МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КІРОВОГРАДСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

МЕХАНІКО-ТЕХНОЛОГІЧНИЙ ФАКУЛЬТЕТ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Паралельні та розподілені обчислення

Методичні вказівки до виконання лабораторних робіт

з елементами кредитно - модульної
системи організації навчального процесу

для студентів денної та заочної форми навчання
за напрямком підготовки 6.050102 «Комп'ютерна інженерія»

Укладачі:
доцент Смірнова Н.В.
доцент Смірнов В.В.

Кіровоград 2015

Паралельні та розподілені обчислення: Методичні вказівки до виконання лабораторних робіт для студентів денної та заочної форми навчання за напрямком підготовки 6.050102 «Комп'ютерна інженерія» / Уклад .: /
Смірнова Н.В. Смірнов В.В., - Кіровоград: КНТУ, 2015 – 52 с.

Затверджено на засіданні кафедри ПЗ: 2 липня 2015 р. протокол № 21.

Укладачі:
Смірнова Наталя Володимирівна, к.т.н., доцент кафедри ПЗ.
Смірнов Володимир Вікторович, к.т.н., доцент кафедри ПЗ,

Для студентів денної та заочної форми навчання, що вивчають навчальну дисципліну "Паралельні та розподілені обчислення" за напрямком підготовки 6.050102 «Комп'ютерна інженерія».

У стислій формі викладені основні принципи побудови об'єктно-орієнтованих подійно-керованих інтерактивних додатків паралельних і розподілених обчислень мовою програмування Java у середовищі розробки NetBeans IDE.

Представлені практичні розв'язки навчальних завдань для кращого освоєння досліджуваного матеріалу, представлені варіанти навчальних завдань для самостійного придбання практичних навичок.

ЗМІСТ

1. Опис навчальної дисципліни "Паралельні та розподілені обчислення"

Основна мета курсу полягає в придбанні зроблених знань і навичок роботи в середовищі розробки NetBeans IDE і мови програмування Java із застосуванням сучасних технологій і інструментальних засобів для побудови об'єктно-орієнтованих подійно-керованих додатків паралельних і розподілених обчислень.

У результаті проведення лекцій студенти повинні одержати теоретичні знання і методику ефективної роботи із сучасними методами створення об'єктно-орієнтованих керованих подіями додатків паралельних і розподілених обчислень.

Завдання вивчення дисципліни

- вивчення теоретичних основ проектування додатків паралельних і розподілених обчислень;
- вивчення теоретичних основ програмування додатків паралельних і розподілених обчислень;
- вивчення теоретичних основ методів створення додатків паралельних обчислень;
- вивчення теоретичних основ методів створення додатків розподілених обчислень;
- рішення завдань керування процесом паралельних і розподілених обчислень на платформі JavaFx;
- придбання практичних навичок в області програмування додатків паралельних і розподілених обчислень на основі технології Java.

Предметом навчальної дисципліни є створення об'єктно-орієнтованих подійно-керованих додатків паралельних і розподілених обчислень у середовищі програмування NetBeans IDE на платформі JavaFx.

У результаті вивчення навчальної дисципліни студент повинен

знати:

- Володіти методами і засобами побудови сучасних паралельних КС
- Аналізувати особливості архітектури паралельних КС
- Володіти способами організації пам’яті і обміну даними у паралельних КС с різною архітектурою
- Володіти методами і засобами програмного забезпечення для паралельних і розподілених комп’ютерних системах

вміти:

- Здійснювати побудову паралельного алгоритму і виконувати його аналіз
- Створювати програми з застосуванням процесів (потоків). Вміти керувати процесами
- Реалізувати взаємодію процесів
- Виконувати моделювання паралельних обчислень
- Створювати та налагоджувати паралельну або розподілену програму

Шкала оцінювання

За шкалою ECTS	**За національною шкалою**	**За шкалою навчального закладу**
A	відмінно	90-100
B-C	добре	75-89
D-E	задовільно	60-74
F-X	незадовільно з можливістю повторної здачі	35-59
F	незадовільно з обов'язковим повторним курсом	1-34

2. Лабораторні роботи з дисципліни "Паралельні та розподілені обчислення"

№ заняття	Теми лабораторних занять	Кількість годин
1.	**№1** Освоєння середовища розробки графічних додатків NetBeans IDE	5
2.	**№2** Створення додатка для паралельних обчислень	5
3.	**№3** Керування процесом паралельних обчислень	6
4.	**№4** Паралельні обчислення в багатопроцесорних системах. Технологія Fork-Join	6
5.	**№5** Розподілені обчислення на базі технології Клієнт-Сервер	6
6.	**№6** Розподілені обчислення. Взаємодія паралельних потоків	6
	усього годин	**34**

3. Змістовні модулі

Навчальне навантаження складається з 2 - змістовних модулів, які містять у собі лекції, лабораторні роботи, самостійну роботу і контроль знань. Система оцінок успішності в балах включає тестовий поточний контроль при виконанні лабораторних робіт, модульний контроль і оцінку самостійної роботи.

Перший модуль.

Лабораторні роботи:

№1 Освоєння середовища розробки графічних додатків NetBeans IDE

№2 Створення додатка для паралельних обчислень

№3 Керування процесом паралельних обчислень

Другий модуль.

Лабораторні роботи:

№4 Паралельні обчислення в багатопроцесорних системах. Технологія Fork-Join

№5 Розподілені обчислення на базі технології Клієнт-Сервер

№6 Розподілені обчислення. Взаємодія паралельних потоків

4. Оцінка успішності в балах при повному виконанні умов і графіку навчального процесу

№ модуля	Лабораторні роботи			Реферат	Відвідування л.р.	Максимальна сума балів
	Теми лабораторних робіт	Тестовий контроль	Виконання л.р.			
1.	**№1** Освоєння середовища розробки графічних додатків NetBeans IDE **№2** Створення додатка для паралельних обчислень **№3** Керування процесом паралельних обчислень	15	21		9	45
2.	**№4** Паралельні обчислення в багатопроцесорних системах. Технологія Fork-Join **№5** Розподілені обчислення на базі технології Клієнт-Сервер **№6** Розподілені обчислення. Взаємодія паралельних потоків	15	21	10	9	55
		30	42	10	18	100

Лабораторна робота №1

Тема: Освоєння середовища розробки графічних додатків NetBeans IDE

Ціль роботи:

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

Завдання:

– Створити простий проект JavaFx в NetBeans IDE.

– 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();
```

```
//=== Крапка входу 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#.

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

В Java увесь код повинен розміщатися усередині 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 наступних строк:

```
// 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.

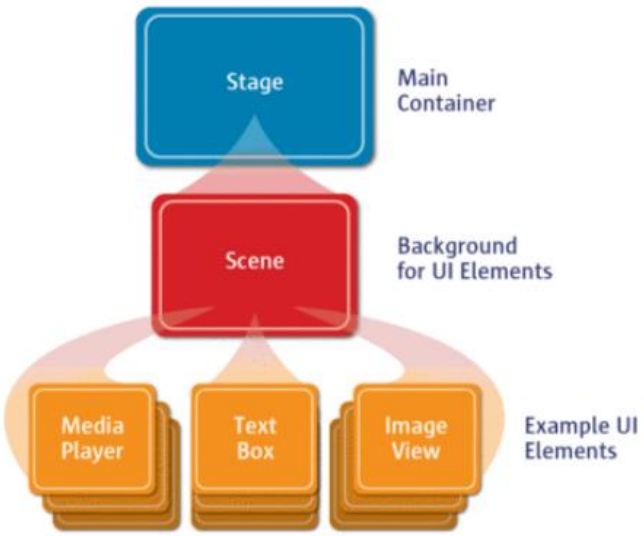
Крапкою входу в JavaFx-додатока служить Java - клас, що розширює абстрактний клас `javafx.application.Application` і який містить метод `main()`:

```
public class JavaFXApp extends Application
{
     public static void main(String[]
args) {
          launch(args);
     }
     @Override
     public void start(Stage primaryStage)
{
          //Установка параметров сцены
          . . .
```

```
            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 установки і відображення сцени JavaFx-додатка.

Перед установкою 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.

