraySizeField, TextField
threadCountField, CheckBox[] processorCheckboxes, TextArea resultArea) {
        Button computeButton = new Button(initialString);
        computeButton.setLayoutX(135);
        computeButton.setLayoutY(225);
```

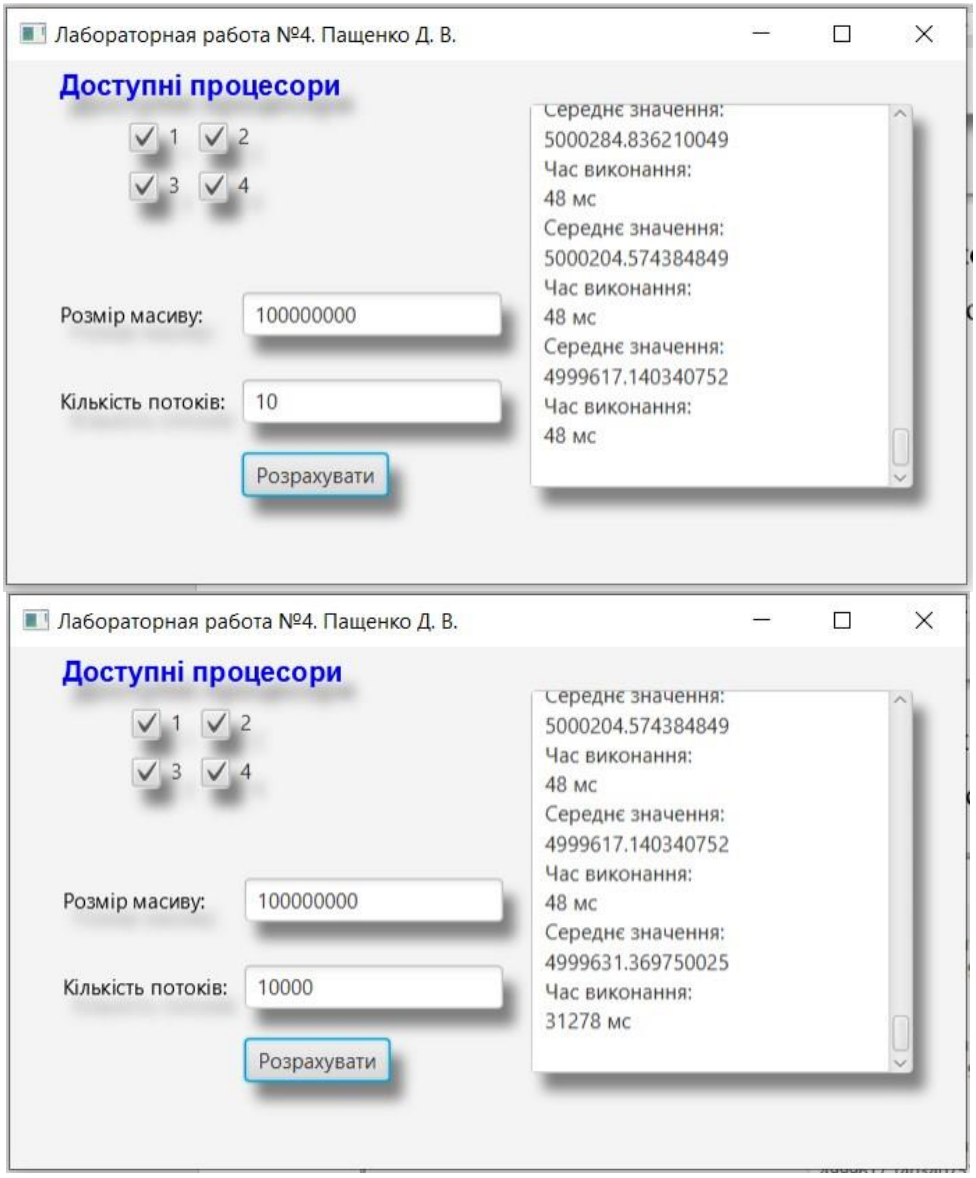
```
        computeButton.setOnAction(e -> calculateAverage(arraySizeField, threadCountField,
resultArea, processorCheckboxes));

        return computeButton;
    }

    //=============================================================
    //Початок
    //=============================================================
    @Override
    public void start(Stage stage) throws Exception {
        //Створення тіні
        Shadow();

        //Створення елементів інтерфейсу
        Text processorsText = createProcessorsText("Доступні процесори");

        CheckBox[] processorCheckboxes = createProcessorsPane();

        TextField arraySizeField = createArraySizePane("Розмір масиву:");

        TextField threadCountField = createThreadCountPane("Кількість потоків:");

        TextArea resultArea = createResultArea();

        Button computeButton = createComputeButton("Розрахувати", arraySizeField,
threadCountField, processorCheckboxes, resultArea);

        //Додавання елементів в групу
        group.getChildren().addAll(processorsText, processorGrid,
                                   arraySizeText, arraySizeField,
                                   threadCountText, threadCountField,
                                   resultArea, computeButton);

        //Застосування тіні до елементів
        group.setEffect(dropShadow);

        // Додавання елементів на панель
        root.getChildren().addAll(group);

        //Створення сцени
        stage.setTitle("Лабораторная работа №4. Пащенко Д. В.");

        //Встановлення сцени та відображення вікна
        stage.setScene(new Scene(root, 550, 300));
        stage.show();
    }

