МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КІРОВОГРАДСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

МЕХАНІКО-ТЕХНОЛОГІЧНИЙ ФАКУЛЬТЕТ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Мультиплатформені мови програмування

Методичні вказівки до виконання лабораторних робіт

з елементами кредитно – модульної
системи організації навчального процесу

для студентів денної форми навчання
за напрямом підготовки 6.050102 «Комп'ютерна інженерія»

Укладачі:
Доцент Смірнова Н.В.
Доцент Смірнов В.В.

Кіровоград 2014

Мультиплатформені мови програмування: Методичні вказівки до виконання лабораторних робіт для студентів денної форми навчання за напрямом підготовки 6.050102 «Комп'ютерна інженерія» / Укл.:
Смірнов В.В., Смірнова Н.В. – Кіровоград: КНТУ, 2014. – 80 с.

Затверджено на засіданні кафедри ПЗ:
11 вересня 2013 р. протокол № 2;
17 вересня 2014 р., протокол № 4.

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

Для студентів денної форми навчання, що вивчають навчальну дисципліну "Мультиплатформені мови програмування" за напрямом підготовки 6.050102 "Комп'ютерна інженерія".

У стислій формі викладені основні принципи побудови обїектно-орієнтованих подієво-керованих додатків на мові програмування Java в середовищі розробки NET_Beans IDE.

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

ЗМІСТ

1. Опис навчальної дисципліни

" Мультиплатформені мови програмування "

Основна мета курсу полягає в придбанні досконалих знань і навичок роботи в середовищі мультиплатформенної мови програмування Java з застосуванням сучасних технологій та інструментальних засобів.

Внаслідок проведення лекцій студенти повинні отримати теоретичні знання та методику ефективної роботи з сучасними методами створення об'єктно-орієнтованих керованих подіями додатків

Завдання вивчення дисципліни

- Вивчення теоретичних основ об'єктно-орієнтованого проектування;
- Вивчення теоретичних основ об'єктно-орієнтованого програмування;
- Вивчення теоретичних основ методів створення багатокомпонентних додатків;
- Вирішення завдань роботи з файлами та дисками;
- Вирішення завдань створення елементів управління програми з використанням бібліотеки Swing;
- Набуття практичних навиків в сфері програмування багатопоточних додатків на основі технології Java.

Предметом навчальної дисципліни є створення багатопоточних об'єктно-орієнтованих подієво-керованих додатків в середовищі програмування Java.

У результаті вивчення навчальної дисципліни студент повинен

знати:

1. Основи міжплатформеної технології Java.
2. Принципи створення подієво-керованих програм.
3. Основи функціонування багатопотокових додатків.

вміти:

1. Вирішувати завдання створення подієво-керованих програм.
2. Вирішувати завдання створення об'єктно-орієнтованих додатків в середовищі розробки JavaBeans.
3. Вирішувати завдання управління потоками в процесі виконання багатопотокового додатку.
4. Програмувати приложения використанням технології JavaBeans і бібліотеки Swing.

2. Теми і зміст лекційних занять

№ теми	Тематика і зміст лекцій	Години
1.	**Введення в Java.** Типи даних. Керуючі конструкції. Створення проекту . Консольний додаток . Компіляція і запуск програми.	2
2.	**Класи.** Основи класів. Об'ява об'єктів. Конструктори. Члени класу. Повернення значення. Передача параметрів. Метод finalize ( ) . Перевантаження методів. Перевантаження конструкторів. Використання об'єктів як параметри. Повернення об'єктів. Ключове слово static . Ключове слово final.	4
3.	**Вкладені і внутрішні класи.** Успадкування. Ключове слово super. Порядок виклику конструкторів. Пакети та інтерфейси. Модифікатори доступу private, public і protected. Імпорт пакетів. Визначення інтерфейсу. Реалізація інтерфейсів. Змінні в інтерфейсах	4
4.	**Бібліотека Swing.** Архітектура MVC. Додаток Swing. Обробка подій. Класи JLabel і ImageIcon. Клас JTextField. Клас JButton. Клас JCheckBox. Клас JRadioButton. Клас JTabbedPane. Клас JScrolPane. Клас JList. Клас JComboBox. Клас JTable. Допоміжні класи.	6

№ теми	**Тематика і зміст лекцій**	Години
5.	**Винятки.** Обробка виняткових ситуацій. Блоки перехоплення винятків. Оператор throw. Ієрархія класів - винятків. Створення власних винятків.	2
6.	**Файли та друк.** Друк документів. Потоки введення/виведення. Консольне введення/виведення. Клас Console. Форматоване виведення. Файлове введення/виведення. Специфікації виведення. Отримання властивостей файлу. Робота з файлом засобами NIO2. Серіалізация об'єктів. Друк в Java. Друк засобами Java 2D.	4
7.	**Багатопотокове програмування.** Пріоритети потоків. Клас Thread і інтерфейс Runnable. Реалізація інтерфейсу Runnable. Розширення класу Thread	2
8.	**Бібліотечні функції Java.** Пакет java.util .	6
9.	**Графіка 2D .** Графічні засоби Java. Малювання фігур. Управління виведенням мультимедіа.	4
	Всього годин:	34

3. Лабораторні роботи з дисципліни "Мультиплатформені мови програмування"

№ заняття	Назва практичного заняття	Кільк. годин
1.	Освоєння IDE JavaBeans	2
2.	Створення класів Java та робота з класами	2
3.	Внутрішні класи і конструктори	2
4.	Бібліотека Swing - створення додатків і обробка подій	2
5.	Робота з елементами керування Swing	2
6.	Робота із класами JMenu, JProgressBar, JSlider і JSpinner	2
7.	Робота з файловою системою. Робота із класом RandomAccessFile	3
8.	Багатопотокове програмування	2
	Всього годин	**17**

4. Змістовні модулі

Навчальне навантаження складається з 3 – змістовних модулів, які включають в себе лекції, практичні роботи, самостійну роботу і контроль знань. Система оцінки успішності в балах включає тестовий поточний контроль при виконанні практичних робіт, модульний контроль і оцінку самостійної роботи.

Перший модуль.

Введення в Java. Класи. Вкладені і внутрішні класи

Лабораторні роботи:

№1 Освоєння IDE JavaBeans

№2 Створення класів Java та робота з класами

№3 Внутрішні класи і конструктори

Другий модуль.

Бібліотека Swing. Виключення. Файли та друк

Лабораторні роботи:

№4 Бібліотека Swing - створення додатків і обробка подій

№5 Робота з елементами керування Swing

Третій модуль.

Багатопотокове програмування. Бібліотечні функції Java. Графіка 2D

Лабораторні роботи:

№6 Робота із класами JMenu, JProgressBar, JSlider і JSpinner

№7 Робота з файловою системою. Робота із класом RandomAccessFile

№8 Багатопотокове програмування

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

№ модуля	Матеріал лекцій	Кількість лекцій (годин)	Бали за вивчення лекційного матеріалу	Лабораторні роботи				Реферат 8б за 1	Науково дослідна робота	Максимальна сума балів
				Теми лабораторних робіт	Години	Тестовий контроль	Виконання л.р.			
1.	**Тема 1.** Введення в Java **Тема 2.** Класи **Тема 3.** Вкладені і внутрішні класи	10	4	**№1** Освоєння IDE JavaBeans **№2** – Створення класів Java та робота з класами **№3** – Внутрішні класи і конструктори	6	10	10			24
2.	**Тема 1.** Бібліотека Swing **Тема 2.** Виключення	8	4	**№4** – Бібліотека Swing - створення додатків і обробка подій **№5** – Робота з елементами керування Swing	6	10	15	8		37
3.	**Тема 1.** Файли та друк **Тема 2.** Багатопотокове програмування **Тема 3.** Бібліотечні функції Java **Тема 4.** Графіка 2D	16	5	**№6** – Робота із класами JMenu, JProgressBar, JSlider і JSpinner **№7** – Робота з файловою системою. Робота із класом RandomAccessFile **№8** Багатопотокове програмування	5	10	10		14	39
	Всього:	34	13		17	30	35	8	14	100

6. Розподіл балів за змістовими модулями для визначення оцінки за результатами вивчення навчальної дисципліни

Модулі	Оцінка		
	«3»	«4»	«5»
Змістовий модуль 1.	10-14	15-19	20-24
Змістовий модуль 2.	24-29	30-33	34-37
Змістовий модуль 3	26-29	30-35	36-39
Всього за семестр	60 - 74	75 - 89	90 - 100

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 повторного складання	35-59
F	незадовільно з обов’язковим повторним курсом	1-34

Лабораторна робота №1

Тема: Освоєння IDE JavaBeans

Ціль роботи

Одержання навичок роботи з IDE **JavaBeans**, створення проекту, введення і компіляція програми, створення файлу *.jar, що виконується, запуск програми.

Завдання:

– Намалювати блок-схему алгоритму програми.

– Створити простий проект Java в IDE JavaBeans. Доповнити програму операцією обчислення різниці чисел a і b.

– 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.

```
//файл "Lab1.java".
class Lab1 {
     // Програма починається зі звертання до main().
     public static void main(String args[]){
          System.out.println("Проста програма Java.");
     }
}
```

Введення коду програми

Для більшості мов програмування ім'я файлу програми не має значення. Однак в Java це не так. Ім'я вихідному файлу, що привласнюється, дуже важливе. У цьому випадку іменем вихідного файлу повинне бути `Lab1.java`. І от чому:

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.

Тому, ім'я головного класу, що містить функцію `main()` повинне збігатися з іменем файлу, що містить програму. Необхідно також, щоб написання імені файлу відповідало імені класу, включаючи малі та великі літери.