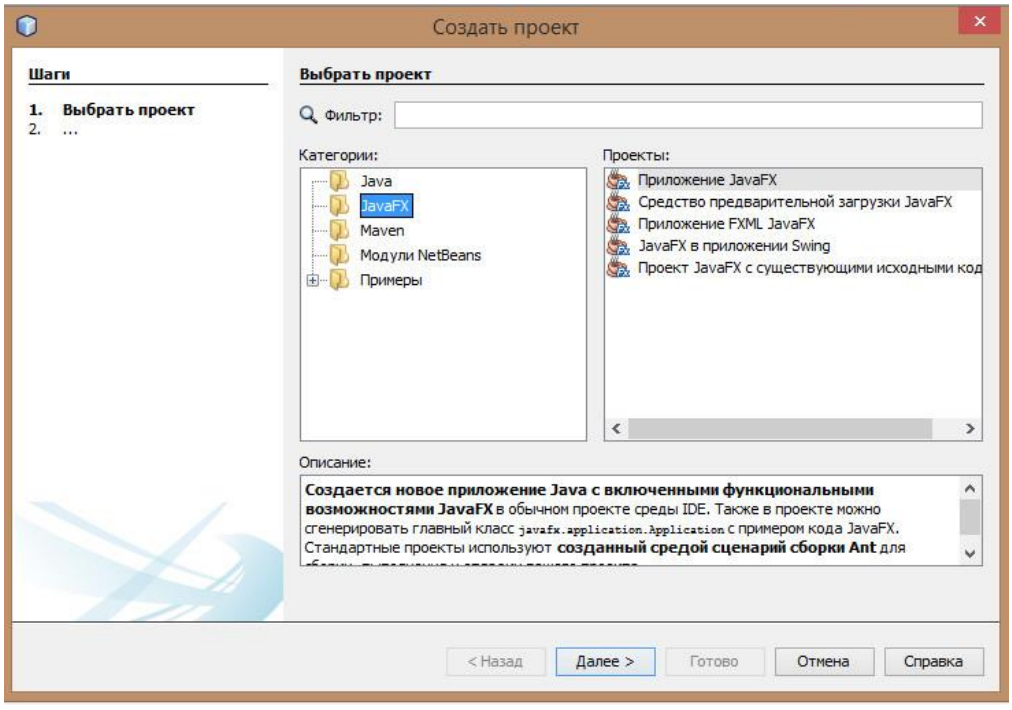
Порядок роботи:

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.

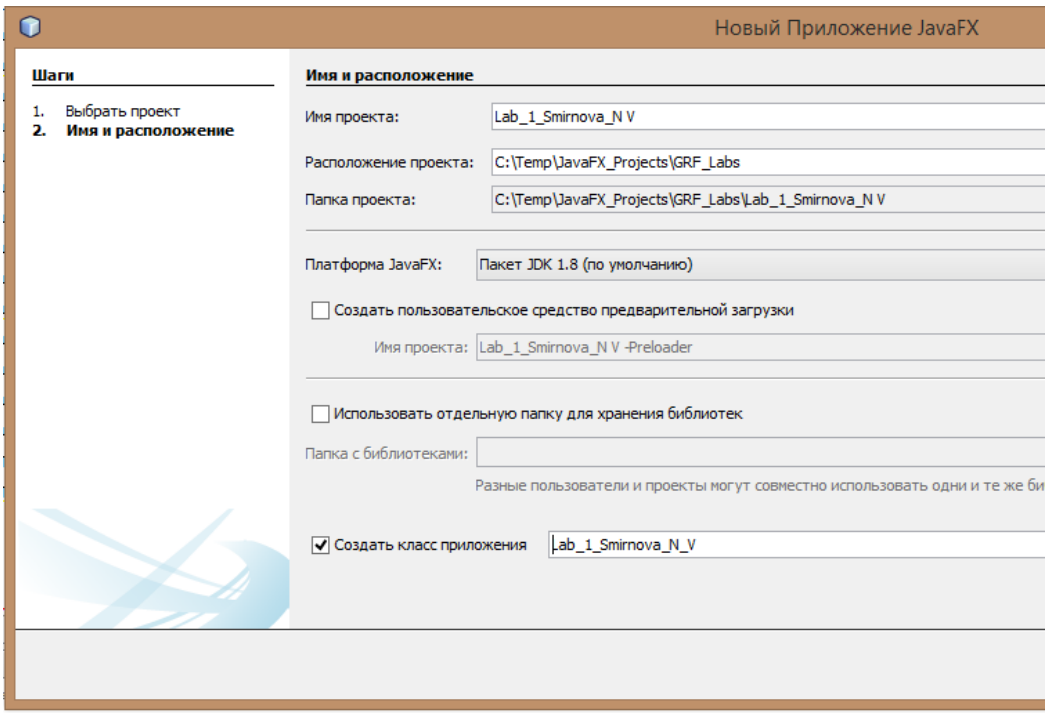
Робота з NetBeans IDE

Створення проекту.

Запустити NetBeans IDE, у меню «Файл» вибрати пункт «Створити проект». Вибрати пункт JavaFx, Додаток JavaFx і нажати кнопку «Далі».



Вибрати папку робочого проекту і ім'я проекту вказати латинським шрифтом, наприклад**:**



Буде створений проект і шаблон програми:

```
public class Lab_1_Smirnova_N_V extends Application {

    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(new EventHandler<ActionEvent>() {

            @Override
            public void handle(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();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }
}
```

Шаблон програми привести до вигляду:

```
//==================================================================
// Лабораторная работа № 1
// class: Lab_1_Smirnova_N_V
// Copyright (c) 2015 Smirnova N.V., гр. |группа|
//==================================================================
package lab_1_smirnova_n_v;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
```

...

```
//==================================================================
// Главный класс
//==================================================================
public class Lab_1_Smirnova_N_V extends Application {

    //===== объявления
    final int THICKNESS_MAX = 50;     //максимальная толщина линии
    //
    Group root = new Group();         //корень узлов сцены
    //
    TextArea display;                 //область вывода
    TextField in;                     //поле ввода
    Text txt;                         //текстовая надпись
    Slider slider;                    //слайдер

//==================================================================
// Начало
//==================================================================
@Override
public void start(Stage stage) {
    //====== заголовок окна
    stage.setTitle("Лабораторная работа №1. доц. Смирнова Н.В.");
    stage.setResizable(false);        //фиксированный размер
    //====== создать узлы сцены с графикой
    CreateGraphNodes();
    //====== создать узлы сцены с элементами управления
    CreateControlNodes();
    //====== создать сцену с размерами и цветом фона
    Scene scene = new Scene(root, 460, 250, Color.CORNSILK);
    //====== поместить сцену в окно
    stage.setScene(scene);
    //====== отобразить окно
    stage.show();
}
//==================================================================
//  Точка входа в программу (запускает метод start)
//==================================================================
public static void main(String[] args) {
    launch(args);
}
```