    //=============================================================
```

```
    // Точка входу в програму (запускає метод start)
    //================================================================
    public static void main(String[] args) {
        launch(args);
    }
}
```





Чим більше процесорів тим швидше виконується обробка але ефективність цих процесів падає до поки додавання нових процесів ні на що не впливає. Тому для цієї програми було достатньо двох процесів, чотири можливо буде швидше але вплив буде малим.



Можна побачити, що чим більше потоків тим повільніша програма, напевно, тому що велика кількість потоків це багато виділення пам’яті і таке інше.

В принципі, краще усього використовувати 4 процесори та не багато потоків 8-12 для максимальної швидкодії і мінімальним виратам.

Вказівки до виконання лабораторної роботи

1. Створити додаток, що реалізує паралельні обчислення в багатопроцесорних системах.
2. У додатку реалізувати обчислення над масивом відповідно до варіанта.
3. Зробити аналіз результатів виконання паралельних обчислень для одного і декількох процесорів.
4. Зробити аналіз ефективності використання великої кількості паралельних потоків обчислень.
5. Зробити висновок про оптимальну кількість процесорів в обчислювальній системі з погляду співвідношення ефективності/вартості.
6. Зробити висновок про оптимальну кількість потоків в обчислювальній системі для оптимальної кількості процесорів.

Контрольні питання:

1. Процес організації паралельних обчислень
2. Організація рекурсивних обчислень на основі технології Fork-Join.
3. Ефективність виконання паралельних обчислень для різної кількості процесорів. (Оптимальна кількість).
4. Ефективність виконання паралельних обчислень для різної кількості паралельних потоків. (Оптимальна кількість).

ВАРІАНТИ:

	Максимум	Мінімум	Середнє
0	+		
1		+	
			+
3	+		
4		+	
5			+
6	+		
7		+	
8			+
9	+		

ЛАБОРАТОРНА РОБОТА № 5

Тема: РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ НА БАЗІ ТЕХНОЛОГІЇ КЛІЄНТ-СЕРВЕР

Мета роботи: Одержати навички створення і налагодження додатків для розподілених обчислень

Завдання:

1. Створити проект для Клієнтської і Серверної програми.
2. Реалізувати програмне забезпечення Сервера.
3. Реалізувати програмне забезпечення Клієнта.
4. Здійснити мережну взаємодію Клієнта і Сервера (установити з'єднання).
5. На стороні Сервера забезпечити виконання обчислень за даними, отриманими від Клієнта.
6. Результати обчислень повернути Клієнту.
7. На стороні клієнта забезпечити виведення результатів обчислень.

ТЕОРЕТИЧНІ ВІДОМОСТІ:

Розподілені обчислення являють собою особливий тип програм, побудованих по архітектурі «клієнт-сервер» (мал. 4.1). Особливість їх полягає в тому, що сам додаток знаходиться і виконується на сервері. Клієнт при цьому одержує тільки результати роботи.

Обчислювальна мережа - це сукупність комп'ютерів і терміналів, з'єднаних за допомогою каналів зв'язку в єдину систему, що задовольняє вимогам розподіленої обробки даних, спільного використання загальних інформаційних і обчислювальних ресурсів.

Розподілені обчислення в комп'ютерних мережах засновані на архітектурі “клієнт-сервер”. Терміни “клієнт” і “сервер” позначають ролі, які відіграють різні компоненти в розподіленому середовищі обчислень.

Компоненти “клієнт” і “сервер” повинні працювати на різних машинах. Додаток - клієнт знаходиться на робочій станції користувача, а сервер - на спеціальній виділеній машині.

Сервер - це спеціальна програма, зазвичай запущена на окремому комп'ютері (хості, від слова host(eng.)), що і виконує якесь коло завдань.

Клієнт, у свою чергу - програма, яка запитує сервер виконати ту або іншу дію (завдання) і повернути отримані дані клієнту.

У цілому алгоритм роботи системи клієнт-сервер виглядає в такий спосіб:

1. Сервер підключається до порту і чекає з'єднання із клієнтом;
2. Клієнт створює сокет і намагається з'єднати його з портом на хості;
3. Якщо створення сокета пройшло успішно, то сервер переходить у режим очікування команд від клієнта;
4. Клієнт формує команду і передає її серверу, переходить у режим очікування відповіді;
5. Сервер приймає команду, виконує її і пересилає відповідь клієнту.

Приклад виконання

Створрити проекти Клієнтську і Серверну частини.

Лістинг програми

Клієнт:

```
//==========================================================
// Лабораторна робота № 5
// class: Server
// Copyright (c) 2025 Pashchenko D. V.
//==========================================================
package com.xemacscode.lab5.server;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import javafx.application.Platform;
import javafx.scene.control.TextArea;
import javafx.scene.text.Text;

/**
 *
 * @author Pashchenko D. V.
 */
public class Server implements Runnable {
    //Сокет та елементи інтерфейсу
    final Socket socket;
    final TextArea computeArea;
    final TextArea connectionArea;
    final Text connectionText;
    int clientCount;

    //==========================================================
    //Конструктор серверу
    //==========================================================
    public Server(Socket socket, TextArea computeArea, TextArea connectionArea, Text
connectionText) {
        this.socket = socket;
        this.computeArea = computeArea;
        this.connectionArea = connectionArea;
        this.connectionText = connectionText;
    }

    //==========================================================
    //Збільшити та зменшити лічильник підключень
    //==========================================================
    private void incrementClientCount() {
        clientCount++;
        Platform.runLater(() -> connectionText.setText("Підключених клієнтів: " + clientCount));
    }

    private void decrementClientCount() {
        clientCount--;
        Platform.runLater(() -> connectionText.setText("Підключених клієнтів: " + clientCount));
```

```
    }

    //=================================================================
    //Обчислення даних
    //=================================================================
    private void compute(String message, PrintWriter writer) {
        //Розбиваємо дані від клієнта та розподіляємо по змінним
        String[] messages = message.split(" ");
        String numberOne = messages[0];
        String numberTwo = messages[1];
        String operator = messages[2];

        computeArea.appendText("Операнди:\n" + numberOne + "\n" + numberTwo);

        //Отримуємо результат обчислення
        double result = getResult(Integer.parseInt(numberOne), Integer.parseInt(numberTwo),
operator);

        computeArea.appendText("\nРезультат:\n" + String.valueOf(result) + "\n");

        try {
            writer.println(String.valueOf(result)); // Надсилаємо повідомлення на сервер
        } catch (Exception ex) {
            computeArea.appendText("Помилка надсилання:\n" + ex.getMessage() + "\n");
        }
    }

    //=================================================================
    //Отримання результату
    //=================================================================
    private double getResult(int numberOne, int numberTwo, String operator) {
        //Перевіряємо оператор та обчислюємо операнди
        switch (operator) {
            case "+":
                return numberOne + numberTwo;
            case "-":
                return numberOne - numberTwo;
            case "sin":
                return Math.sin(numberOne);
            case "cos":
                return Math.cos(numberOne);
        }
        return 0;
    }

    //=================================================================
    // Точка запуску серверу
    //=================================================================
    @Override
    public void run() {
```

```
        try (
            PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader reader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        ){
            //Збільшити лічильник підключень
            incrementClientCount();
            Platform.runLater(() -> connectionArea.appendText(">>>Connected: " +
socket.getInetAddress() + "\n"));

            //Отримання даних від клієнта
            String message;
            while ((message = reader.readLine()) != null) {
                String finalMessage = message;

                Platform.runLater(() -> compute(finalMessage, writer));
            }

        } catch (IOException ex) {
            Platform.runLater(() -> connectionArea.appendText("Помилка в обробці клієнта: \n" +
ex.getMessage() + "\n"));
        } finally {
            //Зменшити лічильник підключень
            decrementClientCount();
            try {
                socket.close();
                Platform.runLater(() -> connectionArea.appendText("<<<Disconnected: " +
socket.getInetAddress() + "\n"));
            } catch (IOException ex) {
                Platform.runLater(() -> connectionArea.appendText("Не вдалося закрити сокет: " +
ex.getMessage() + "\n"));
            }
        }
    }
}
```

Сервер:

```
//============================================================
// Лабораторна робота № 5
// class: Laboratorna5Server
// Copyright (c) 2025 Pashchenko D. V.
//============================================================
package com.xemacscode.lab5.server;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import javafx.application.Application;
import javafx.application.Platform;
import javafx.concurrent.Task;
```

```
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.TextArea;
import javafx.scene.effect.DropShadow;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;

/**
 *
 * @author Pashchenko D. V.
 */
public class Laboratorna5Server extends Application {
    Pane root = new Pane();         //Група для вузлів сцени
    Group group = new Group();      //Група для вузлів сцени
    DropShadow dropShadow;          //Тінь
    int PORT = 8000;

    Task server_task;               //Задача серверу

    //=================================================================
    //Тінь
    //=================================================================
    private void Shadow() {
        dropShadow = new DropShadow();      //створити об'єкт тіні
        dropShadow.setRadius(10.0);         //округлення кутів тіні
        dropShadow.setOffsetX(7.0);         //зміщення тіні по X та Y
        dropShadow.setOffsetY(10.0);
        dropShadow.setColor(Color.GRAY);    //колір тіні
    }

    //=================================================================
    //Створити сервер
    //=================================================================
    private void createServerTask(TextArea computeArea, TextArea connectionArea, Text
connectionText) {
        //Поток серверу
        server_task = new Task<>() {
            @Override
            protected Void call() throws Exception {
                try {
                    //Створення сокету
                    ServerSocket serverSocket = new ServerSocket(PORT);
                    //Створення серверу
                    while(true) {
                        Socket socket = serverSocket.accept();
```