Це обумовлене тим, що код Java чутливий до регістру символів.

Компіляція програми з командного рядка

Щоб скомпілювати програму `Lab1`, треба запустити компілятор `(javac)`, указавши ім'я вихідного файлу в командному рядку.

C:\>javac `Lab1`.java

Компілятор `javac` створить файл `Lab1.class`. - код віртуальної машини Java.

Щоб виконати програму, необхідно запустити віртуальну машину `Java`. При цьому їй буде потрібно передати ім'я класу `Lab1` у якості аргументу командного рядка, як показано в наступному прикладі.

C:\>java `Lab1`

При виконанні програми на екрані відобразиться наступне виведення.

```
Проста програма Java.
```

Компіляція і виконання програми в IDE JavaBeans

Компіляція програми здійснюється натисканням кнопки **F9.**

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

Створення файлу, що виконується (*.jar), здійснюється натисканням **кнопки F11.**

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, застосовувані 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.

```
class Lab1 {
```

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 відкриваючою { і закриваючою } 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 компілятора, 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[]` повідомляє 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 яких повинен містити метод `main()`, щоб виконання було можливим.

Наступний рядок коду.

```
System.out.println("Проста програма Java.");
```

Цей рядок виводить на екран рядок тексту `"Проста програма Java."`,

Рядок починається з ідентифікатора класу `System`. `System` – це статичний клас, `out` – вихідний потік, `println` - метод для виведення даних на консоль.

У реальних програмах Java консольне введення/виведення використовується рідко. У більшості випадків консольне введення/виведення застосовується в простих службових і демонстраційних програмах.

Розробка програмного забезпечення

Розробка програмного забезпечення здійснюється мовою програмування Java в інтегрованому середовищу розробки IDE JavaBeans фірми Oracle.

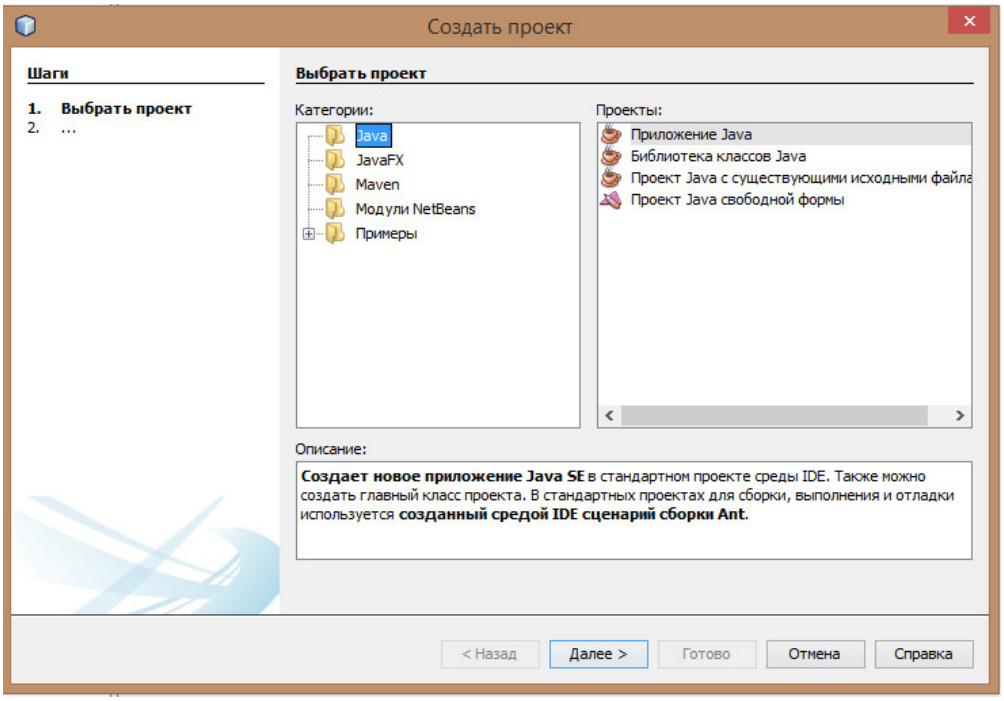
Порядок роботи:

1. Створити свій директорій для файлів проекту.
2. Створити проект в інтегрованому середовищі розробки IDE JavaBeans (c:\Temp\Ваша_Папка).
3. Написати програму мовою програмування Java (демонстраційна програма). Доповнити програму операцією обчислення різниці чисел a і b.
4. Скомпілювати програму.
5. Створити файл, що виконується, *.jar

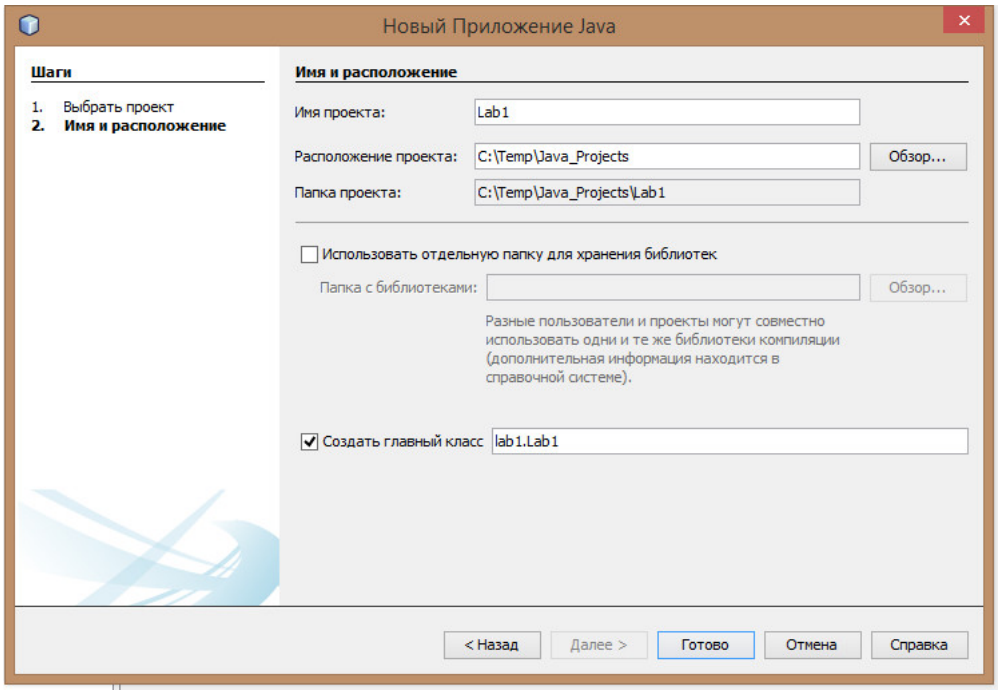
Робота з IDE JavaBeans

Створення проекту.

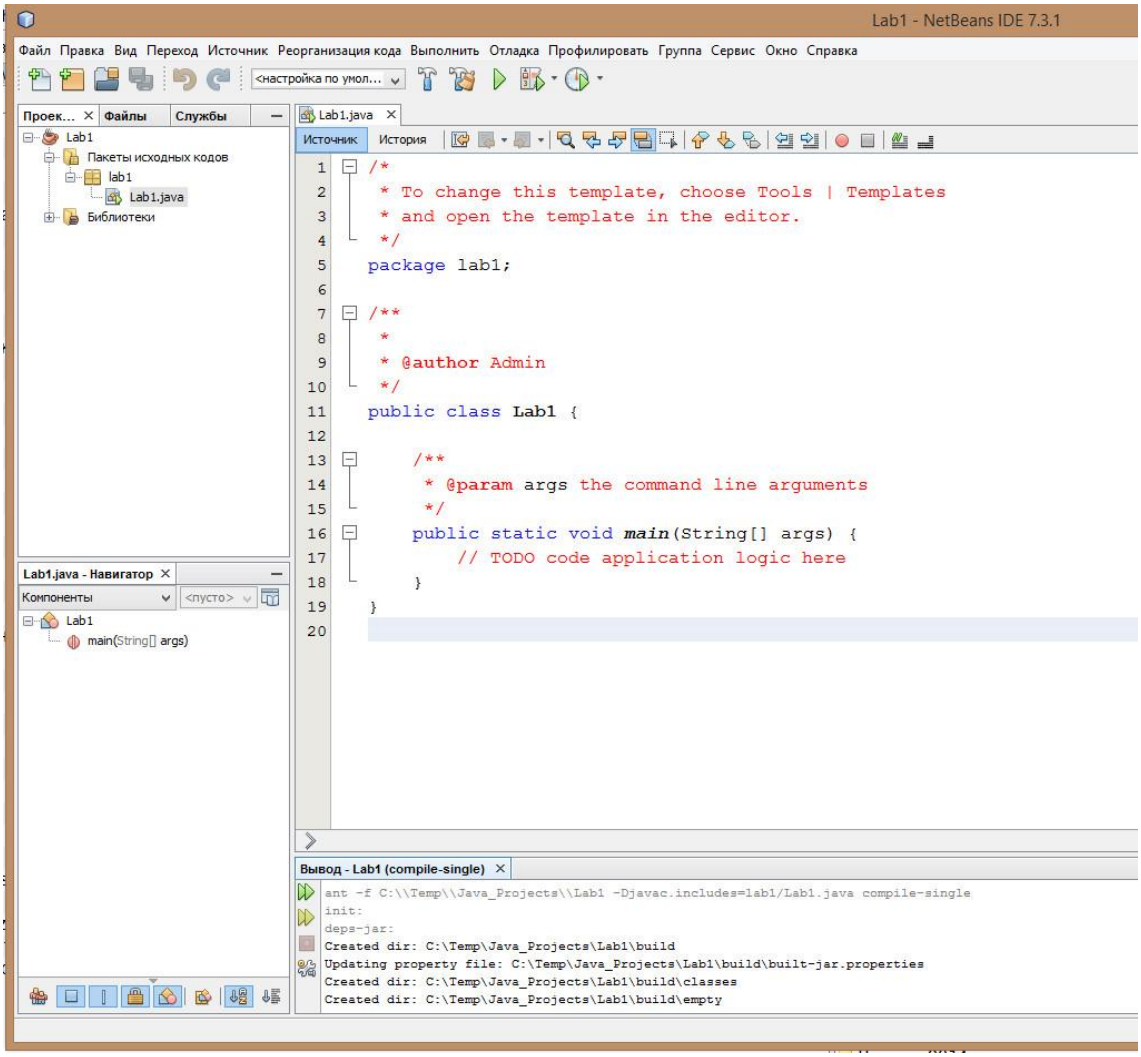
Запустити IDE, у меню «Файл» вибрати пункт «Створити проект». Вибрати свій директорій і записати в нього проект під будь-яким іменем латинським шрифтом:



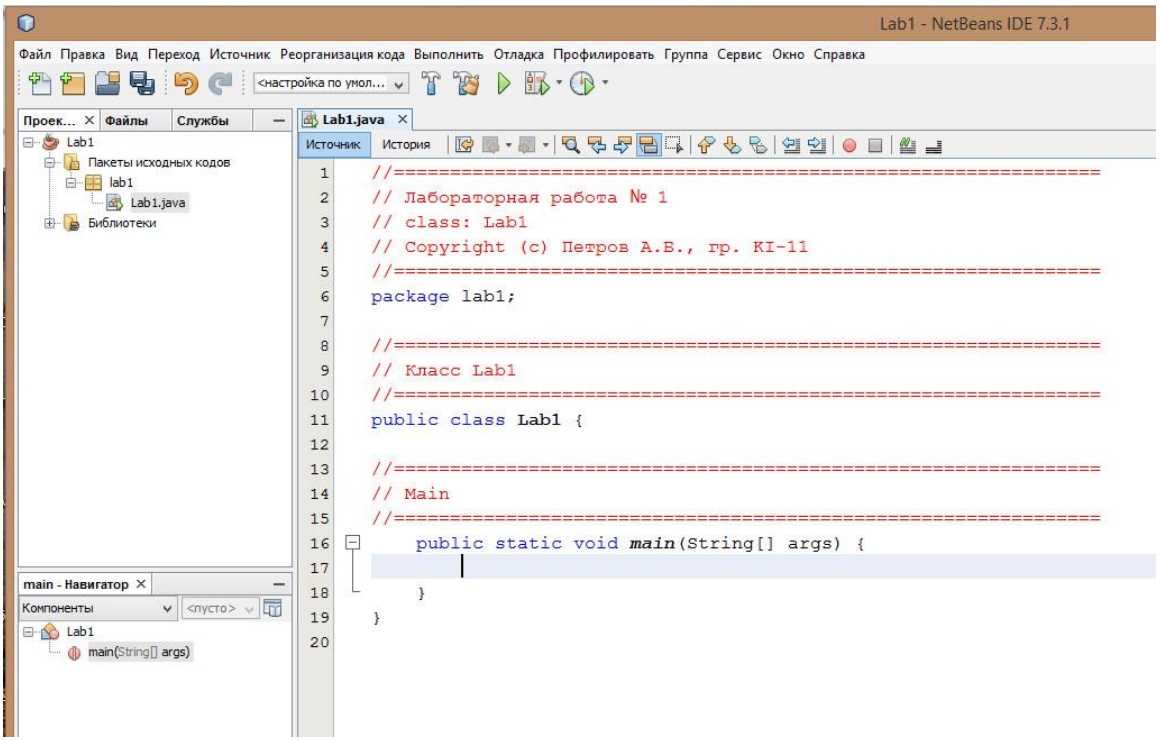
Вибрати папку робочого проекту і ім'я проекту, наприклад**:**



Буде створений проект і шаблон програми:



Шаблон програми привести до нормального вигляду:



Ввести код демонстраційної програми:

```
//=============================================================
// Лабораторная работа № 1
// class: Lab1
// Copyright (c) Петров А.Б., гр. КІ-11
//=============================================================
package lab1;
import javax.swing.*;

//=============================================================
// Класс Lab1
//=============================================================
public class Lab1 {

//=============================================================
// Main
//=============================================================
    public static void main(String[] args){
        //====== приветствие
        System.out.println("Студент Петров. гр.КІ-11");
        //====== объявление переменных
        int a = 3;
        int b = 5;
        //====== объявление с присвоением
        int sum = a+b;
        double div = (double)a/(double)b;
        int mul = a*b;
        String msg = "== Арифметические операции ==";

        //====== вывод на консоль
        System.out.println(msg);
        System.out.println("Сумма: "+ sum);
        System.out.println("Частное: "+ div);
        System.out.println("Произведение: "+ mul);

        //====== Вывод в окно. Необходим импорт классов библиотеки Swing
        //====== поместить в начало пакета: import javax.swing.*;
        JOptionPane.showMessageDialog(null, "Студент Петров. гр.КІ-11");
        msg += "\nСумма: "+ sum +"\n";
        msg += "Частное: "+ div +"\n";
        msg += "Произведение: "+ mul +"\n";
        JOptionPane.showMessageDialog(null, msg);

        //====== Диалог.
        String string_a = JOptionPane.showInputDialog("Введите первое число");
        //====== преобразовать тип String в тип int
        a = Integer.parseInt(string_a);
        String string_b = JOptionPane.showInputDialog("Введите второе число");
        //====== преобразовать тип String в тип int
        b = Integer.parseInt(string_b);
        //====== вычисления
        sum = a+b;
        div = (double)a/(double)b;
        mul = a*b;
        //====== вывод
        msg = "Вывод в окно\n";
        msg += "Сумма: "+ sum +"\n";
        msg += "Частное: "+ div +"\n";
        msg += "Произведение: "+ mul +"\n";
        JOptionPane.showMessageDialog(null, msg);

        JOptionPane.showMessageDialog(null, "Студент Петров задание выполнил");


    }
}
```

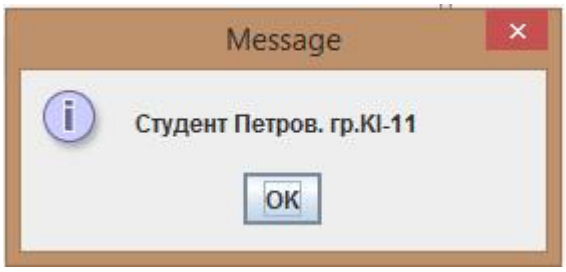
Компіляція програми здійснюється натисканням кнопки **F9.**

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

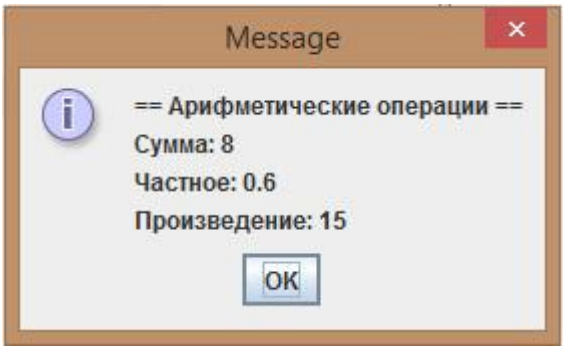
Створення файлу, що виконується, здійснюється натисканням **кнопки F11.**

Виведення повідомлень здійснюється за допомогою оператора :

```
JOptionPane.showMessageDialog(null, "Студент Петров. гр.КІ-11");
```

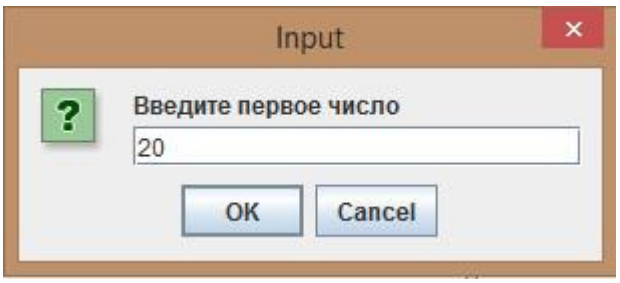


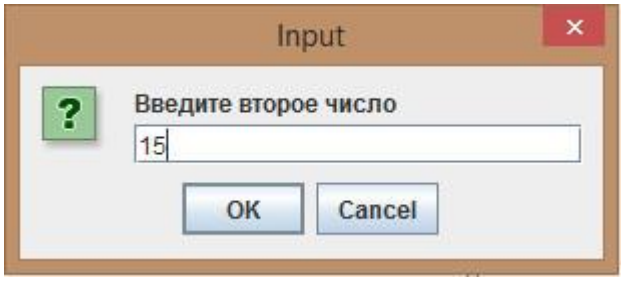

```
JOptionPane.showMessageDialog(null, msg);
```



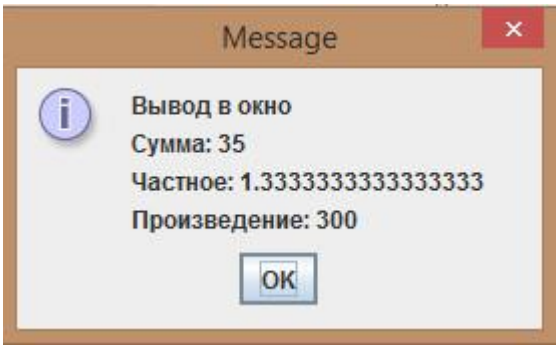
Діалог здійснюється за допомогою оператора:

```
String string_a = JOptionPane.showInputDialog("Сообщение");
```