Ввести код демонстраційної програми:

```
//=============================================================
// Лабораторная работа № 1
// class: Lab_1_Smirnova_N_V
// Copyright (c) 2015 Smirnova N.V.
//=============================================================
package pro_lab_1_smirnova_n_v;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
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.ColorPicker;
import javafx.scene.control.RadioButton;
import javafx.scene.control.Slider;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.control.ToggleGroup;
import javafx.scene.effect.DropShadow;
import javafx.scene.layout.Border;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Pane;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.scene.shape.StrokeLineCap;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import javafx.stage.StageStyle;

/**
 *
 * @author Smirnova N.V.
 */
//=================================================================
// Главный класс
//=================================================================
public class PRO_Lab_1_Smirnova_N_V extends Application {

    Pane root_pane = new Pane();        //группа для узлов сцены
    Group group = new Group();          //группа для узлов сцены
    //===== элементы управления
    Slider slider;                      //слайдер
    Button btn_inc;                     //кнопка инкремент
    Button btn_dec;                     //кнопка декремент
    //====== 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;                  //дисплей
    //===== графика
    DropShadow dropShadow;              //тень
    Text text;                          //текстовая надпись
    //====== счетчик
    int counter = 0;
    int MAX_VAL = 50;                   //максимальное значение счетчика

    //#################################################################
    //  Создание узлов (nodes)
    //#################################################################
    //=================================================================
    // Тень
    //=================================================================
    private void Shadow() {
        dropShadow = new DropShadow();      //создать объект тень
        dropShadow.setRadius(5.0);          //закругление углов тени
        dropShadow.setOffsetX(5.0);         //смещение тени по X и Y
        dropShadow.setOffsetY(5.0);
        dropShadow.setColor(Color.GRAY);    //цвет тени
    }
    //=================================================================
    // Графика
    //=================================================================
    private void CreateGraphNodes() {
        //====== создать тень для узлов сцены
        Shadow();
        //====== применить эффект для всех узлов группы
        //для класса Pane тень не устанавливается, поэтому в Pane помещаем
Group
        group.setEffect(dropShadow);
        //=== надпись
        text = new Text();                  //создать текст
        text.setFont(Font.font("Arial", FontWeight.BOLD, 16));
        text.setFill(Color.BLUE);           //цвет текста
    }
    //=================================================================
    // Элементы управления
    //=================================================================

    private void CreateControlNodes() {

        //====== RadioButton
        ToggleGroup tg = new ToggleGroup();     //группа 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 инкремент
btn_inc = new Button();                //создать кнопку
btn_inc.setText("Увеличить");          //надпись на кнопке


btn_inc.setLayoutX(120);               //расположение кнопки на сцене
btn_inc.setLayoutY(200);
btn_inc.setTextFill(Color.BROWN);
btn_inc.setFont(Font.font("Arial", FontWeight.BOLD, 12));

//====== кнопка декремент
btn_dec = new Button();                //создать кнопку
btn_dec.setText("Уменьшить");          //надпись на кнопке
btn_dec.setLayoutX(20);                //расположение кнопки на сцене
btn_dec.setLayoutY(200);
btn_dec.setTextFill(Color.BROWN);
btn_dec.setFont(Font.font("Arial", FontWeight.BOLD, 12));

//====== дисплей
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();                  //создать слайдер
slider.setLayoutX(290);                 //координаты
slider.setLayoutY(200);
slider.setPrefWidth(200);               //длина слайдера
slider.setBlockIncrement(1.0);          //единица изменения
slider.setMajorTickUnit(10);            //большие деления на шкале
slider.setMinorTickCount(5);            //малые деления на шкале
slider.setMax(MAX_VAL);                 //максимальное значение
slider.setMin(1);                       //минимальное значение
slider.setShowTickLabels(true);         //показать значения шкалы
slider.setShowTickMarks(true);          //показать деления шкалы
slider.setSnapToTicks(false);           //не привязывать к делениям
//=== инициализация слайдера
slider.setValue(counter);               //установить ползунок слайдера

//====== скомпоновать сцену
GridPane gp = new GridPane();               //создать компоновщик
gp.setHgap(50);                             //шаг по горизонтали
gp.setVgap(10);                             //шаг по верткали
gp.setPadding(new Insets(25, 0, 0, 20));    //отступы от края окна
//====== поместить объекты в компоновщик. (узел, столбец, строка)
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

        //====== поместить все узлы в корень
        group.getChildren().addAll(gp, slider, btn_dec, btn_inc, textArea);
        root_pane.getChildren().addAll(group);
        //====== отобразить установки параметров
        toDisplay(counter);
    }

    //=================================================================
    // Обработчики событий
    //=================================================================
    private void onAction() {
        ////////////////////////////////////////////////////////////
        //====== Button
        //=== обработчик события нажатия кнопки
        btn_inc.setOnAction((ActionEvent event) -> {
            if (counter < MAX_VAL) {
                counter += 1;                //увеличить счетчик
            }
            toDisplay(counter);              //отобразить значение счетчика
        });
        //=== обработчик события нажатия кнопки
        btn_dec.setOnAction((ActionEvent event) -> {
            if (counter > 1) {
                counter -= 1;                //уменьшить толщину линии
            }
            toDisplay(counter);              //отобразить значение счетчика
        });
        ////////////////////////////////////////////////////////////
        //====== Slider
        //=== обработчик события ползунка слайдера
        slider.valueProperty().addListener((observable) -> {
            if (slider.isValueChanging()) {
                counter = (int) slider.getValue(); //получить значение
                                                   //ползунка
                toDisplay(counter);                //отобразить значение
                                                   //счетчика
            }
        });
        ////////////////////////////////////////////////////////////
        //====== CheckBox
        //=== обработчик события CheckBox_1
        cb_1.setOnAction((ActionEvent event) -> {
            if(cb_1.isSelected()){
                textArea.appendText("\nУстановлен CheckBox 1");
            }else{
                textArea.appendText("\nСнят CheckBox 1");
            }
        });
        //=== обработчик события CheckBox_2
        cb_2.setOnAction((ActionEvent event) -> {
```

```
        if(cb_2.isSelected()){
            textArea.appendText("\nУстановлен CheckBox 2");
        }else{
            textArea.appendText("\nСнят CheckBox 2");
        }
    });
    //=== обработчик события CheckBox_3
    cb_3.setOnAction((ActionEvent event) -> {
        if(cb_3.isSelected()){
            textArea.appendText("\nУстановлен CheckBox 3");
        }else{
            textArea.appendText("\nСнят CheckBox 3");
        }
    });
    ////////////////////////////////////////////////////////
    //====== RadioButton
    //=== обработчик события RadioButton_1
    rb_1.setOnAction((ActionEvent t) -> {
        textArea.appendText("\nУстановлен RadioButton 1");
    });

    //=== обработчик события RadioButton_2
    rb_2.setOnAction((ActionEvent t) -> {
        textArea.appendText("\nУстановлен RadioButton 2");
    });
    //=== обработчик события RadioButton_3
    rb_3.setOnAction((ActionEvent t) -> {
        textArea.appendText("\nУстановлен RadioButton 3");
    });
}

//================================================================
// Вывод значения счетчика
//================================================================
public void toDisplay(double val) {
    String s = Integer.toString(counter);
    text.setText("Счетчик: " + s);      //значение счетчика
    textArea.appendText("\nСчетчик: " + s);   //значение счетчика
    slider.setValue(counter);           //установить ползунок слайдера
}

//================================================================
// Начало
//================================================================
@Override
public void start(Stage stage) {
    //====== заголовок окна
    stage.setTitle("Лабораторная работа №1. доц. Смирнова Н.В.");
    stage.setResizable(false);       //фиксированный размер
    //====== создать узлы сцены с графикой
    CreateGraphNodes();
    //====== создать узлы сцены с элементами управления
    CreateControlNodes();
    //====== создать обработчики событий
    onAction();
    //root_pane.setOpacity(0.50);
    //====== создать сцену с размерами и цветом фона
     Scene scene = new Scene(root_pane, 500, 250, Color.TRANSPARENT);
```

```
        //====== поместить сцену в окно
        stage.setScene(scene);
        //====== отобразить окно
        stage.show();
    }

    //=================================================================
    //  Точка входа в программу (запускает метод start)
    //=================================================================
    public static void main(String[] args) {
        launch(args);
    }

}
```

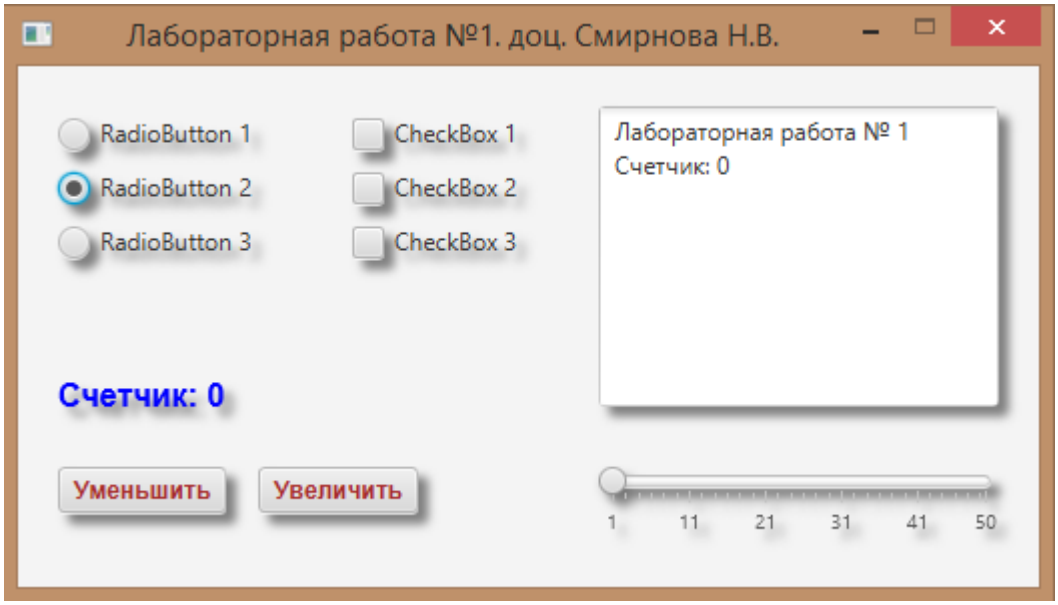
Код демонстраційної програми знаходиться в папці лабораторної роботи №1.

Компіляція програми здійснюється натисканням кнопки **F9.**

Компіляція і запуск програми здійснюється натисканням кнопки **F6.**

Створення файлу, що виконується, здійснюється **натисканням кнопки F11.**

Результат виконання програми:



Методичні вказівки до виконання лабораторної роботи:

1. Створити свій директорій для файлів проекту.
2. Створити проект в інтегрованому середовищі розробки NetBeans IDE (c:\Temp\Ваша_Папка).
3. **Створити алгоритм програми**
4. Написати програму мовою програмування Java (демонстраційна програма).
5. Змінити прізвище викладача на свою. Указати групу.
6. Скомпілювати програму.
7. Створити файл *.jar, що виконується, виконати.

Файл *.jar буде знаходиться в папці dist проекту.

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`.

Зміст звіту

У звіті повинні бути представлені:

- лістинг програми.
- висновки за результатами роботи.

Примітки: атрибути тексту програми

1. Шрифт – Courier New 10 пт.
2. Одиночний інтервал.
3. Файл програми з розширенням *.jar , що виконується і демонстраційний ролик з розширенням *.wmv знаходяться у папці “Приклади виконання”

Лабораторна робота №2

Тема: Створення додатка для паралельних обчислень

Ціль роботи

Одержати навички створення додатків для паралельних обчислень.