```
                    new Thread(new Server(socket, computeArea, connectionArea,
connectionText)).start();
                }
            } catch(IOException ex) {
                Platform.runLater(() -> connectionArea.appendText(ex.getMessage() + "\n"));
                server_task.cancel();
            }
            return null;
        }
    };
    new Thread(server_task).start();
}

//==============================================================
//Створення елементів інтерфейсів
//==============================================================

//Текст заголовку програми
private Text createTopText(String initialText) {
    Text topText = new Text(20, 20, initialText);
    topText.setFont(Font.font("Arial", FontWeight.BOLD, 16));
    topText.setFill(Color.BLUE);

    return topText;
}

//Текстові поля для port, підключення
private Text createPortText(String initialText) {
    Text portText = new Text(260, 23, initialText);

    return portText;
}

private Text createConnectionText(String initialText) {
    Text connectionText = new Text(350, 23, initialText);

    return connectionText;
}

//Створити обчислювальну область та область підключень
private TextArea createComputeArea() {
    TextArea computeArea = new TextArea();
    computeArea.setEditable(false);
    computeArea.setLayoutX(15);
    computeArea.setLayoutY(30);
    computeArea.setPrefWidth(230);
    computeArea.setPrefHeight(150);

    return computeArea;
}
```

```
private TextArea createConnectionArea() {
    TextArea connectionArea = new TextArea();
    connectionArea.setEditable(false);
    connectionArea.setLayoutX(250);
    connectionArea.setLayoutY(30);
    connectionArea.setPrefWidth(230);
    connectionArea.setPrefHeight(150);

    return connectionArea;
}

//=============================================================
//Початок
//=============================================================
@Override
public void start(Stage stage) throws Exception {
    //Створення тіні
    Shadow();

    //Створення елементів інтерфейсу
    Text topText = createTopText("Сервер");

    Text portText = createPortText("Порт: 8000");

    Text connectionText = createConnectionText("Підключень: 0");

    TextArea computeArea = createComputeArea();

    TextArea connectionArea = createConnectionArea();

    //Запуск серверу
    createServerTask(computeArea, connectionArea, connectionText);

    //Додавання елементів в групу
    group.getChildren().addAll(topText, portText, connectionText,
                        computeArea, connectionArea);

    //Застосування тіні до елементів
    group.setEffect(dropShadow);

    // Додавання елементів на панель
    root.getChildren().addAll(group);

    //Створення сцени
    stage.setTitle("Лабораторная работа №5 - Сервер. Пащенко Д. В.");

    //Встановлення сцени та відображення вікна
    stage.setScene(new Scene(root, 495, 200));
    stage.show();
```

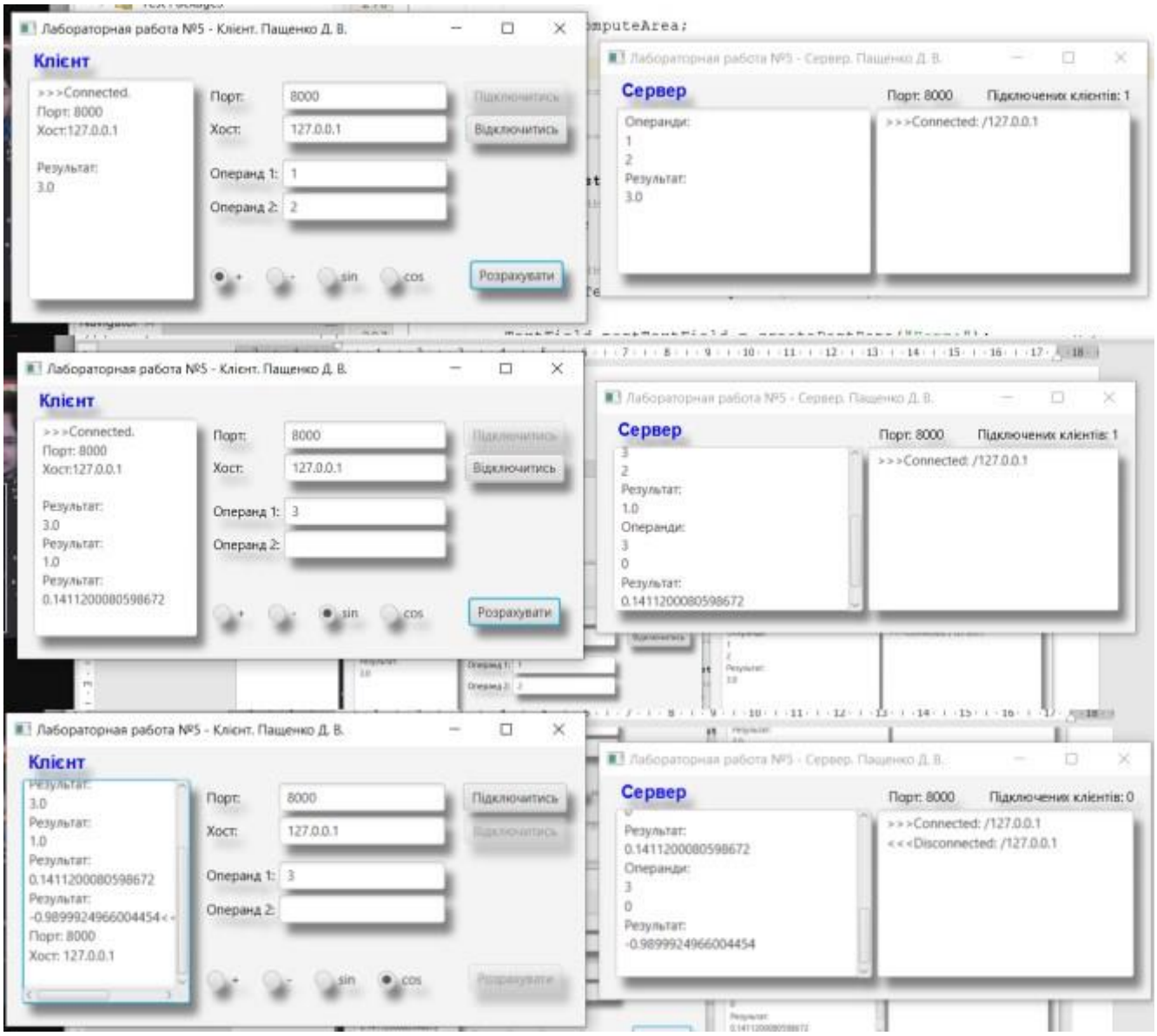
```
    }

    //================================================================
    // Точка входу в програму (запускає метод start)
    //================================================================
    public static void main(String[] args) {
        launch(args);
    }
}
```

Сервер - клас:

```
//================================================================
// Лабораторна робота № 5
// class: Server
// Copyright (c) 2025 Pashchenko D. V.
//================================================================
package com.xemacscode.lab5.server;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import javafx.application.Platform;
import javafx.scene.control.TextArea;
import javafx.scene.text.Text;

/**
 *
 * @author Pashchenko D. V.
 */
public class Server implements Runnable {
    //Сокет та елементи інтерфейсу
    final Socket socket;
    final TextArea computeArea;
    final TextArea connectionArea;
    final Text connectionText;
    int clientCount;

    //================================================================
    //Конструктор серверу
    //================================================================
    public Server(Socket socket, TextArea computeArea, TextArea connectionArea, Text
connectionText) {
        this.socket = socket;
        this.computeArea = computeArea;
        this.connectionArea = connectionArea;
        this.connectionText = connectionText;
    }