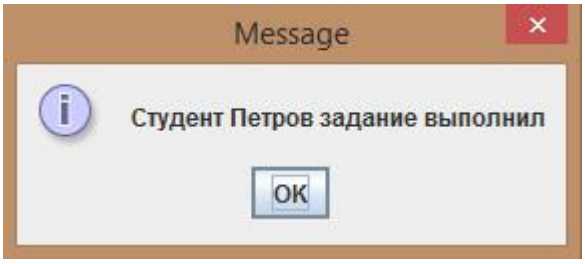




Результат виконання програми:



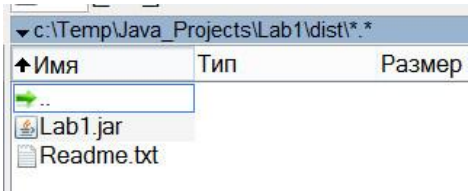
Завершення програми:



Методичні вказівки до виконання лабораторної роботи:

1. Створити свій директорій для файлів проекту.
2. Створити проект в інтегрованому середовищі розробки IDE JavaBeans (c:\Temp\Ваша_Папка).
3. **Створити алгоритм програми**
4. Написати програму мовою програмування Java (демонстраційна програма). Доповнити програму операцією обчислення різниці чисел a і b.
5. Скомпілювати програму.
6. Створити файл *.jar, що виконується, виконати.

Файл *.jar буде перебувати в папці dist проекту:



Контрольні питання:

1. Дати коротку характеристику мови програмування Java.
2. Описати елементарні типи даних, їхня розмірність.
3. Ідентифікатори, динамічна ініціалізація, приведення типів.
4. Дати визначення інкапсуляції, спадкування і поліморфізму.
5. Призначення елементів в описі функції `main`.

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

Зміст звіту

У звіті повинні бути представлені:

- блок-схема алгоритму роботи програми.
- лістинг програми.
- висновки за результатами роботи.

Примітки: атрибути тексту програми

1. Шрифт – Courier New 10 пт.
2. Одиночний інтервал.

Лабораторна робота №2

Тема: Створення класів Java та робота з класами

Ціль роботи

Одержати навички роботи з класами Java, конструкторами і перекритими методами, що реалізують параметричний поліморфізм.

Завдання:

- Створити проект.
- Створити три класи, у яких реалізувати:
  1. Функції виведення на консоль значень різних типів даних.
  2. Функції виведення у вікна повідомлень значень різних типів даних.
  3. Функціональність програми.
- Виконати програму

Теоретичні відомості:

Загальна форма класу

При визначенні класу повідомляють його дані та код, який взаємодіє з цими даними.

Хоча дуже прості класи можуть містити тільки код або тільки дані, більшість класів, які застосовуються у реальних програмах, містять обидва ці компонента.

Для оголошення класу служить ключове слово **class**. Спрощена загальна форма визначення класу має такий вигляд.

```
class ім'я_класу {
     тип змінна_екземпляра1;
     тип змінна_екземпляра2;
     // . . .
     тип змінна_ екземпляраN;
     тип ім'я_методу1(список_параметрів){
          // тіло методу
```

```
        }
        тип ім'я_методу2(список_параметрів){
              // тіло методу
        }
        // . . .
        тип ім'я_ методуN (список_параметрів){
              // тіло методу
        }
}
```

Дані, або змінні, визначені усередині класу, називаються *змінними екземпляра.*

Код міститься усередині *методів.* Визначені усередині класу методи і змінні разом називають *членами* класу.

У більшості класів дії зі змінними екземплярів і доступ до них здійснюють методи, визначені в цьому класі. Таким чином, у загальному випадку саме методи визначають спосіб використання даних класу.

Визначені усередині класу змінні називають змінними екземпляра, оскільки кожний екземпляр класу (тобто кожний об'єкт класу) містить власні копії цих змінних.

Таким чином, дані одного об'єкта відділені і відрізняються від даних іншого об'єкта.

Класи Java можуть не містити метод `main()`. Його обов'язково вказувати тільки в тих випадках, коли даний клас служить початковою точкою програми.

У мові Java оголошення класу і реалізація методів зберігаються в одному місці, а не визначені окремо, як у C++. Іноді ця особливість приводить до створення дуже великих файлів .java, оскільки будь-який клас повинен бути повністю визначений в одному файлі вихідного коду.

Простий клас

Нижче наведений код класу Box (Паралелепіпед), який визначає три змінні екземпляра: `width` (ширина), `height` (висота) і `depth` (глибина). У даний момент клас Box не містить ніяких методів.

```
class Box {
      double width;
      double height;
      double depth;
};
```

Як уже було сказано, клас визначає новий тип даних. У цьому випадку новий тип даних названий Box. Це ім'я буде використовуватися для оголошення об'єктів типу `Box`.

Важливо пам'ятати, що оголошення `class` створює тільки шаблон, але не дійсний об'єкт. Таким чином, наведений код не приводить до появи ніяких об'єктів типу `Box`.

Щоб дійсно створити об'єкт класу `Box`, потрібно використовувати оператор `new`:.

```
Box mybox = new Box(); // створення об'єкта mybox класу Box
```

Після виконання цього оператора об'єкт `mybox` стане екземпляром класу `Box`. Тобто він знайде "фізичне" існування.

Оголошення об'єктів

Створення об'єктів класу – двоступінчастий процес.

Спочатку необхідно оголосити змінну типу класу. Ця змінна не визначає об'єкт. Вона являє собою всього лише змінну, яка може *посилатися на* об'єкт.

Потім буде потрібно одержати дійсну, фізичну копію об'єкта і присвоїти її цій змінної. Це можна зробити за допомогою оператора `new`.

Цей оператор динамічно (тобто під час виконання) резервує пам'ять під об'єкт і повертає посилання на нього.

Загалом це посилання являє собою адресу об'єкта в пам'яті, зарезервованої оператором `new`. Потім це посилання зберігається в змінній. Таким чином, в Java усі об'єкти класів повинні створюватися динамічно. Розглянемо цю процедуру більш докладно.

У наведеному раніше прикладі програми рядок, подібний наступного, використовується для оголошення об'єкта класу `Box`.

```
Box mybox = new Box();
```

Цей оператор поєднує тільки що описані етапи. Щоб кожний етап був більш очевидним, його можна переписати в такий спосіб.

```
Box mybox; // оголошення посилання на об'єкт
mybox = new Box(); // резервування пам'яті для об'єкта Box
```

Перший рядок повідомляє **mybox** посиланням на об'єкт класу **Box**. Після виконання цього рядка змінна **mybox** містить значення **null**, що свідчить про те, що вона ще не вказує на реальний об'єкт.

Будь-яка спроба використання змінної **mybox** на цьому етапі приведе до помилки часу компіляції. Наступний рядок резервує пам'ять під реальний об'єкт і привласнює змінній **mybox** посилання на цей об'єкт.

Після виконання другого рядка змінну **mybox** можна використовувати

Докладний розгляд оператора new

Оператор **new** динамічно резервує пам'ять для об'єкта. Загальна форма цього оператора має такий вигляд.

```
змінна_класу = new ім'я_класу ();
```

Тут *змінна_класу* – змінна створюваного класу. *Ім'я_класу* – це ім'я класу, конкретизація якого виконується.

Ім'я класу, за яким ідуть круглі дужки, указує *конструктор* даного класу. Конструктор визначає дії, виконувані при створенні об'єкта класу.

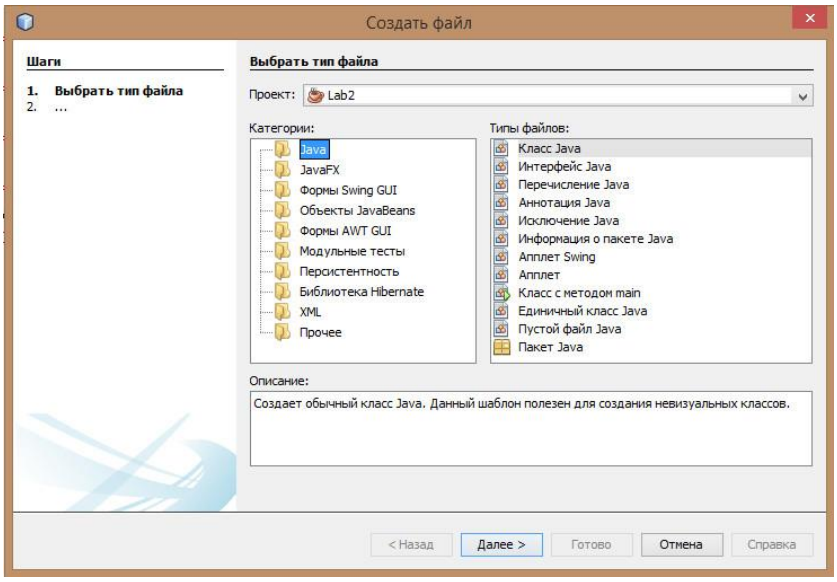
Важливо розуміти, що оператор **new** резервує пам'ять для об'єкта під час виконання. Перевага цього підходу полягає в тому, що програма може створювати рівно стільки об'єктів, скільки потрібно під час її виконання. Однак оскільки обсяг пам'яті обмежений, можлива ситуація, коли оператор **new** не зможе виділити пам'ять для об'єкта через її нестачу.