Завдання:

- Створити проект.
- Вибрати колір, довжину і ширину прямокутника відповідно до варіанта.
- Створити задачі і паралельні потоки обчислень.
- У кожному потоці обчислити площу прямокутника в пікселах.
- В окремому потоці обчислити сумарну площу прямокутників у пікселах.
- 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.

В 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

Багатопотокова система Java вбудована в клас `Thread`, що і доповнює його інтерфейс `Runnable`. 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 класом `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:

```
arcHeight, 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);
        }

    }
};
```

У 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:

```
//====== закрыть потоки при закрытии окна
stage.setOnCloseRequest(new EventHandler<WindowEvent>() {
    @Override
    public void handle(WindowEvent event) {
        toConsole("Close");
        task_1.cancel();
        task_2.cancel();
        task_3.cancel();
        task_4.cancel();
        task_5.cancel();

    }
});
```

Методичні 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.

Створення прямокутників здійснюється в такий спосіб:

```
//================================================================
// Создать прямоугольник. Базовый метод
//================================================================
Rectangle createRectangle(double beg_x, double beg_y, double w,
                          double h, Color color) {
    //====== создать прямоугольник
    Rectangle r = new Rectangle(beg_x, beg_y, w, h);
    r.setFill(color);
    return r;
}
```

```
//====== прямоугольники
rect_1 = createRectangle(100, 110, width, height, Color.RED);
rect_2 = createRectangle(250, 110, width, height, Color.GREEN);
```

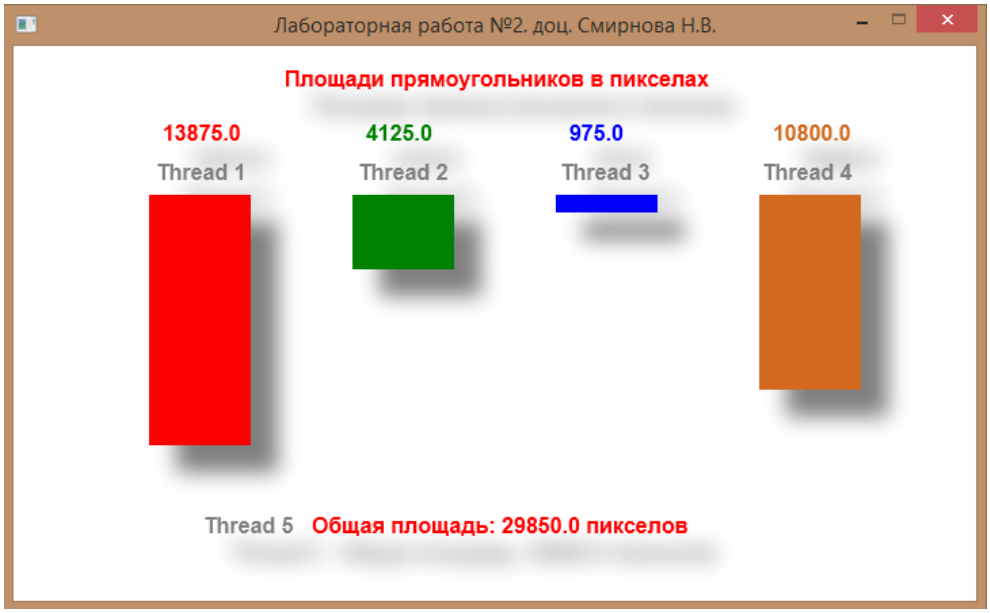
Базовий метод для створення текстових написів:

```
//=============================================================
// Создать дисплей. Базовый метод
//=============================================================
Text createDisplay(Color color) {
    Text t = new Text();              //создать текст
    t.setFont(Font.font("Arial", FontWeight.BOLD, 16));
    t.setFill(color);                          //цвет текста
    return t;
}
```

Зміна розмірів прямокутника здійснюється в такий спосіб:

```
//=============================================================
// Изменение размеров прямоугольника
//=============================================================
//====== 1 прямоугольник
public void pulseRectangle_1() {
    //===== изменение высоты
    if (flag_direction_1 == true) {
        rect_1.setHeight(rect_1.getHeight() + 1);
    } else {
        rect_1.setHeight(rect_1.getHeight() - 1);
    }
    //===== проверка границ
    if (rect_1.getHeight() == 1) {
        flag_direction_1 = true;
    }
    if (rect_1.getHeight() == MAX_HEIGHT) {
        flag_direction_1 = false;
    }
}
```

Результат виконання програми:



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.

Зміст звіту:

- лістинг програми.
- висновки за результатами роботи.

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

Примітки: Демонстраційний ролик з розширенням *.wmv знаходиться у папці “Приклади виконання”.

Лабораторна робота №3

Тема: Керування процесом паралельних обчислень

Ціль: Одержати навички керування процесом паралельних обчислень.

Завдання:

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

Теоретичні відомості:

Створення потоку

В Java для цього визначено два образи.

- За допомогою реалізації інтерфейсу `Runnable`.
- 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 `Runnable`. Щоб 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()`.

Методичні вказівки до виконання лабораторної роботи

В JavaFx модель потоків більш розвинена і стала схожою на багатозадачність. Тому потоки для програміста знаходяться у двох станах: стан виконання і стан останова.

Більш того, стало можливим виконувати в потоці не тільки класи і методи, але і окремі оператори. Наприклад, обчислення суми площ прямокутників простіше задати у вигляді задачи `Task` і помістити задачу в потік виконання `Thread`:

```
////////////////////////////////////////////
private void CreateTask_5() {
    //====== создать задачу 5
    task_5 = new Task<Void>() {
        String msg = new String("Общая площадь: ");
        @Override
        protected Void call() throws Exception {
            //====== постоянно
            while (true) {
                //===== поставить в очередь
                Platform.runLater(new Runnable() {
                    @Override
                    public void run() {
                        //=== вычислить сумму площадей
                        square_sum = (int) square_1 + (int) square_2 + (int) square_3 + (int) square_4;
                        //=== отобразить
                        square_display_sum.setText(msg + Double.toString((int) square_sum) + " кв.см");
                    }
                });
                //=== задержка
                Thread.sleep(10);
            }
        }
    };
    flag_created_5 = true;
    //======== запустить поток
    Thread t_5 = new Thread(task_5);
    t_5.setName("Поток 5");
    t_5.start();
}
```

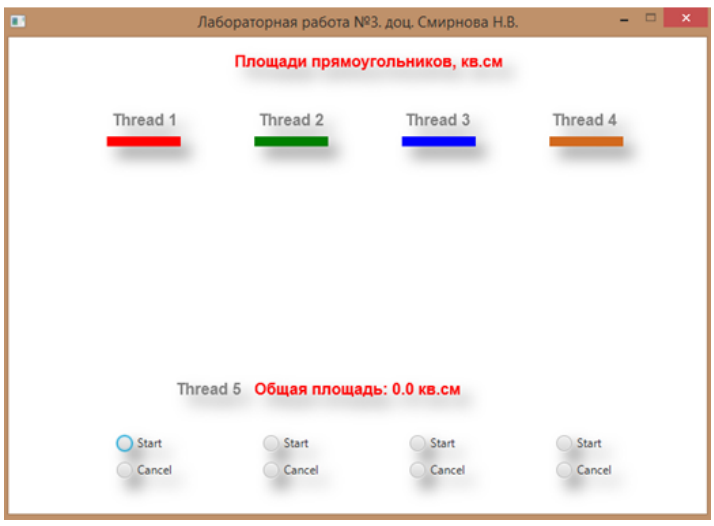
Використовуючи результати лабораторних робіт №2 створити головне вікно, прямокутники і розмістити об'єкти в сцені.

Елементи керування повинні здійснювати наступні дії:

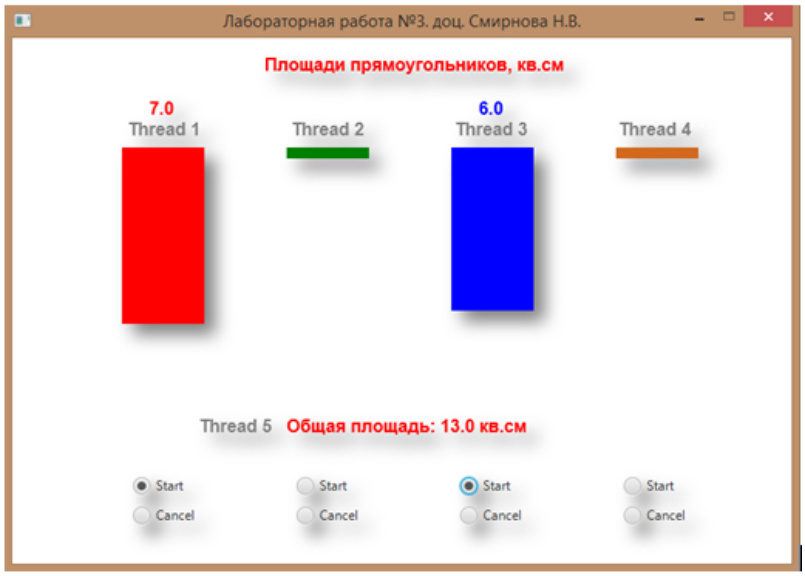
- Запустити потік.
- Зупинити потік.