    //================================================================
```

```
    //Збільшити та зменшити лічильник підключень
    //=================================================================
    private void incrementClientCount() {
        clientCount++;
        Platform.runLater(() -> connectionText.setText("Підключених клієнтів: " + clientCount));
    }

    private void decrementClientCount() {
        clientCount--;
        Platform.runLater(() -> connectionText.setText("Підключених клієнтів: " + clientCount));
    }

    //=================================================================
    //Обчислення даних
    //=================================================================
    private void compute(String message, PrintWriter writer) {
        //Розбиваємо дані від клієнта та розподіляємо по змінним
        String[] messages = message.split(" ");
        String numberOne = messages[0];
        String numberTwo = messages[1];
        String operator = messages[2];

        computeArea.appendText("Операнди:\n" + numberOne + "\n" + numberTwo);

        //Отримуємо результат обчислення
        double result = getResult(Integer.parseInt(numberOne), Integer.parseInt(numberTwo),
operator);

        computeArea.appendText("\nРезультат:\n" + String.valueOf(result) + "\n");

        try {
            writer.println(String.valueOf(result)); // Надсилаємо повідомлення на сервер
        } catch (Exception ex) {
            computeArea.appendText("Помилка надсилання:\n" + ex.getMessage() + "\n");
        }
    }

    //=================================================================
    //Отримання результату
    //=================================================================
    private double getResult(int numberOne, int numberTwo, String operator) {
        //Перевіряємо оператор та обчислюємо операнди
        switch (operator) {
            case "+":
                return numberOne + numberTwo;
            case "-":
                return numberOne - numberTwo;
            case "sin":
                return Math.sin(numberOne);
            case "cos":
```

```
            return Math.cos(numberOne);
        }
        return 0;
    }

    //=============================================================
    // запуск сервера
    //=============================================================
    @Override
    public void run() {
        try (
            PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader reader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        ){
            //Збільшити лічильник підключень
            incrementClientCount();
            Platform.runLater(() -> connectionArea.appendText(">>>Connected: " +
socket.getInetAddress() + "\n"));

            //Отримання даних від клієнта
            String message;
            while ((message = reader.readLine()) != null) {
                String finalMessage = message;

                Platform.runLater(() -> compute(finalMessage, writer));
            }

        } catch (IOException ex) {
            Platform.runLater(() -> connectionArea.appendText("Помилка в обробці клієнта: \n" +
ex.getMessage() + "\n"));
        } finally {
            //Зменшити лічильник підключень
            decrementClientCount();
            try {
                socket.close();
                Platform.runLater(() -> connectionArea.appendText("<<<Disconnected: " +
socket.getInetAddress() + "\n"));
            } catch (IOException ex) {
                Platform.runLater(() -> connectionArea.appendText("Не вдалося закрити сокет: " +
ex.getMessage() + "\n"));
            }
        }
    }
}
```



Вказівки до виконання лабораторної роботи

1. Створити проект для Клієнтської і Серверної програми.
2. Реалізувати програмне забезпечення Сервера. Поставити у відповідність додатку порт № 8000.
3. Реалізувати програмне забезпечення Клієнта.
4. Здійснити мережна взаємодію Клієнта і Сервера (установити з'єднання).
5. На стороні Сервера забезпечити виконання обчислень за даними, отриманими від Клієнта.
6. Результати обчислень повернути Клієнту.
7. На стороні Клієнта забезпечити виведення результатів обчислень

Контрольні питання:

1. Створення мережного клієнта
2. Створення мережного сервера
3. Поняття номера порту, адреси хоста і сокета.

4. Процес встановлення і розриву з'єднання
5. Процес виконання розподілених обчислень.

ВАРІАНТИ:

	Сума	Різниця	Множення	Розподіл	Синус	Косинус	Тангенс	Котангенс
0	+		+		+		+	
1		+		+		+		+
2	+		+		+		+	
3		+			+	+		+
4								
5		+		+		+	+	
6	+			+		+		+
7		+	+				+	+
8	+	+			+	+		
9			+	+	+			+

ЛАБОРАТОРНА РОБОТА № 6

Тема: РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ. ВЗАЄМОДІЯ ПАРАЛЕЛЬНИХ ПОТОКІВ

Мета роботи: Одержати навички керування потоками розподілених обчислень шляхом передачі повідомлень між потоками виконання.

Завдання:

1. Використовуючи результати лабораторних робіт № 3 і № 4, 5 здійснити взаємодію між потоками Клієнтської і Серверної програми.
2. Реалізувати програмне забезпечення Сервера.
3. Реалізувати програмне забезпечення Клієнта.
4. Здійснити мережне керування потоками розподілених обчислень сторони Сервера з боку Клієнта .
5. Забезпечити безперервне обчислення площ прямокутників у паралельних потоках на стороні Сервера.
6. Забезпечити безперервну передачу результатів обчислень Клієнту
7. На стороні Клієнта забезпечити виведення результатів обчислень.

ТЕОРЕТИЧНІ ВІДОМОСТІ:

Див. лабораторні роботи № 3 і № 4, 5

Вказівки до виконання лабораторної роботи

1. Об'єднати лабораторні роботи № 3 і № 4, 5. Вмонтувати сервер у програму керування потоками.
2. Вилучити елементи керування потоками
3. Елементи керування потоками вмонтувати в клієнтську частину програми лабораторної роботи № 5.
4. Забезпечити передачу повідомлень (команд) від клієнта до сервера.
5. На стороні сервера реалізувати командний процесор для виконання команд клієнта.
6. Здійснити процедуру обчислення площ прямокутників і реалізувати передачу результатів обчислень Клієнту.

Приклад виконання

Створення проектів Клієнтської та Серверної частини.
Лістинг програми
Клієнт:

```
//============================================================
// Лабораторна робота № 5
// class: Laboratorna6Client
// Copyright (c) 2025 Pashchenko D. V.
//============================================================
package com.xemacscode.lab6.client;
```

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.HashMap;
import java.util.Map;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.application.Platform;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.RadioButton;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.control.ToggleGroup;
import javafx.scene.effect.DropShadow;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;

/**
 *
 * @author Pashchenko D. V.
 */
public class Laboratorna6Client extends Application {
    Pane root = new Pane();         //Група для вузлів сцени
    Group group = new Group();      //Група для вузлів сцени
    DropShadow dropShadow;          //Тінь

    Map<Integer, String> threadsMap = new HashMap<>(); // Мапа для ідентифікації потоків

    // Текстові об'єкти для відображення інформації
    Text portText;
    Text hostText;
    Button connectButton;
    Button disconnectButton;
    TextArea connectionArea;
    Text totalText;
    Text rect1Text;
    Text rect2Text;
    Text rect3Text;
    Text rect4Text;

    //Сокет та I/O змінні для сервера
```

```
    Socket socket;
    PrintWriter writer;
    BufferedReader reader;

    //=================================================================
    //Тінь
    //=================================================================
    private void Shadow() {
        dropShadow = new DropShadow();      //створити об'єкт тіні
        dropShadow.setRadius(10.0);         //округлення кутів тіні
        dropShadow.setOffsetX(7.0);         //зміщення тіні по X та Y
        dropShadow.setOffsetY(10.0);
        dropShadow.setColor(Color.GRAY);    //колір тіні
    }

    //=================================================================
    //Відправка повідомлення на сервер
    //=================================================================
    private void sendMessage(String isActive, String thread) {
        try {
            // Форматування повідомлення для сервера
            String message = isActive + " " + thread;
            writer.println(message); // Надсилаємо повідомлення на сервер
        } catch (Exception ex) {
            connectionArea.appendText("Помилка надсилання:\n" + ex.getMessage() + "\n");
        }
    }

    //=================================================================
    //Обробка підключення до сервера
    //=================================================================
    private void handleConnect(TextField portTextField, TextField hostTextField, TextArea
computeArea) {
        //Отримуємо host та port з текстових полів
        String host = hostTextField.getText().trim();
        String port = portTextField.getText().trim();
        int portNumber;

        //Перевірка на число та перевід в int
        try {
            portNumber = Integer.parseInt(port);
        } catch (NumberFormatException e) {
            connectionArea.appendText("Помилка: Порт має бути\nчислом.\n");
            return;
        }

        connectButton.setDisable(true);
        disconnectButton.setDisable(false);

        //Поток підключення до сервера
```

```
        new Thread(() -> {
            try {
                socket = new Socket(host, portNumber);
                writer = new PrintWriter(socket.getOutputStream(), true);
                reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

                Platform.runLater(() -> computeArea.appendText(">>>Connected.\nПорт: " + port +
"\nХост:" + host + "\n"));

                // Очікування повідомлень від сервера
                String message;
                while ((message = reader.readLine()) != null) {
                    String finalMessage = message;
                    Platform.runLater(() -> getData(finalMessage));
                }


            } catch (IOException e) {
                if(!socket.isClosed()) {
                    Platform.runLater(() -> connectionArea.appendText("Помилка підключення: \n" +
e.getMessage() + "\n"));
                }
            }
        }).start();
    }

    //=============================================================
    //Отримати дані від сервера
    //=============================================================
    private void getData(String message) {
        //Розбиває дані на список
        String[] messages = message.split(" ");

        //Перевіряємо для кого дані та оновлюємо їх
        switch(messages[0]) {
            case "total":
                totalText.setText("Сумарна площа пікселів\n\n\t\t\t" + messages[1]);
                break;
            case "Thread1":
                rect1Text.setText(messages[1]);
                break;
            case "Thread2":
                rect2Text.setText(messages[1]);
                break;
            case "Thread3":
                rect3Text.setText(messages[1]);
                break;
            case "Thread4":
                rect4Text.setText(messages[1]);
                break;
```

```
        }
    }

    //=============================================================
    //Обробка відключення від сервера
    //=============================================================
    private void handleDisconnect(TextArea computeArea, TextField portTextField, TextField
hostTextField) {
        connectButton.setDisable(false);
        disconnectButton.setDisable(true);

        //Закриваємо сокет та потоки I/O
        try {
            if (socket != null && !socket.isClosed()) {
                socket.close();
            }
            if (writer != null) {
                writer.close();
            }
            if (reader != null) {
                reader.close();
            }
            Platform.runLater(() -> computeArea.appendText("<<<Disconnected.\nПорт: " +
portTextField.getText() + "\nХост: " + hostTextField.getText() + "\n"));
            totalText.setText("Сумарна площа пікселів\n\n\t\t\t0");
        } catch (IOException e) {
            Platform.runLater(() -> computeArea.appendText("Помилка відключення: " +
e.getMessage() + "\n"));
        }
    }

    //=============================================================
    //Створення елементів інтерфейсів
    //=============================================================

    //Текст заголовку програми
    private Text createTopText(String initialText) {
        Text topText = new Text(20, 20, initialText);
        topText.setFont(Font.font("Arial", FontWeight.BOLD, 16));
        topText.setFill(Color.BLUE);

        return topText;
    }

    //Текстові поля для port, host
    private TextField createPortPane(String initialText) {
        portText = new Text(210, 50, initialText);
        TextField portTextField = new TextField();
        portTextField.setLayoutX(245);
        portTextField.setLayoutY(33);
```

```
    return portTextField;
}

private TextField createHostPane(String initialText) {
    hostText = new Text(210, 80, initialText);
    TextField hostTextField = new TextField();
    hostTextField.setLayoutX(245);
    hostTextField.setLayoutY(63);

    return hostTextField;
}

//Створення кнопок
private Button createConnectButton(String initialText) {
    connectButton = new Button(initialText);
    connectButton.setLayoutX(430);
    connectButton.setLayoutY(33);

    return connectButton;
}

private Button createDisconnectButton(String initialText) {
    disconnectButton = new Button(initialText);
    disconnectButton.setLayoutX(430);
    disconnectButton.setLayoutY(63);
    disconnectButton.setDisable(true);

    return disconnectButton;
}

//Текстова область для повідомлень
private void createClientArea() {
    connectionArea = new TextArea();
    connectionArea.setEditable(false);
    connectionArea.setLayoutX(15);
    connectionArea.setLayoutY(30);
    connectionArea.setPrefWidth(150);
    connectionArea.setPrefHeight(100);
}

//Створення горизонтальної лінії
private Line createLine() {
    Line line = new Line();
    line.setStartX(10);
    line.setStartY(145);
    line.setEndX(590);
    line.setEndY(145);

    return line;
```

```
    }

    //Заповнення масиву назвами потоків
    private void fillThreadsMap() {
        for(int index = 0; index < 4; index++) {
            threadsMap.put(index, "Thread" + String.valueOf(index + 1));
        }
    }

    //Текст текстів
    private void createThreadsTexts() {
        Text threadText;
        for(int index = 0; index < 4; index++) {
            threadText = new Text(240 + (90*index), 170, threadsMap.get(index));
            group.getChildren().addAll(threadText);
        }
    }

    private Text createText(int positionX, String initialText) {
        Text text = new Text(245 + positionX, 245, initialText); //створення тексту
        text.setFont(Font.font("Arial", FontWeight.BOLD, 16));  //Розмір тексту та шрифт, жирність
тексту
        text.setFill(Color.RED);  //Колір тексту
        return text;
    }

    private Text createTotalText(String initialText) {
        Text totalText = new Text(30, 175, initialText);
        totalText.setFont(Font.font("Arial", FontWeight.BOLD, 14));
        totalText.setFill(Color.RED);
        return totalText;
    }

    private Text createBottomText(String initialText) {
        Text topText = new Text(20, 245 , initialText);
        topText.setFont(Font.font("Arial", FontWeight.BOLD, 12));
        return topText;
    }

    //=============================================================
    //Створення RadioButtons
    //=============================================================
    private void createRadioButtons(int positionX, String thread) {
        ToggleGroup toggleGroup = new ToggleGroup();
        //Група перемикачів, які вмикають задачі/потоки
        RadioButton enableButton = new RadioButton("Увімкнути");
        enableButton.setToggleGroup(toggleGroup); //Додавання до групи
        enableButton.setLayoutX(245 + positionX); //Позиція по X та Y
        enableButton.setLayoutY(180);
        enableButton.setOnAction(e -> sendMessage("true", thread));
```

```
        //Група перемикачів, які вимикають задачі/потоки
        RadioButton disableButton = new RadioButton("Вимкнути");
        disableButton.setToggleGroup(toggleGroup);
        disableButton.setSelected(true); //Увімкнутий за замовчуванням
        disableButton.setLayoutX(245 + positionX);
        disableButton.setLayoutY(200);
        disableButton.setOnAction(e -> sendMessage("false", thread));

        group.getChildren().addAll(enableButton, disableButton); //Додавання перемикачів на
панель
    }

    //=============================================================
    //Початок
    //=============================================================
    @Override
    public void start(Stage stage) throws Exception {
        //Створення тіні
        Shadow();

        //Створення елементів інтерфейсу
        Text topText = createTopText("Клієнт");

        TextField portTextField = createPortPane("Порт:");

        TextField hostTextField = createHostPane("Хост:");

        connectButton = createConnectButton("Підключитись");

        disconnectButton = createDisconnectButton("Відключитись");

        createClientArea();

        Line line = createLine();

        fillThreadsMap();

        createThreadsTexts();

        //Створення елементів керування
        createRadioButtons(0, threadsMap.get(0));
        createRadioButtons(90, threadsMap.get(1));
        createRadioButtons(180, threadsMap.get(2));
        createRadioButtons(270, threadsMap.get(3));

        //Створеннння тексту площи прямокутників
        rect1Text = createText(0, "0");
        rect2Text = createText(90, "0");
        rect3Text = createText(180, "0");
```

```
        rect4Text = createText(270, "0");

        //Текст сумарної площі
        totalText = createTotalText("Сумарна площа пікселів\n\n\t\t\t0");

        Text BottomText = createBottomText("Площі прямокутників в пікселях");

        //Встановлення обробники подій на кнопки
        connectButton.setOnAction(event -> handleConnect(portTextField, hostTextField,
connectionArea));

        disconnectButton.setOnAction(event -> handleDisconnect(connectionArea, portTextField,
hostTextField));

        //Додавання елементів в групу
        group.getChildren().addAll(topText, portTextField, portText, hostTextField, hostText, line,
                        rect1Text, rect2Text, rect3Text, rect4Text, totalText, BottomText,
                        connectButton, disconnectButton, connectionArea);

        //Застосування тіні до елементів
        group.setEffect(dropShadow);

        // Додавання елементів на панель
        root.getChildren().addAll(group);

        //Створення сцени
        stage.setTitle("Лабораторная работа №6 - Клієнт. Пащенко Д. В.");

        //Встановлення сцени та відображення вікна
        stage.setScene(new Scene(root, 600, 280));
        stage.show();
    }

    //=============================================================
    // Точка входу в програму (запускає метод start)
    //=============================================================
    public static void main(String[] args) {
        launch(args);
    }
}
```