Методичні вказівки до виконання лабораторної роботи:

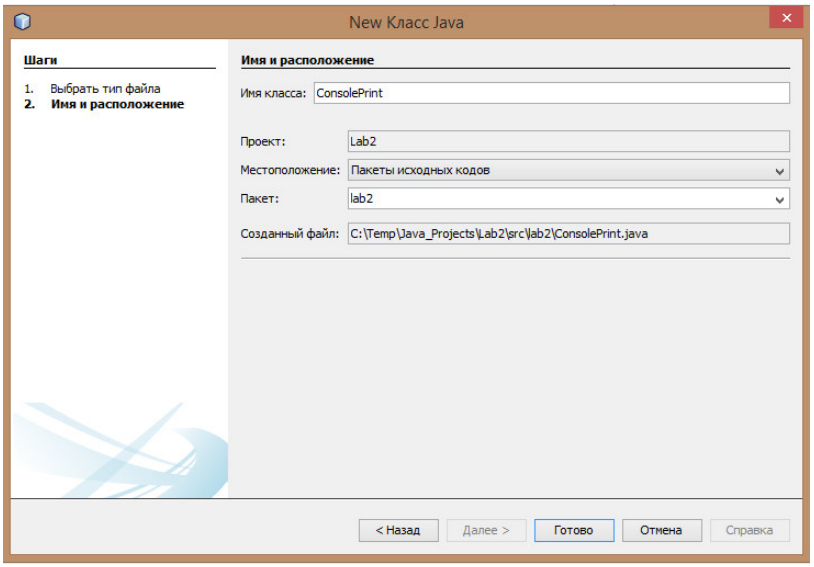
1. Створити свій директорій для файлів проекту.
2. Створити проект в інтегрованому середовищі розробки IDE JavaBeans (c:\Temp\Ваша_Папка).

3. Створити алгоритм програми
4. Реалізувати завдання – створити необхідні класи, типи даних і функції виведення на консоль і у вікна повідомлень.
5. Скомпілювати програму.

Створення класу здійснюється через меню «Файл» -> «Створити файл»:



Указати ім'я класу:



Аналогічні дії виконати для створення інших класів.

Клас із функцією main() повинен мати такий вигляд:

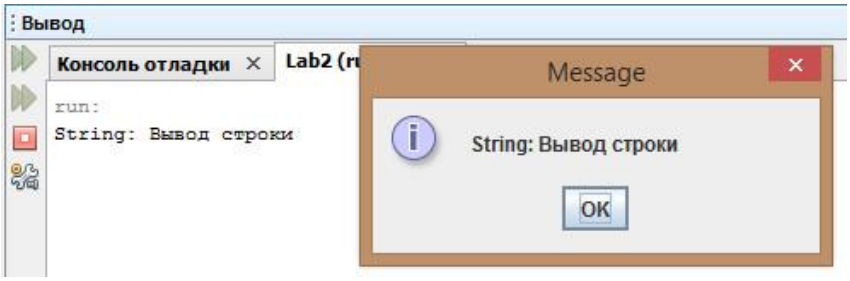
```
//==============================================================
// Лабораторная работа № 2
// class: Lab2
// Copyright (c) Петров А.Б., гр. КІ-11
//==============================================================
package lab2;

//==============================================================
// Класс Lab2
//==============================================================
public class Lab2 {
//==============================================================
// Main
//==============================================================
    public static void main(String[] args) {
        //=== создать главный класс. Здесь будет выполняться программа лаб.2
        MainClass prg = new MainClass();
        //=== запустить программу на выполнение
        prg.programm();
    }
}
```

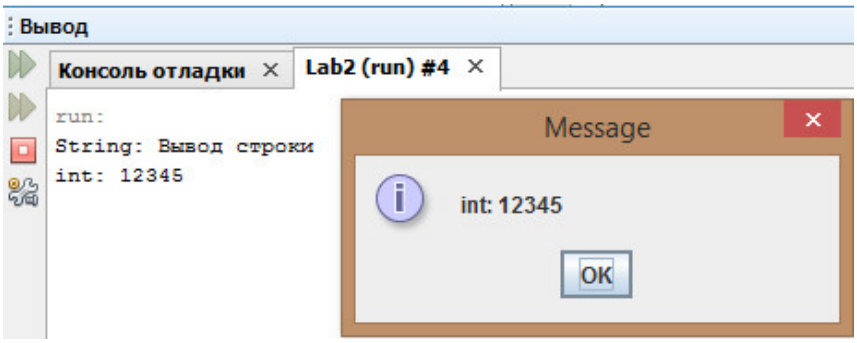
Важливо: у функції main() повинен бути створений тільки один клас, що реалізує функціональність програми.

Результат виконання програми:

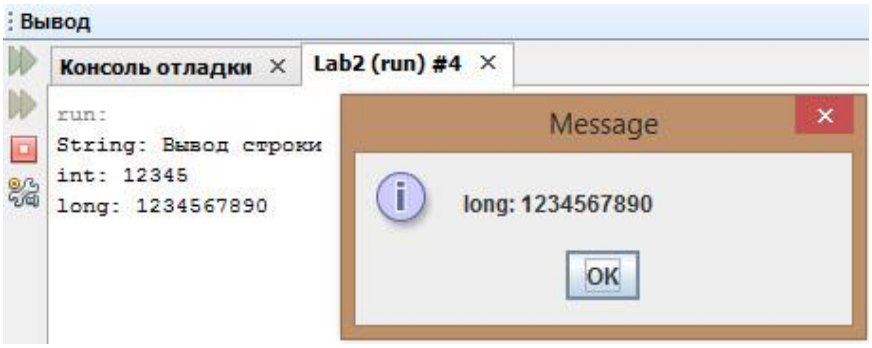
Виведення рядка:



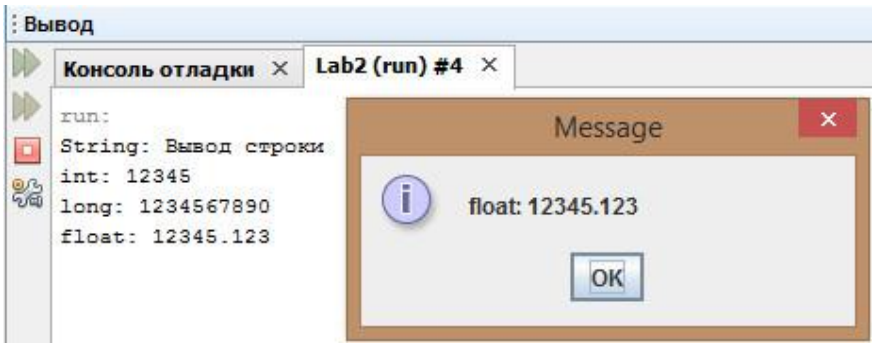
Виведення значення int:



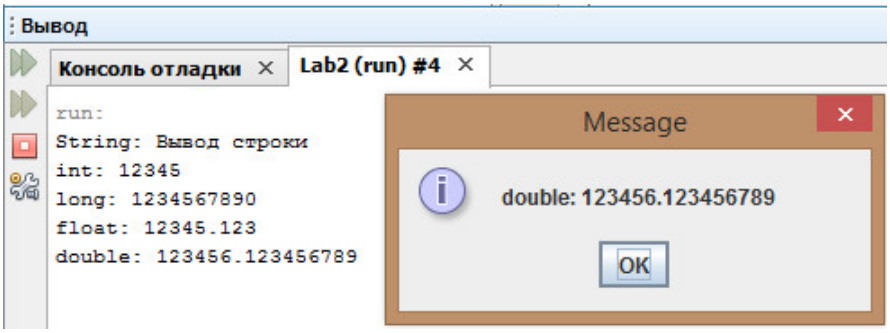
Виведення значення long:



Виведення значення float:



Виведення значення double:



Контрольні питання:

1. Порядок оголошення класу і створення об'єкта
2. Порядок створення конструктора та об'єктів у конструкторі.
3. Члени класу, їх призначення.
4. Порядок повернення значень із методу.
5. Призначення методу finalize(), ключового слова final і static

Зміст звіту:

- Структурна схема програми.
- лістинг програми.
- висновки за результатами роботи.

Лабораторна робота №3

Тема: Внутрішні класи і конструктори

Ціль роботи

Одержання навичок створення класів, розширення класів шляхом спадкування, і застосування конструкторів у класах

Завдання:

- Намалювати структурну схему програми.
- Створити проект.
- Використовуючи результати лабораторної роботи № 2:
  - створити клас консольного виведення,
  - створити клас виведення у вікно, розширивши попередній клас,
  - створити кінцевий клас, розширивши попередній. У результаті буде отриманий клас без єдиного методу, але, який зберігає функціональність двох класів.
- За допомогою кінцевого класу вивести на консоль і у вікно повідомлень значень різних типів даних, аналогічно завданню лабораторної роботи № 2.
- У кожний клас включити конструктор, у якому здійснити виведення повідомлення про конструктор класу.
- Виконати програму

Теоретичні відомості:

Спадкування

Одним з фундаментальних понять об'єктно-орієнтованого програмування є спадкування, оскільки воно дозволяє створювати ієрархічні класифікації. Використовуючи спадкування, можна створити загальний клас, який визначає характеристики, загальні для набору зв'язаних елементів.

Потім цей клас може успадковуватися іншими, більш спеціалізованими, класами, кожний з яких буде додавати свої унікальні характеристики.

У термінології Java наслідуваний клас називають ***суперкласом***. клас, що успадковує, зветься ***підкласом***.