Результат виконання програми:

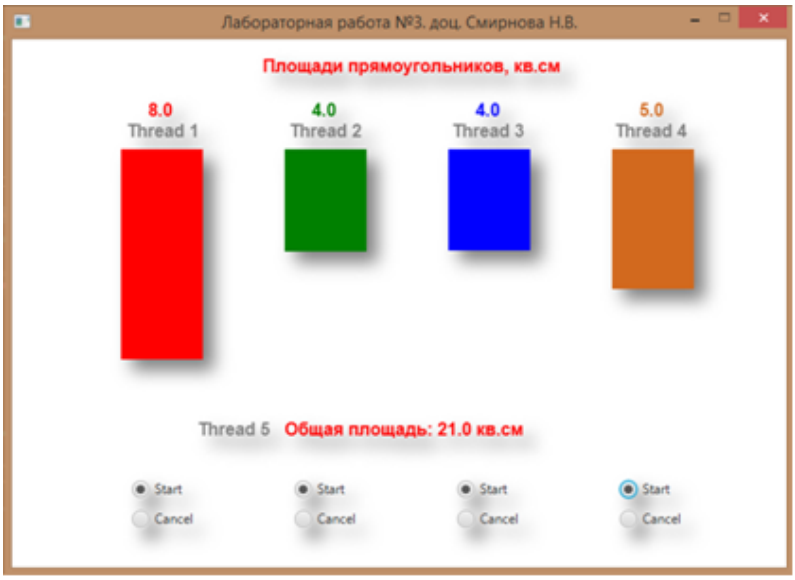
Потоки не запущені:



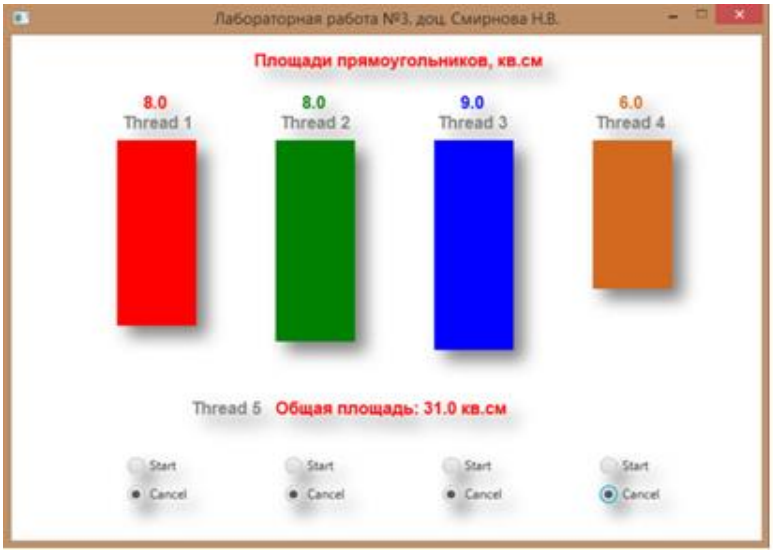
Запущені два потоки:



Запущені всі потоки:



Зупинені всі потоки:



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.

Зміст звіту:

1. Лістинг програми.
2. Висновки за результатами роботи.

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

Примітки: Демонстраційний ролик з розширенням *.wmv знаходиться у папці “Приклади виконання”.

Лабораторна робота № 4

Тема: Паралельні обчислення в багатопроцесорних системах. Технологія Fork-Join

Ціль: Одержати навички створення паралельних обчислень у багатопроцесорних системах.

Завдання:

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

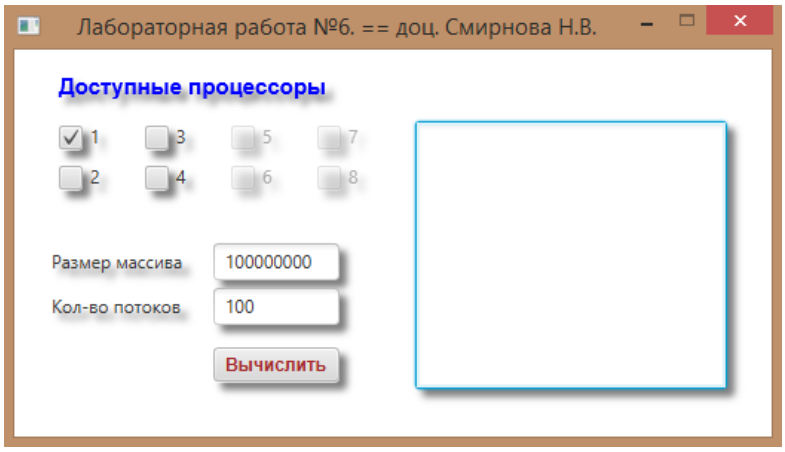
Методичні вказівки до виконання лабораторної роботи

запис

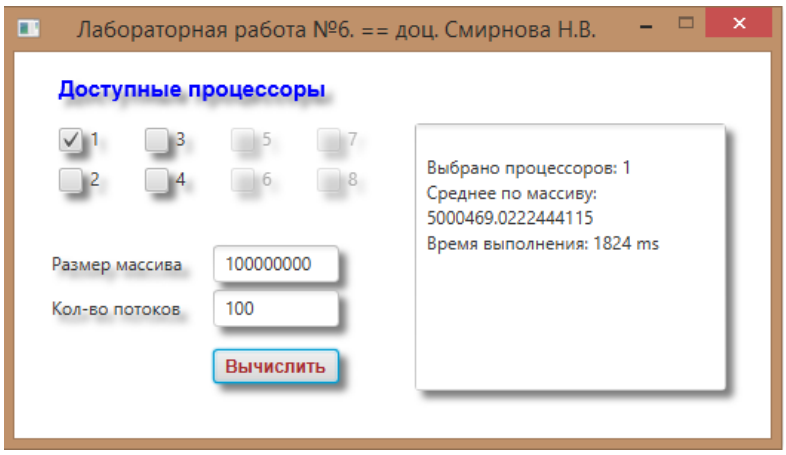
1. Створити додаток, що реалізує паралельні обчислення в багатопроцесорних системах.
2. У додатку реалізувати обчислення над масивом відповідно до варіанта.
3. Зробити аналіз результатів виконання паралельних обчислень для одного і декількох процесорів.
4. Зробити аналіз ефективності використання великої 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.

Результат виконання програми:

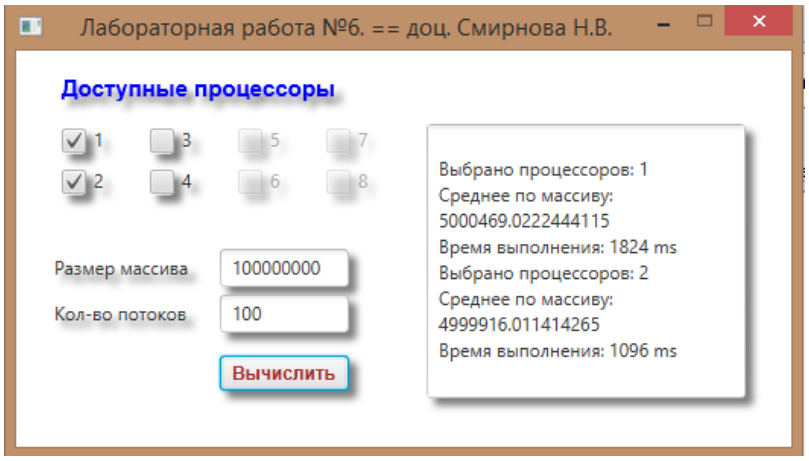
Запуск програми



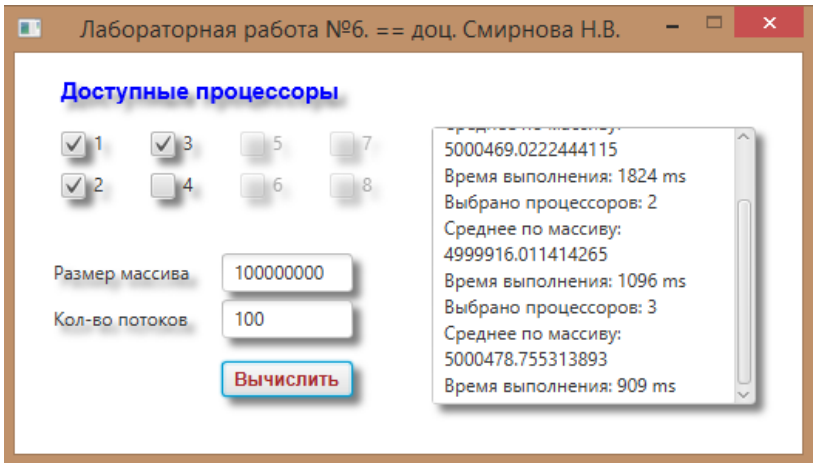
Паралельні обчислення на 1 процесорі



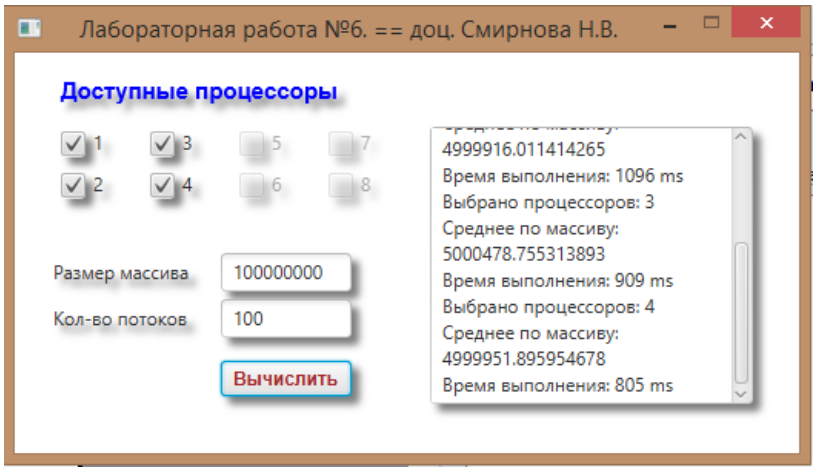
Паралельні обчислення на 2-х процесорах



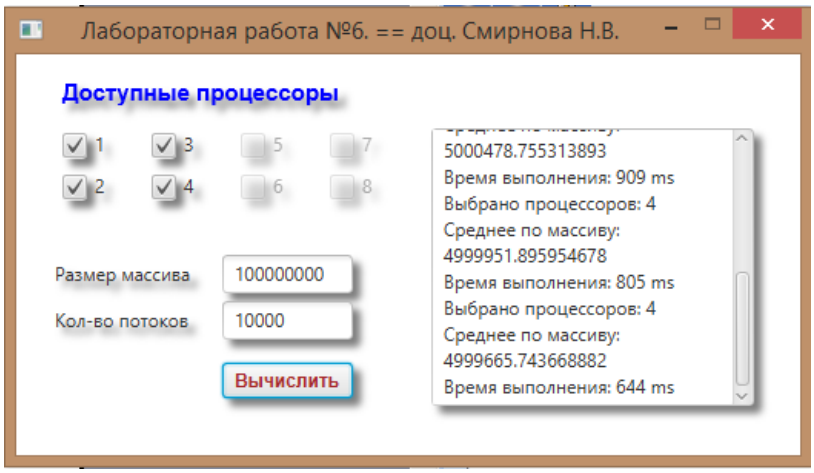
Паралельні обчислення на 3-х процесорах



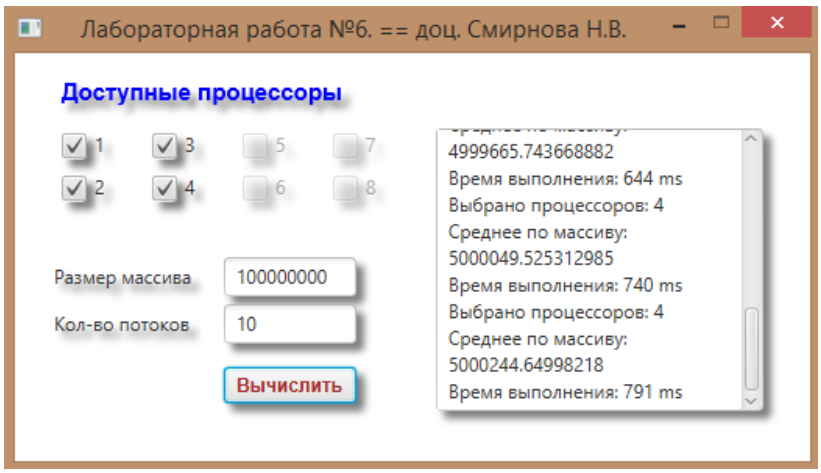
Паралельні обчислення на 4-х процесорах



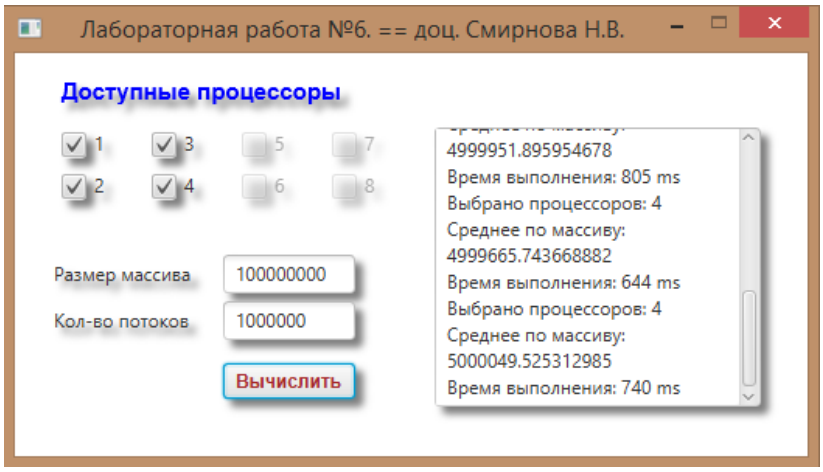
Паралельні обчислення на 4-х процесорах. Потоків – 10000



Паралельні обчислення на 4-х процесорах. Потоків – 10



Паралельні обчислення на 4-х процесорах. Потоків – 1000000



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).

Зміст звіту:

1. Лістинг програми.
2. Висновки за результатами роботи.

Варіанти:

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

Примітки: Демонстраційний ролик з розширенням *.wmv знаходиться у папці “Приклади виконання”.

Лабораторна робота № 5

Тема: Розподілені обчислення на базі технології Клієнт-Сервер

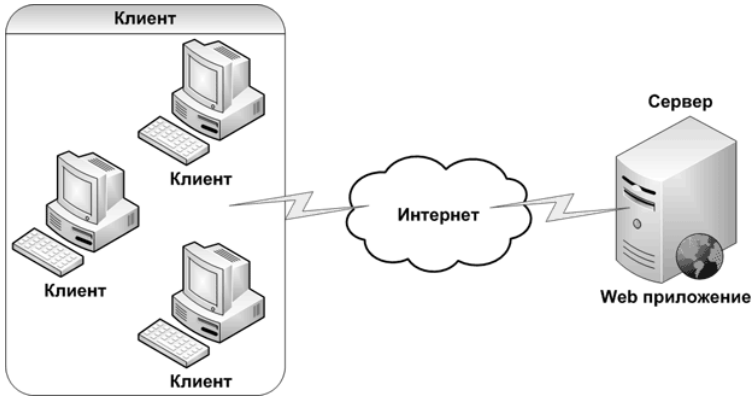
Ціль: Одержати навички створення і налагодження додатків для розподілених обчислень

Завдання:

- Створити проект для Клієнтської і Серверної програми.
- Реалізувати програмне забезпечення Сервера.
- Реалізувати програмне забезпечення Клієнта.
- Здійснити мережну взаємодію Клієнта і Сервера (установити з'єднання).
- На стороні Сервера забезпечити виконання обчислень за даними, отриманими від Клієнта.
- Результати обчислень повернути Клієнту.
- 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.



Малюнок 4.1 – Клієнт-Серверна архітектура Web-Додатків

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:

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

Методичні вказівки до виконання лабораторної роботи

1. Створити проект для Клієнтської і Серверної програми.
2. Реалізувати програмне забезпечення Сервера. Поставити у відповідність додатку порт № 8000.

Сервер створюється в такий спосіб:

```
//================================================================
// Создать сервер
//================================================================
private void createServerTask() {
    //====== создать задачу 1
    server_task = new Task<Void>() {
        @Override
        protected Void call() throws Exception {
            try {

                //====== создать сокет сервера
                ServerSocket serverSocket = new ServerSocket(port);

                while (true) {
                    //====== ожидать подключения клиента
                    Socket socket = serverSocket.accept();
                    //====== запустить поток сервера
                    new Thread(new Server(socket)).start();

                }

            } catch (IOException ex) {
                ex.printStackTrace();
                server_task.cancel();
            }
            return null;
        }
    };
}
```

3. Реалізувати програмне забезпечення Клієнта.