Сервер:

```
//=============================================================
// Лабораторна робота № 5
// class: Laboratorna6Server
// Copyright (c) 2025 Pashchenko D. V.
//=============================================================
package com.xemacscode.lab6.server;

import java.io.IOException;
```

```
import java.net.ServerSocket;
import java.net.Socket;
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.application.Platform;
import javafx.concurrent.Task;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.TextArea;
import javafx.scene.effect.DropShadow;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;

/**
 *
 * @author Pashchenko D. V.
 */
public class Laboratorna6Server extends Application {
    Pane root = new Pane();         //Група для вузлів сцени
    Group group = new Group();      //Група для вузлів сцени
    DropShadow dropShadow;          //Тінь
    int PORT = 8000;

    Task server_task;               //Задача серверу

    //=============================================================
    //Тінь
    //=============================================================
    private void Shadow() {
        dropShadow = new DropShadow();      //створити об'єкт тіні
        dropShadow.setRadius(10.0);         //округлення кутів тіні
        dropShadow.setOffsetX(7.0);         //зміщення тіні по X та Y
        dropShadow.setOffsetY(10.0);
        dropShadow.setColor(Color.GRAY);    //колір тіні
    }

    //=============================================================
    //Створити сервер
    //=============================================================
    private void createServerTask(TextArea connectionArea, Text connectionText, Rectangle[]
rectangles, Text totalText) {
        //Поток серверу
        server_task = new Task<>() {
            @Override
            protected Void call() throws Exception {
```

```
            try {
                //Створення сокету
                ServerSocket serverSocket = new ServerSocket(PORT);
                //Створення серверу
                while(true) {
                    Socket socket = serverSocket.accept();
                    new Thread(new Server(socket, connectionArea, connectionText, rectangles,
totalText)).start();
                }
            } catch(IOException ex) {
                Platform.runLater(() -> connectionArea.appendText(ex.getMessage() + "\n"));
                server_task.cancel();
            }
            return null;
        }
    };
    new Thread(server_task).start();
}

//=============================================================
//Створення елементів інтерфейсів
//=============================================================

//Текст заголовку програми
private Text createLabel(Rectangle rectangle, String initialText) {
    Text textLabel = new Text(rectangle.getX() - 10, rectangle.getY() - 5, initialText);
    textLabel.setFont(Font.font("Arial", FontWeight.BOLD, 16));
    textLabel.setFill(Color.GRAY);
    return textLabel;
}

//Далі так само створення текстів
private Text createTotalText(String initialText) {
    Text totalText = new Text(133, 300, initialText);
    totalText.setFont(Font.font("Arial", FontWeight.BOLD, 16));
    totalText.setFill(Color.RED);
    return totalText;
}

private Text createTotalTextThreadText(String initialText) {
    Text threadText = new Text(57, 300, initialText);
    threadText.setFont(Font.font("Arial", FontWeight.BOLD, 16));
    threadText.setFill(Color.GRAY);
    return threadText;
}

private Text createTopText(String initialText) {
    Text topText = new Text(95, 20 , initialText);
    topText.setFont(Font.font("Arial", FontWeight.BOLD, 16));
    topText.setFill(Color.RED);
```

```
    return topText;
}

//============================================================
//Створення прямокутників
//============================================================
private Rectangle createRectangle(double x, double y, double width, double height, Color color) {
    Rectangle rectangle = new Rectangle(x, y, width, height);
    rectangle.setFill(color);
    return rectangle;
}

//Текстові поля для port, підключення
private Text createPortText(String initialText) {
    Text portText = new Text(433, 23, initialText);

    return portText;
}

private Text createConnectionText(String initialText) {
    Text connectionText = new Text(520, 23, initialText);

    return connectionText;
}

//Створити область підключень
private TextArea createConnectionArea() {
    TextArea connectionArea = new TextArea();
    connectionArea.setEditable(false);
    connectionArea.setLayoutX(420);
    connectionArea.setLayoutY(30);
    connectionArea.setPrefWidth(200);
    connectionArea.setPrefHeight(150);

    return connectionArea;
}

//============================================================
//Початок
//============================================================
@Override
public void start(Stage stage) throws Exception {
    //Створення тіні
    Shadow();

    //Створення елементів інтерфейсу
    Text portText = createPortText("Порт: 8000");

    Text connectionText = createConnectionText("Підключень: 0");
```

```
    TextArea connectionArea = createConnectionArea();

    // Створення прямокутників
    Rectangle rect1 = createRectangle(50, 70, 30, 3, Color.PINK);
    Rectangle rect2 = createRectangle(150, 70, 30, 3, Color.PINK);
    Rectangle rect3 = createRectangle(250, 70, 30, 3, Color.PINK);
    Rectangle rect4 = createRectangle(350, 70, 30, 3, Color.PINK);

    //Додавання прямокутників в масив
    Rectangle[] rectangles = { rect1, rect2, rect3, rect4 };

    //Створенян текстів
    Text topText = createTopText("Площі прямокутників в пікселях");

    //Назви потоків
    Text rect1Label = createLabel(rect1, "Поток 1");
    Text rect2Label = createLabel(rect2, "Поток 2");
    Text rect3Label = createLabel(rect3, "Поток 3");
    Text rect4Label = createLabel(rect4, "Поток 4");

    //Сумарна площа
    Text totalText = createTotalText("Сумарна площа: 0 пікселів");
    Text threadText = createTotalTextThreadText("Поток 5");

    //Створення потоку серверу
    createServerTask(connectionArea, connectionText, rectangles, totalText);

    //Додавання елементів в групу
    group.getChildren().addAll(topText, totalText, threadText,
                               rect1Label,   rect2Label,
                               rect3Label,   rect4Label,
                               rect1, rect2, rect3, rect4,
                               portText, connectionText, connectionArea);

    //Застосування тіні до елементів
    group.setEffect(dropShadow);

    // Додавання елементів на панель
    root.getChildren().addAll(group);

    //Створення сцени
    stage.setTitle("Лабораторная работа №6 - Сервер. Пащенко Д. В.");

    //Встановлення сцени та відображення вікна
    stage.setScene(new Scene(root, 650, 320));
    stage.show();
}

//============================================================
// Точка входу в програму (запускає метод start)
```

```
    //=============================================================
    public static void main(String[] args) {
        launch(args);
    }
}
```