Щоб успадковувати клас, досить просто вставити визначення одного класу в інший з використанням ключового слова extends. У якості ілюстрації розглянемо короткий приклад. Наступна програма створює суперклас A и підклас B. Зверніть увагу на використання ключового слова extends для створення підкласу класу A.

```
// Простий приклад спадкування.
class A {
      int i, j;
      void showij(){
            System.out.println("i та j:"+i+" " + j);
      }
}
// Створення підкласу за рахунок розширення класу A.
class B extends A {
      int k;
      void showk() {
            System.out.println("k: " + k);
      }
      void sum() {
            System.out.println("i+j+k:"+(i+j+k));
      }
}
```

Порядок виклику конструкторів

В ієрархії класів конструктори викликаються в порядку спадкування, починаючи із суперкласу і закінчуючи підкласом. Більше того, оскільки метод ***super ()*** повинен бути першим оператором, виконуваним у конструкторі підкласу, цей порядок залишається незмінним, незалежно від того, чи використовується форма ***super ().***

*Я*кщо метод ***super()*** не застосовується, програма використовує конструктор кожного суперкласу, заданий за замовчуванням або не утримуючий параметрів. У наступній програмі демонструється порядок виконання конструкторів.

```
// Демонстрація порядку виклику конструкторів. x/ Створення суперкласу.
class A {
      A(){
            System.out.printin("Усередині конструктора A.");
      }
}
// Створення підкласу за рахунок розширення класу A.
class B extends A {
      B(){
            System.out.printin("Усередині конструктора B.");
      }
}
// Створення ще одного підкласу за рахунок розширення класу B.
class C extends B {
      C()  {
            System.out.println("Усередині конструктора C.");
      }
}
```

Методичні вказівки до виконання лабораторної роботи:

– Використовуючи результати лабораторної роботи № 2:

- створити клас консольного виведення,
- створити клас виведення у вікно, розширивши попередній клас,
- створити кінцевий клас, розширивши попередній. У результаті буде отриманий клас без єдиного методу, але, що зберігає функціональність двох класів.

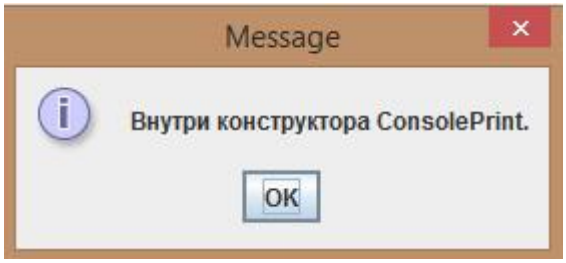
– За допомогою кінцевого класу вивести на консоль і у вікно повідомлень значень різних типів даних, аналогічно завданню лабораторної роботи № 2.

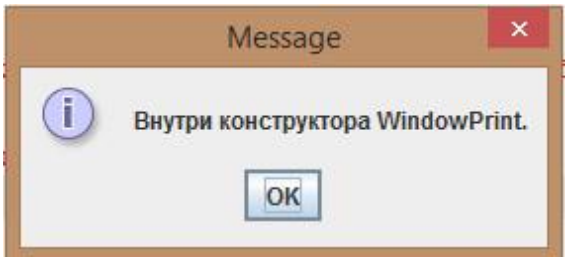
– У кожний клас включити конструктор, у якому здійснити виведення повідомлення про конструктор класу.

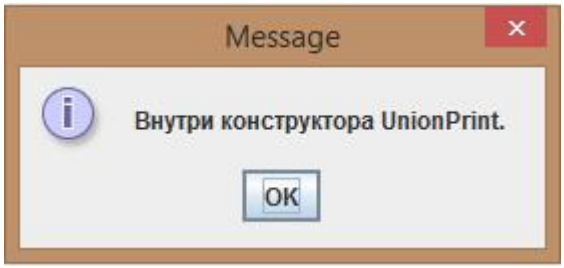
– Виконати програму

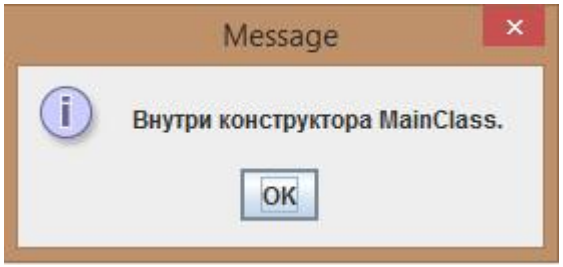
Результат виконання програми:

Повідомлення конструкторів класів:

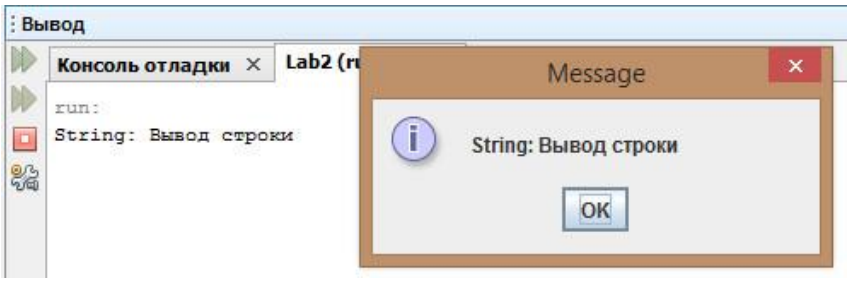




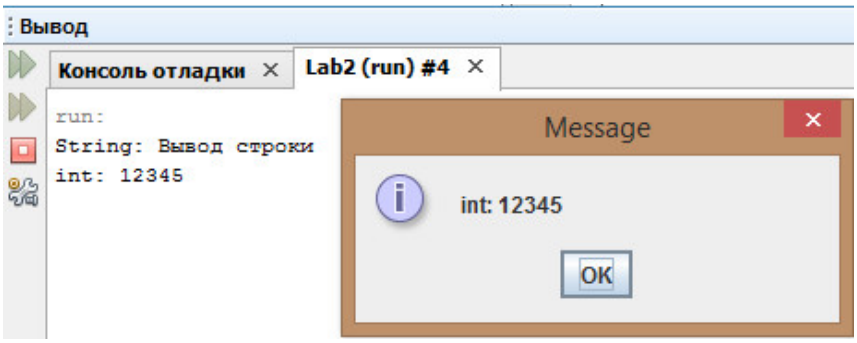




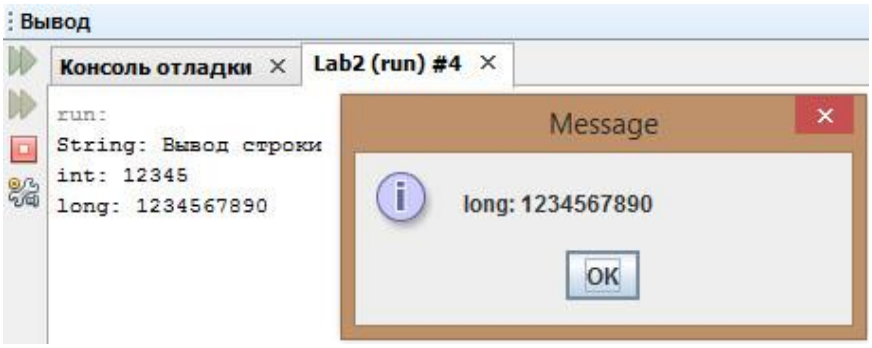
Виведення рядка:



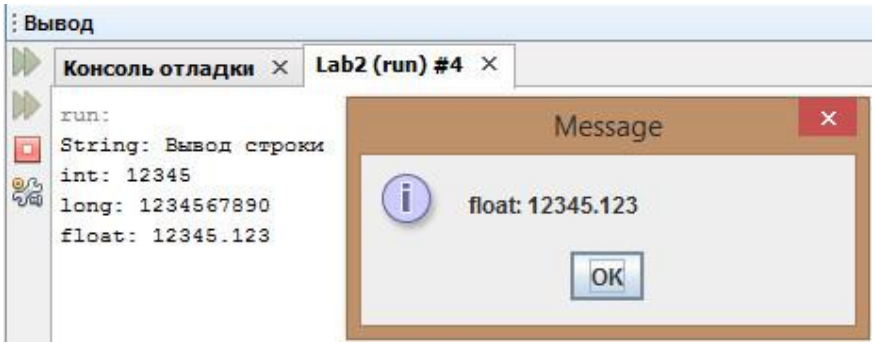
Виведення значення int:



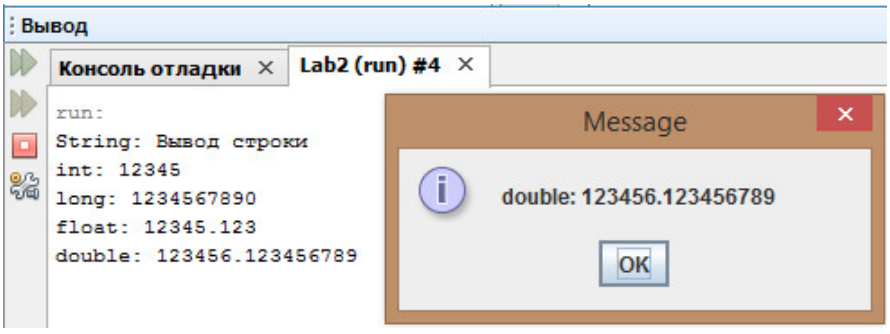
Виведення значення long:



Виведення значення float:



Виведення значення double:



Контрольні питання:

1. Порядок створення і спадкування класів. Обмеження.
2. Призначення і порядок створення конструкторів класу.
3. Призначення ключового слова super.
4. Порядок доступу до членів класу при розширенні класу.
5. Як називається батьківський і наслідуваний клас в Java.

Зміст звіту:

У звіті повинні бути представлені:

- блок-схема алгоритму роботи програми.
- лістинг програми.
- висновки за результатами роботи.