Сокет клієнта створюється в такий спосіб:

```
protected Void call() throws Exception {
    try {
        //====== создать сокет клиента
        try {
            socket = new Socket(host, port);
            flag_connected = true;
            //====== вывод сообщения Connected
            Platform.runLater(()
                    -> textArea.appendText(">>> Connected: "
                            + "\nPort: " + port
                            + "\nHost: " + host + '\n'
                    ));
        } catch (IOException ex) {
            flag_connected = false;
            //====== вывод сообщения Connected
            Platform.runLater(()
                    -> textArea.appendText("== Connection failed: "
                            + "\nPort: " + port
                            + "\nHost: " + host + '\n'
                    ));
            client_task.cancel();
            return null;
        }
```

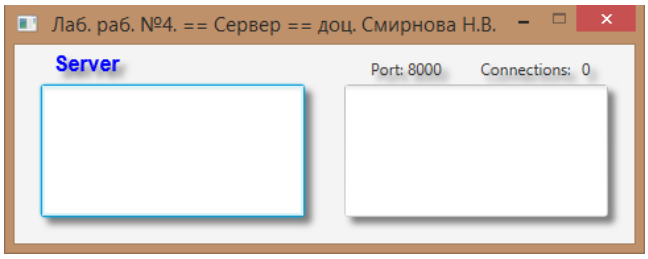
4. Здійснити мережна взаємодію Клієнта і Сервера (установити з'єднання).

5. На стороні Сервера забезпечити виконання обчислень за даними, отриманими від Клієнта.
6. Результати обчислень повернути Клієнту.

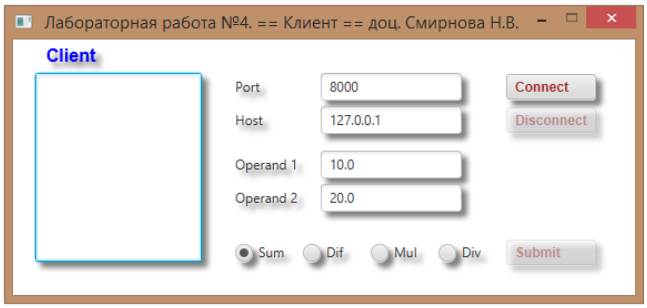
На стороні Клієнта забезпечити виведення результатів обчислень

Результат виконання програми:

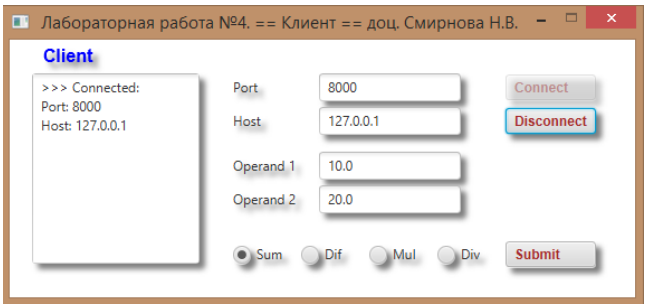
Сервер запущений



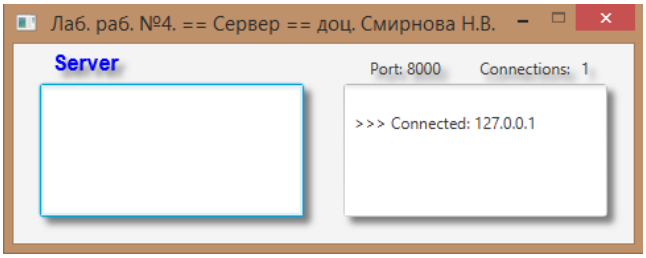
Клієнт запущений



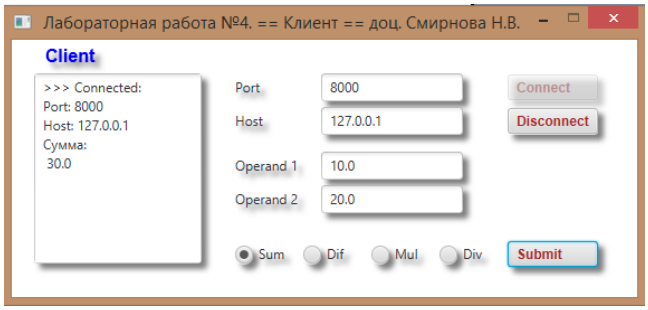
Підключення клієнта до сервера



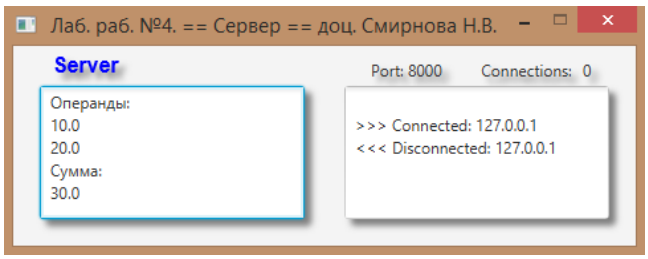
Клієнт підключений



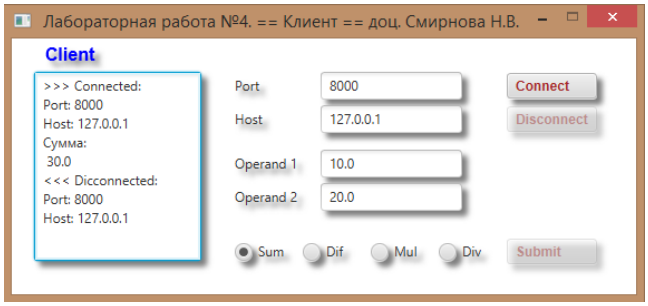
Клієнт робить запит на проведення обчислень сервером. Сервер обчислює суму чисел і повертає результат клієнту. Клієнт відображає результат обчислень.



Обчислення на стороні сервера. Сервер отримує завдання від клієнта, проводить обчислення і відсилає результат обчислення клієнту



Клієнт здійснює операцію відключення від сервера.



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.

Зміст звіту:

1. Лістинг програми.
2. Висновки за результатами роботи.

Примітки: Демонстраційний ролик з розширенням *.wmv знаходиться у папці “Приклади виконання”.

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

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

Завдання:

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

Теоретичні відомості:

Див. лабораторні роботи № 3 і № 4, 5

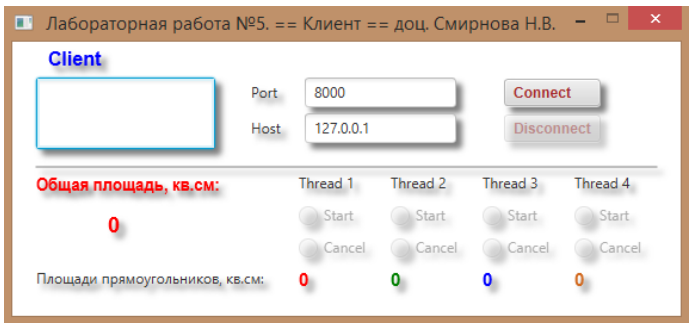
Методичні вказівки до виконання лабораторної роботи

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.

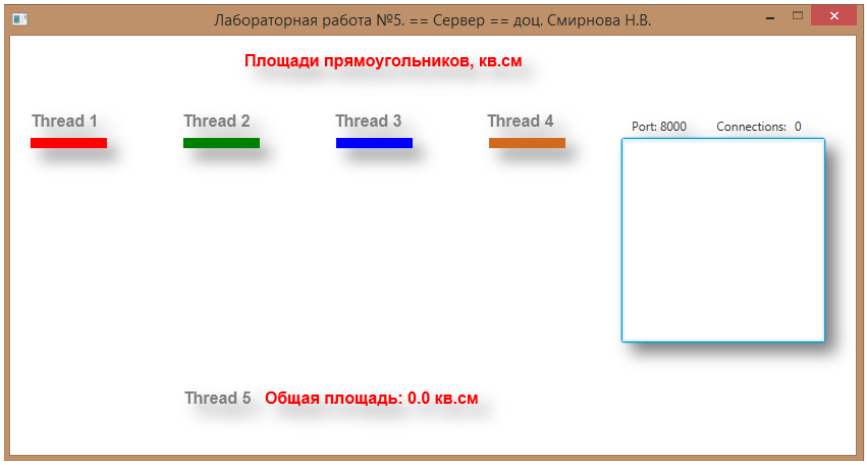
6. Здійснити процедуру обчислення площ прямокутників і реалізувати передачу результатів обчислень Клієнту.