Сервер - клас:

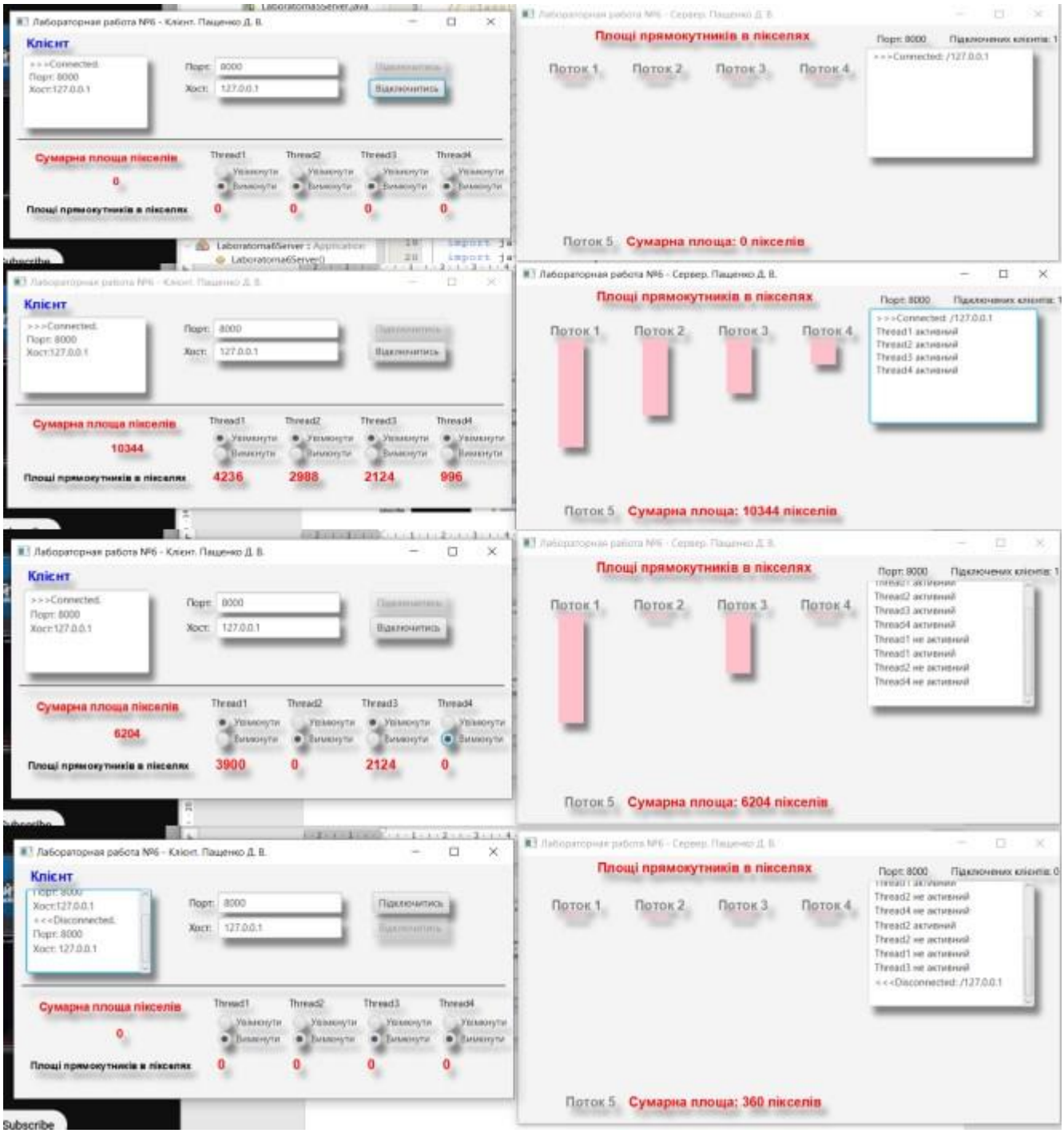
```
//=============================================================
// Лабораторна робота № 6
// class: Server
// Copyright (c) 2025 Pashchenko D. V.
//=============================================================
package com.xemacscode.lab6.server;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import javafx.application.Platform;
import javafx.concurrent.Task;
import javafx.scene.control.TextArea;
import javafx.scene.input.MouseEvent;
import javafx.scene.shape.Rectangle;
import javafx.scene.text.Text;

/**
 *
 * @author Pashchenko D. V.
 */
public class Server implements Runnable {
    final Socket socket;
    final TextArea connectionArea;
    final Text connectionText;
    final Rectangle[] rectangles;
    final Text totalText;
    int clientCount;
    int MAX_HEIGHT = 190; //максимальне значення довжини

    boolean[] taskActive = {false, false, false, false};    //Прапори активності задач/потоків

    //=============================================================
    //Конструктор серверу
    //=============================================================
    public Server(Socket socket, TextArea connectionArea, Text connectionText, Rectangle[]
rectangles, Text totalText) {
        this.socket = socket;
        this.connectionArea = connectionArea;
        this.connectionText = connectionText;
        this.rectangles = rectangles;
        this.totalText = totalText;
```

```
    }

    //=================================================================
    //Збільшити та зменшити лічильник підключень
    //=================================================================
    private void incrementClientCount() {
        clientCount++;
        Platform.runLater(() -> connectionText.setText("Підключених клієнтів: " + clientCount));
    }

    private void decrementClientCount() {
        clientCount--;
        Platform.runLater(() -> connectionText.setText("Підключених клієнтів: " + clientCount));
    }

    //=================================================================
    //Зміна розміру прямокутників (тільки довжини)
    //=================================================================
    private void enableResize(Rectangle rectangle, int index) {
        rectangle.setOnMouseDragged((MouseEvent event) -> {     //Зміна розміру при затисканні
клавіши миші
            if(!taskActive[index]) {                        //Перевірка чи увікнена задача
                return;
            }

            double newHeight = event.getY() - rectangle.getY(); //Обчислення розміру
            if (newHeight > 1 && newHeight <= MAX_HEIGHT) {     //Перевіряємо розмір
                rectangle.setHeight(newHeight);                 //Задаємо новий розмір
            }
        });
    }


//=====================================================================
    //Задачи та потоки(1, 2, 3, 4) для обчислення площи кожного прямокутника

//=====================================================================
    private void startCalculationTask(Rectangle rectangle, int index, PrintWriter writer, String
threadName) {
        Task<Void> task = new Task<>() {
            @Override
            protected Void call() throws Exception {
                while (true) {
                    if (taskActive[index]) {  //перевірка, якщо задача активна
                        Platform.runLater(() -> {
                            enableResize(rectangle, index); //застосування можливості зміни розміру
прямокутників
                            int area = (int) (rectangle.getWidth() * rectangle.getHeight()); //Обчислення
площу прямокутника
```

```
                    try {
                        writer.println(threadName + " " + area);  // Надсилаємо повідомлення на
сервер
                    } catch (Exception ex) {
                        connectionArea.appendText("Помилка надсилання:\n" + ex.getMessage() +
"\n");
                    }
                });
            }
            else {
                rectangle.setHeight(3); //Встановлення мінімальної висоти прямокутника, якщо
задача вимкнена
                try {
                    writer.println(threadName + " " + 0); // Надсилаємо повідомлення на сервер
                } catch (Exception ex) {
                    connectionArea.appendText("Помилка надсилання:\n" + ex.getMessage() + "\n");
                }
            }
            Thread.sleep(100); // Затримка між обчисленнями
        }
    }
};
Thread thread = new Thread(task); //Створення потоку
thread.setDaemon(true);  //Поток працює на фоні основного потоку програми
thread.start();     //Запуск потоку
}

//=============================================================
//Задача та поток(5) для обчислення площи прямокутників
//=============================================================
private Task startTotalTask(PrintWriter writer) {
    Task<Void> task = new Task<>() {
        @Override
        protected Void call() throws Exception {
            while (true) {
                Platform.runLater(() -> {
                    int totalArea = 0;
                    for (Rectangle rect : rectangles) {
                        totalArea += (int)(rect.getWidth() * rect.getHeight());//Обчислення загальної
площи
                    }
                    totalText.setText("Сумарна площа: " + totalArea + " пікселів");  //Вивід
результату
                    try {
                        writer.println("total" + " " + totalArea); // Надсилаємо повідомлення на сервер
                    } catch (Exception ex) {
                        connectionArea.appendText("Помилка надсилання:\n" + ex.getMessage() + "\n");
                    }
                });
                Thread.sleep(100); // Затримка між обчисленнями
```

```
            }
        }
    };
    Thread thread = new Thread(task); //Створення потоку
    thread.setDaemon(true);          //Поток працює на фоні основного потоку програми
    thread.start();                  //Запуск потоку

    return task;
}

//=================================================================
//Перевірити активності потоку
//=================================================================
private boolean checkThreadActivite(String bool, String thread) {
    boolean boolFromString = Boolean.valueOf(bool);

    //Якщо активний
    if(boolFromString) {
        connectionArea.appendText(thread + " активний\n");
        return boolFromString;
    }
    else { //Якщо неактивний
        connectionArea.appendText(thread + " не активний\n");
        return boolFromString;
    }
}

//=================================================================
//Обробка даних
//=================================================================
private void recieveData(String message, PrintWriter writer) {
    //Розбиває дані на список
    String[] messageData = message.split(" ");

    //Перевіряємо для кого дані та оновлюємо їх
    switch (messageData[1]) {
        case "Thread1":
            taskActive[0] = checkThreadActivite(messageData[0], messageData[1]);
            startCalculationTask(rectangles[0], 0, writer, messageData[1]);
            break;
        case "Thread2":
            taskActive[1] = checkThreadActivite(messageData[0], messageData[1]);
            startCalculationTask(rectangles[1], 1, writer, messageData[1]);
            break;
        case "Thread3":
            taskActive[2] = checkThreadActivite(messageData[0], messageData[1]);
            startCalculationTask(rectangles[2], 2, writer, messageData[1]);
            break;
        case "Thread4":
            taskActive[3] = checkThreadActivite(messageData[0], messageData[1]);
```

```
                startCalculationTask(rectangles[3], 3, writer, messageData[1]);
                break;
            }
        }

        //=============================================================
        // Точка запуску серверу
        //=============================================================
        @Override
        public void run() {
            try (
                PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);
                BufferedReader reader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            ){
                //Збільшити лічильник підключень
                incrementClientCount();
                Platform.runLater(() -> connectionArea.appendText(">>>Connected: " +
socket.getInetAddress() + "\n"));

                //Отримання даних від клієнта
                String message;
                while ((message = reader.readLine()) != null) {
                    String finalMessage = message;
                    Platform.runLater(() -> {
                        recieveData(finalMessage, writer);
                        startTotalTask(writer);
                    });
                }

            } catch (IOException ex) {
                Platform.runLater(() -> connectionArea.appendText("Помилка в обробці клієнта: \n" +
ex.getMessage() + "\n"));
            } finally {
                //Зменшити лічильник підключень
                decrementClientCount();
                try {
                    socket.close();
                    Platform.runLater(() -> connectionArea.appendText("<<<Disconnected: " +
socket.getInetAddress() + "\n"));
                } catch (IOException ex) {
                    Platform.runLater(() -> connectionArea.appendText("Не вдалося закрити сокет: " +
ex.getMessage() + "\n"));
                }
            }
        }
}
```