Лабораторна робота № 4

Тема: Бібліотека Swing - створення додатків і обробка подій

Ціль роботи

Одержання навичок роботи з графічним інтерфейсом користувача в середовищі бібліотеки Swing

Завдання:

– Намалювати структурну схему програми.

– Створити проект Java Swing.

– Створити форму, до складу якої включити:

– Кнопки арифметичних операцій, кнопку Exit, два поля введення чисел, область для виведення повідомлень і мітки, що містять написи.

– Написати програму, що реалізує введення двох чисел, обчислення арифметичних операцій із цими числами і виведення результату в область виведення.

– В область виведення необхідно виводити всі повідомлення про події форми і її компонентів.

Теоретичні відомості:

Оскільки дана програма є подійно-орієнтованою, то описати її роботу у вигляді алгоритму не представляється можливим.

Алгоритмом можна описати процедурно-орієнтовану програму або окремий метод програми, але ніяк не подійно-орієнтовану систему.

Подійно-орієнтована система характеризується наступними блоками:

– Подіями.

– Станами.

– Функціями, що виконуються в кожному зі станів.

Бібліотека Swing – це набір класів, які пропонують потужні і гнучкі компоненти GUI. Сучасний графічний користувацький інтерфейс Java побудований на основі бібліотеки Swing, що використовує подійно-орієнтовану технологію.

Архітектура MVC

У загальному випадку візуальний компонент визначається трьома окремими аспектами:

- як компонент виглядає під час його візуалізації на екрані;
- як компонент взаємодіє з користувачем;
- інформацією про стан компонента.

Незалежно від того, яка архітектура використовується для реалізації компонента, вона повинна неявно включати ці три аспекти. Свою виняткову ефективність довела архітектура *"моделъ-представлення-контролер*" (Model-View-Controller – MVC).

Успіх архітектури MVC пояснюється тим, що кожний елемент дизайну відповідає деякому аспекту компонента.

Виходячи з термінології MVC, ***модель*** *(дані)* відповідає інформації про стан, пов'язану з компонентом. Наприклад, у випадку прапорця модель містить поле, яке показує, чи встановлений прапорець.

Представлення визначає, як компонент відображається на екрані, включаючи будь-які аспекти представлення, що залежать від поточного стану моделі.

Контролер визначає, як компонент буде реагувати на дії користувача. Наприклад, коли користувач клацає на прапорці, контролер реагує зміною моделі, щоб відбити вибір користувача (оцінка прапорця або зняття оцінки).

Компоненти і контейнери

Графічний користувацький інтерфейс бібліотеки Swing складається із двох ключових елементів: компонентів і контейнерів.

Компонент являє собою незалежний візуальний елемент керування начебто кнопки або повзунка.

Контейнер уміщає групу компонентів.

Усі компоненти бібліотеки Swing представлені класами, визначеними в пакеті ***javax.swing***. Нижче перераховані імена класів компонентів бібліотеки Swing (включаючи компоненти, використовувані в якості контейнерів).

JApplet	***JButton***	***JCheckBox***	***JCheckBoxMenuItem***
JColorChooser	***JComboBox***	***JComponent***	***JDesktopPane***
JDialog	***JEditorPane***	***JFileChooser***	***JFormattedTextField***
JFrame	***JInternalFrame***	***JLabel***	***JLayer***
JLayeredPane	***JList***	***JMenu***	***JMenuBar***
JMenuItem	***JOptionPane***	***JPanel***	***JPassWordField***
JPopupMenu	***JProgressBar***	***JRadioButton***	***JRadioButtonMenuItem***
JRootPane	***JScrollBar***	***JScrolPane***	***JSeparator***
JSlider	***JSpinner***	***JSplitPane***	***JTabbedPane***
JTable	***JtextArea***	***JTextField***	***JTextPane***
JTogglebutton	***JToolBar***	***JToolTip***	***JTree***
JViewport	***JWindow***		

Контейнери

У бібліотеці Swing визначено два типи контейнерів. Перший тип-це контейнери верхнього рівня. До них відносяться контейнери класів ***JFrame***, ***JApplet***, ***JWindow*** і ***JDialog.*** Класи цих контейнерів не є спадкоємцями класу ***JComponent.***

Однак вони успадковують класи бібліотеки AWT ***Component*** і ***Container.***

Простий додаток Swing

У додатку використовується два компоненти бібліотеки Swing: ***JFrame*** і ***JLabel***.

Клас ***JFrame*** – контейнер верхнього рівня, який зазвичай використовується в додатках Swing. Клас ***JLabel*** – компонент бібліотеки Swing, що створює мітку, яка є компонентом, що відображає інформацію.

Мітка – це найпростіший компонент бібліотеки Swing, оскільки вона є пасивним компонентом. Тобто мітка не реагує на дії користувача. Вона служить для відображення вихідних даних.

Програма використовує контейнер класу ***JFrame*** для зберігання екземпляра класу ***JLabel***. Мітка відображає коротке текстове повідомлення.

```
// Простий додаток Swing
import javax.swing.*;
class SwingDemo {
      SwingDemo()      {
            // Створення нового контейнера JFrame.
            JFrame jfrm = new     JFrame("Простий додаток Swing");
            // Задаємо фрейму вихідний розмір,
            jfrm.setSize(275, 100);
            // Припиняємо роботу програми, якщо користувач
            // закриває додаток.
            jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
            // Створюємо мітку з текстом.
            JLabel jlab = new JLabel("Hello Swing!");
            // Додаємо мітку на панель вмісту,
            jfrm.add(jlab);
            // Відображаємо фрейм,
            jfrm.setVisible(true);
      }
      public static void main(String args[])     {
            // Створюємо фрейм у потоці диспетчеризації подій.
            Swingutllities.invokeLater(new Runnable()  {
                  public void run() {
                        new SwingDemo();
                  }
            }) ;
      }
}
```

Спочатку оголошується клас ***SwingDemo*** і конструктор для цього класу. Конструктор починається зі створення екземпляра класу ***JFrame*** за допомогою наступного рядка коду.

```
JFrame jfrm = new JFrame("Простий додаток Swing");
```

У результаті буде створений контейнер ***jfrm,*** *що* визначає прямокутне вікно з рядком заголовка, кнопками закриття, згортання, розгортання і відновлення, а також із системним меню. Таким чином, програма створює стандартне вікно верхнього рівня. Конструктору передається заголовок вікна. Потім задаються розміри вікна за допомогою наступного оператора.

```
jfrm.setSize(275, 100);
```

Метод ***setSize ()*** задає розміри вікна, обумовлені в пікселях. Він має таку форму.

```
void setSize(int ширина, int висота)
```

У цьому прикладі ширина вікна рівняється 275 пікселям, а висота -100.

За замовчуванням, коли закривається вікно верхнього рівня (наприклад, коли користувач клацає на кнопці закриття), вікно видаляється з екрана, але робота додатка не припиняється.

Тому, при закритті вікна верхнього рівня потрібно припиняти роботу додатка шляхом виклику методу ***setDefaultCloseOperation (),*** що і робиться в програмі.

```
jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Тепер робота всього додатки буде припинятися при закритті вікна. Загальна

форма методу ***setDefaultCloseOperation ()*** виглядає так.

```
void setDefaultCloseOperation(int app)
```

Значення параметра *app* визначає, що відбувається при закритті вікна. Крім значення ***JFrame . EXIT__ON_CLOSE***, доступно ще кілька значень.

```
DISPOSE_ON_CLOSE

HIDE_ON_CLOSE

DO_NOTHING_ON_CLOSE
```

У їхніх іменах відбиті виконувані дії. Ці константи оголошені в класі ***Windowconstants,*** який є інтерфейсом, оголошеним у пакеті ***javax.swing***, реалізованим контейнером класу ***JFrame.***

```
JLabel jlab = new JLabel("Hello Swing!");
```

Клас ***JLabel*** – найпростіший і легкий у використанні компонент, тому що він не приймає від користувача вхідних даних. Він просто відображає інформацію, яка може представляти текст, значок або їх комбінацію. Мітка, створена програмою, містить тільки текст, який передається її конструктору.

Наступний рядок коду додає мітку в панель вмісту фрейму.

```
jfrm.add(jlab);
```

Останній оператор у конструкторі класу ***SwingDemo*** потрібний для того, щоб зробити вікно видимим.

```
jfrm.setVisible(true);
```

Метод ***setVisible()*** успадковується від класу бібліотеки AWT ***Component***. Якщо його аргумент буде рівний ***true***, то вікно буде відображатися. А якщо ні, то воно буде приховано. За замовчуванням контейнер класу ***JFrame*** є невидимим, тому, щоб показати його, потрібно викликати метод ***setVisible (true).***

Усередині методу ***main ()*** створюється об'єкт класу ***SwingDemo,*** який відображає вікно і мітку. Зверніть увагу на те, що конструктор класу ***SwingDemo*** викликається за допомогою наступних трьох рядків коду.