Результат виконання програми:

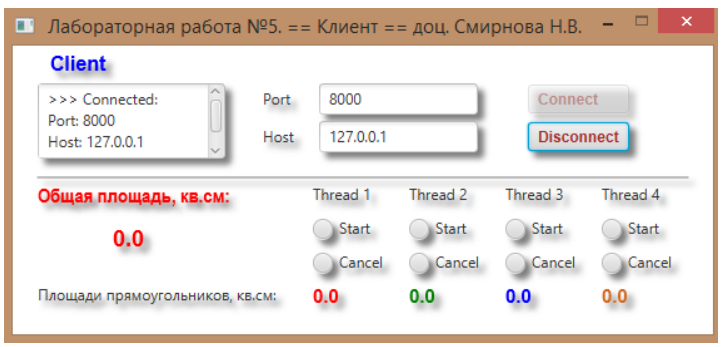
Запущена клієнтська програма на стороні клієнта



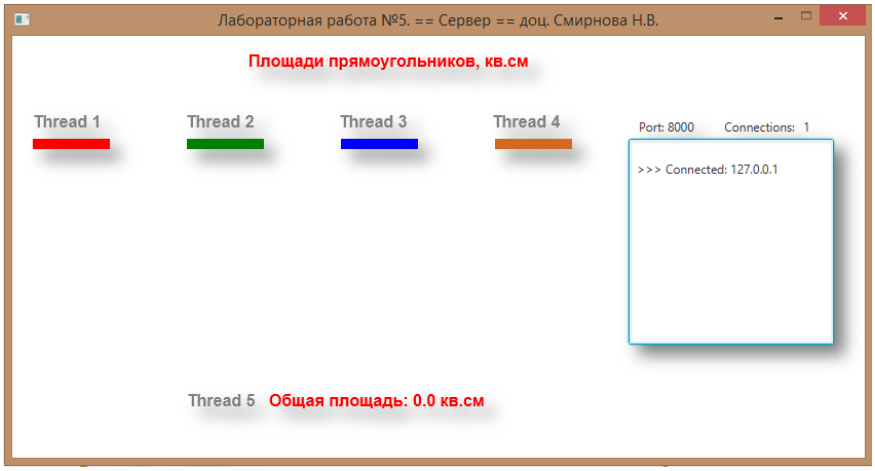
Запущена багатопотокова програма на стороні сервера



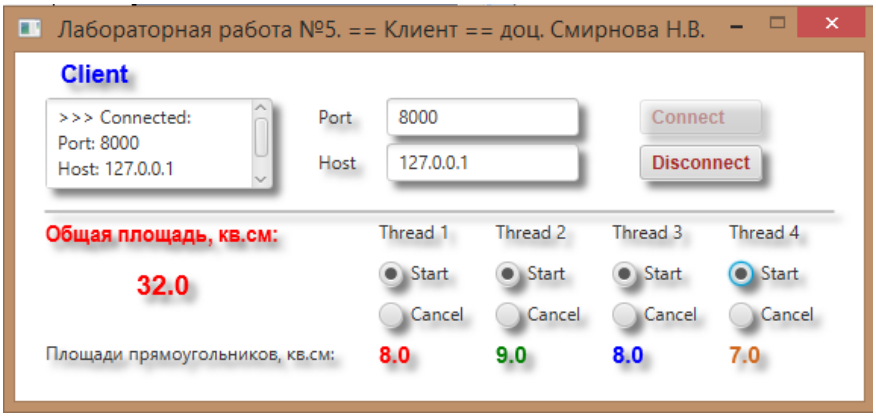
Виконане з'єднання клієнта і сервера. Клієнт



Виконане з'єднання клієнта і сервера. Сервер

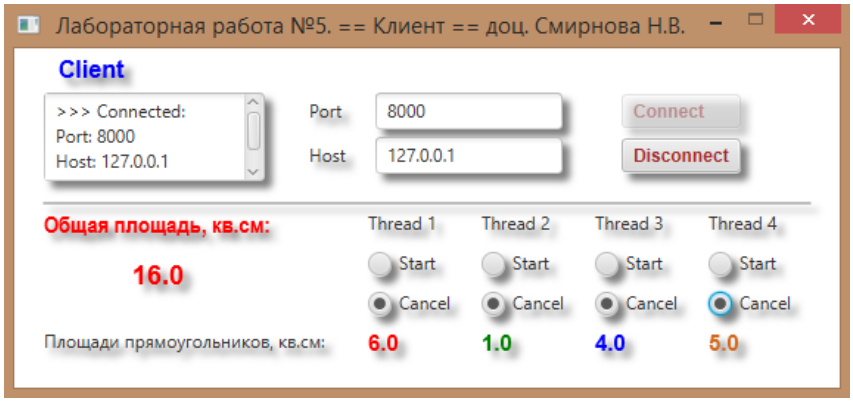


Клієнт дає команду серверу запустити потоки

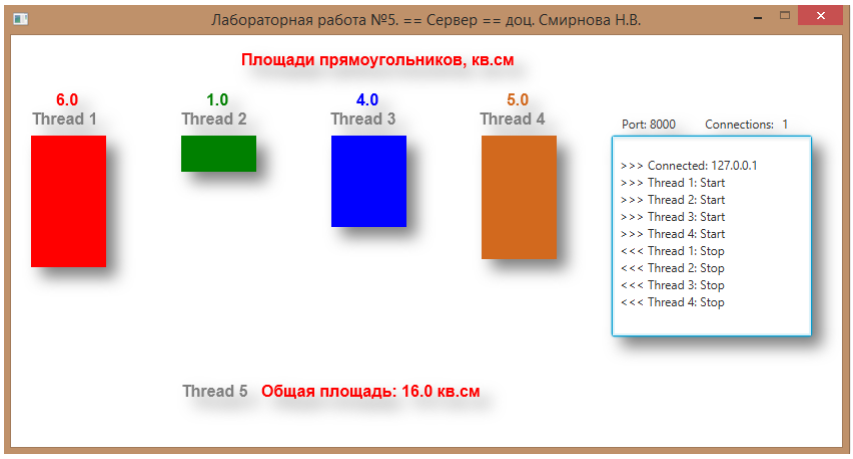


Сервер виконує команду і запускає потоки

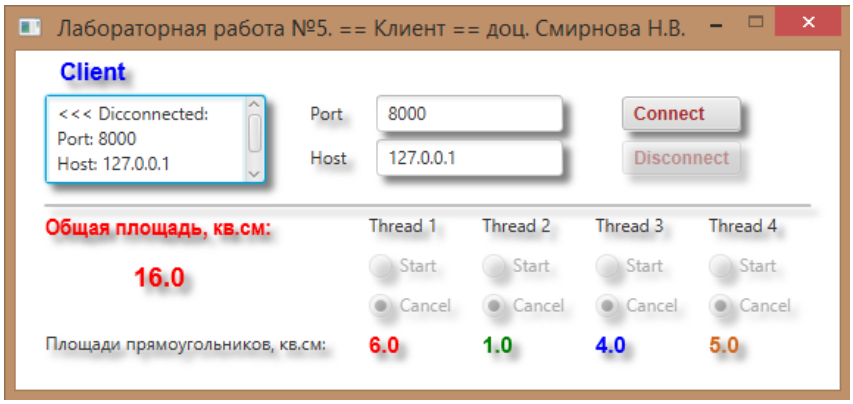
Клієнт дає команду серверу зупинити потоки

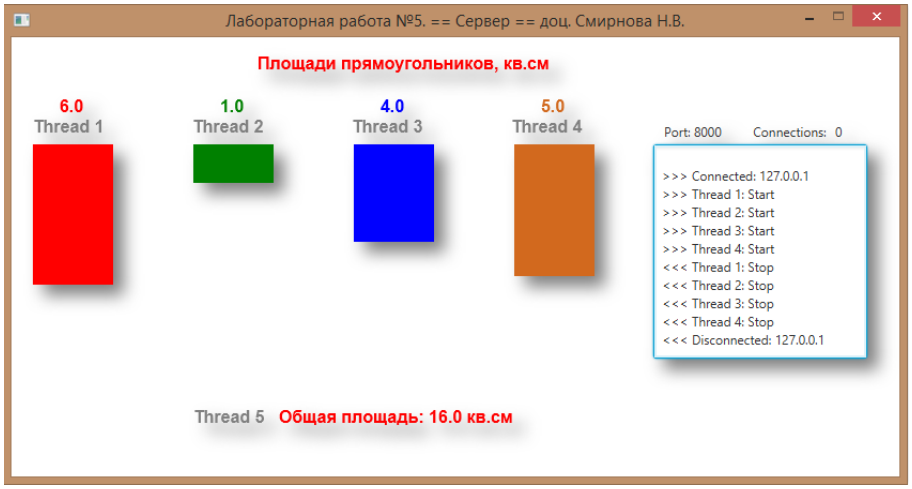


Сервер виконує команду і зупиняє потоки



Клієнт розриває з'єднання із сервером





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.

Зміст звіту:

1. Лістинг програми.
2. Висновки за результатами роботи.

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

Примітки: Демонстраційний ролик з розширенням *.wmv знаходиться у папці “Приклади виконання”.

Рекомендована література

1. http://www.oracle.com/
2. http://docs.oracle.com/apps/searchttparch.jsp?category=java&product=&q=JavaFx
3. Johan Vos Pro JavaFX 8 / Johan Vos, Weiqi Gao, 2014. – APRESS. – 604 p.
4. Carl Dea JavaFx 8: Introduction by Example / Carl Dea, Mark Heckler. - APRESS, 2014. - 420 p.
5. Тимур Машнин JavaFx 2.0: Разработка RIA-приложений / Тимур Машнин. – Спб.: БХВ-Петербург, 2012. - 320 с.