Контрольні питання:

1. Процес взаємодії потоків виконання клієнта і сервера
2. Обмін повідомленнями і командами між потоками
3. Робота командного процесора на стороні сервера.
4. Клієнтські команди керування потоками обчислень на стороні сервера.

ВАРІАНТИ:

	Довжина, пікс	Ширина, пікс	Колір
0	110	20	червоний
1	120	30	сірий
2	130	40	синій
3	140	50	зелений
4	150	60	кавовий
5	160	60	чорний
6	170	50	фіолетовий
7	180	40	блакитний
8	190	30	рожевий
9	200	20	темно-сірий

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Кузьменко Б.В., Чайковська О.А. Технологія розподілених систем та паралельних обчислень. (конспект лекцій, частина 1. Розподілені об’єктні системи, паралельні обчислювальні системи та паралельні обчислення, паралельне програмування на основі MPI) Навчальний посібник. – К.: Видавничий центр КНУКІМ, 2011 – 126 с.

2. Малашонок Г. І., Сідько А. А. Паралельні обчислення на розподіленій пам'яті: OpenMPI, Java, Math Partner : підручник. – Київ : НаУКМА, 2020. – 266 с. 3. Дорошенко А.Ю. Паралельні обчислювальні системи. Методичний посібник і конспект лекцій. – Київ: Видавничий дім «КМ Академія», 2013.–46 с.

4. Рольщиков В.Б. Технології розподілених систем та паралельних обчислень. Конспект лекцій. Одеса: ОДЕКУ 2016.155с

5. Паралельні та розподілені обчислення: навчальний посібник для вищих закладів освіти / К.Т. Кузьма, О.В. Мельник. – Миколаїв: ФОП Швець В.М., 2020. – 172 с.

6. Ashwin Pajankar. Raspberry Pi Supercomputing and Scientific Programming. – Nashik, Maharashtra, India, 2017. – 170 p.

7. Семеренко, В. П. Технології паралельних обчислень : навчальний посібник / Семеренко В. П. – Вінниця : ВНТУ, 2018. – 104 с.

8. Gropp, William. Using MPI : portable parallel programming with the Message-Passing Interface / William Gropp, Ewing Lusk, and Anthony Skjellum. Third edition. – Massachusetts Institute of Technology, 2014. – 330 с.

9. Качко О.Г. Паралельне програмування. – Харків. нац. ун-т радіоелектроніки. – Харків : ХНУРЕ, 2016. – 403 с.

10. Минайленко Р.М Паралельні та розподілені обчислення: навчальний посібник / М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. - Кропивницький : ЦНТУ, 2021. - 153 с. http://dspace.kntu.kr.ua/jspui/handle/123456789/10630

11.Дистанційна освіта ЦНТУ. – URL: https://moodle.kntu.kr.ua/?lang=en

12.Технології паралельного програмування URL: http://www.parallel.ru/tech

13. Сайт Української команди розподілених обчислень.– Режим доступу: http://distributed.org.ua/.

14. Стандарти MPI.– Режим доступу: http://www.mpiforum.org