```
Swingutilities.invokeLater(new Runnable()  {
     public void run(){
          new SwingDemo();
     }
}) ;
```

Виконання цієї послідовності коду приводить до створення об'єкта класу ***SwingDemo*** у ***потоці диспетчеризації подій***, а ***не в головному потоці додатка***.

У загальному випадку програми Swing управляються подіями. Наприклад, коли користувач взаємодіє з компонентом, відбувається подія.

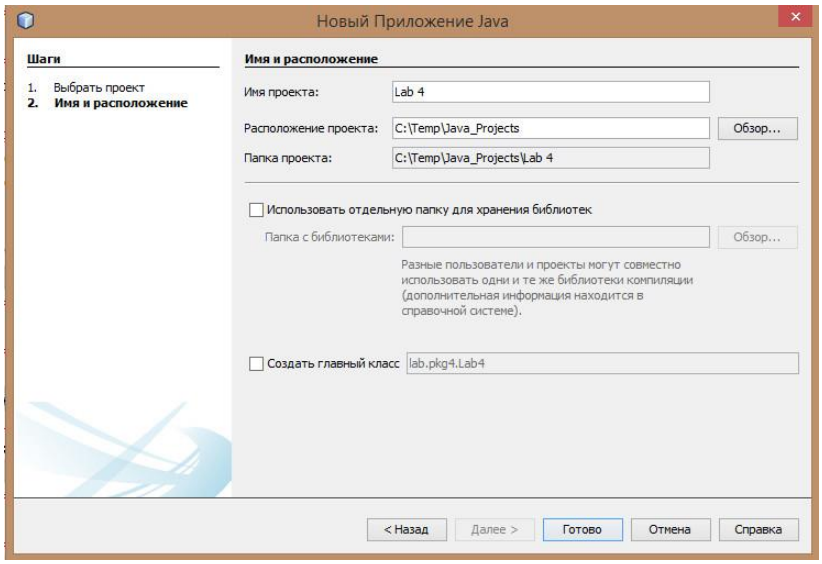
Повідомлення про подію передається в додаток викликом оброблювача подій, визначеного в додатку.

Однак оброблювач виконується в потоці диспетчеризації подій, що підтримується бібліотекою Swing, а не в головному потоці додатка. Таким чином, хоча оброблювачі подій і визначені в програмі, вони викликаються в потоці, який не був створений вашою програмою.

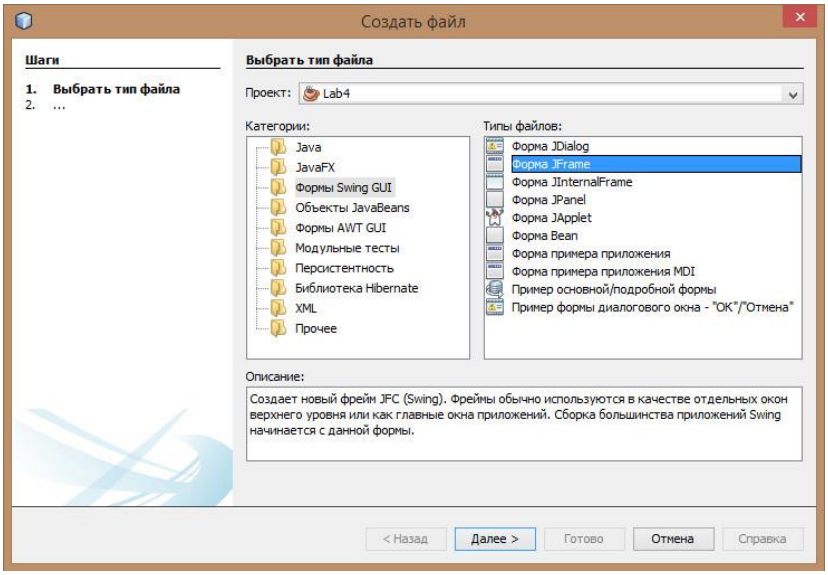
Щоб уникнути цієї проблеми (включаючи ймовірність виникнення взаємного блокування), ***усі компоненти GUI бібліотеки Swing потрібно створювати і обновлять з потоку диспетчеризації подій, а не з головного потоку додатка***.

Методичні вказівки до виконання лабораторної роботи:

Створити порожній проект, що не містить класу з функцією main():



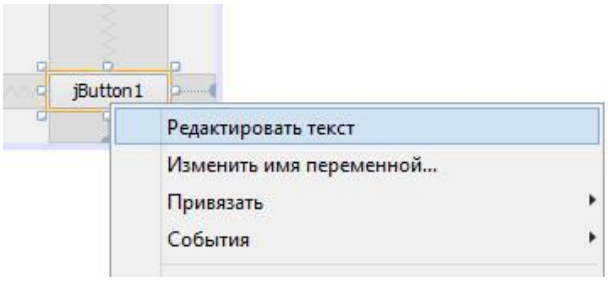
Створити клас форми:



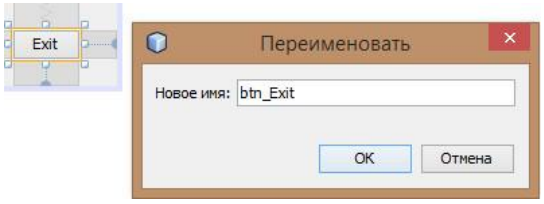
Створити кнопки, мітки, поля введення і виведення, перетаскуючи їх із правої панелі на форму.

Створення кнопки:

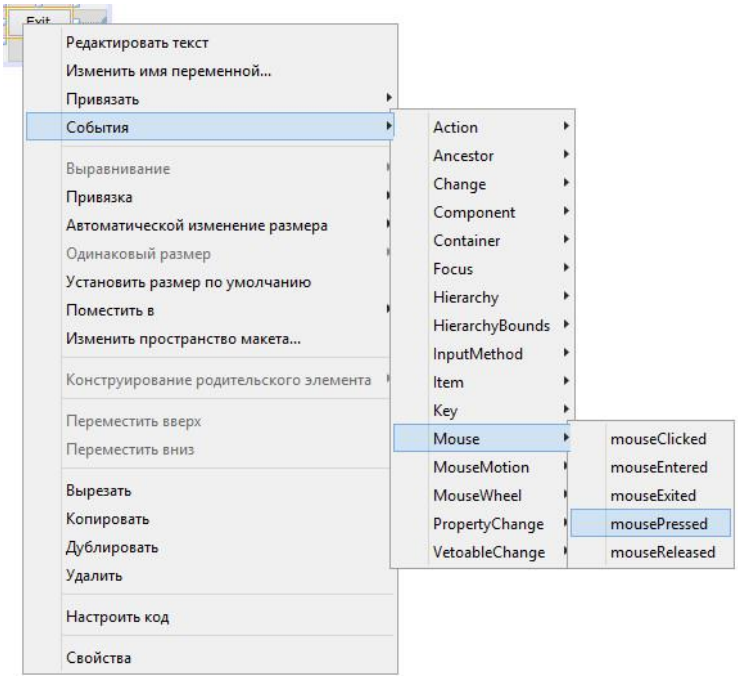
Зміни тексту на кнопці здійснюється через випадаюче меню при натисканні правої кнопки миші, знаходячись на об'єкті:



Зміна імені об'єкта:



Для об'єкта форми необхідно визначити подію, яка об'єкт буде генерувати та оброблювач цієї події:



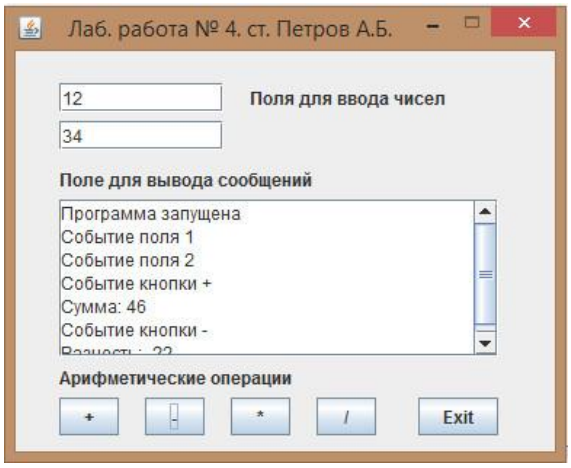
Наприклад, для кнопки Exit визначена подія миші MousePressed, яке виникне, якщо на об'єкті нажати кнопку миші.

Оброблювач події буде створений середовищем JavaBeans.

Далі, привести в нормальний вигляд код, створений середовищем JavaBeans:

```
//==============================================================
// Лабораторная работа № 4
// class: JFrame_Lab4
// Copyright (c) Петров А.В., гр. КІ-11
//==============================================================
package lab4;
//==============================================================
// JFrame_Lab4
//==============================================================
public class JFrame_Lab4 extends javax.swing.JFrame {
    //======конструктор
    public JFrame_Lab4() {
        initComponents();
    }
    @SuppressWarnings("unchecked")
    Generated Code
//==============================================================
// Обработчики событий
//==============================================================
    //==== Кнопка Exit
    private void btn_ExitMousePressed(java.awt.event.MouseEvent evt) {
        // TODO add your handling code here:
        System.exit(0);
    }
//==============================================================
// Создание формы
//==============================================================
   public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new JFrame_Lab4().setVisible(true);
            }
        });
    }
   //===== компоненты формы
    // Variables declaration - do not modify
    private javax.swing.JButton btn_Exit;
    // End of variables declaration
}
```

І продовжити побудову форми. Зразковий вигляд форми:



Потім визначити необхідні події для об'єктів форми і їх оброблювачі. В оброблювачах реалізувати функціонал програми. Для цього використовувати інший клас, у якому повинен утримуватися основний код програми.

Скласти діаграму процесів програми.

Приклад виконання лабораторної роботи представлений файлом Lab4.jar

Контрольні питання:

1. Порядок створення елементів керування і призначення подій.
2. Оброблювачі подій, призначення.
3. Класи подій.
4. Контейнери і компоненти. Їхня відмінність і особливості
5. Призначення операторів у функції main() форми.

Зміст звіту:

У звіті повинні бути представлені:

- Структурна схема програми
- Діаграма процесів програми
- Лістинг програми.
- висновки за результатами роботи.

Лабораторна робота № 5

Тема: Робота з елементами керування Swing

Ціль роботи

Одержання навичок створення, програмування і роботи з елементами керування GUI у середовищі бібліотеки Swing

Завдання:

- Намалювати структурну схему програми.
- Створити проект Java Swing.
- Створити форму, до складу якої включити:

Кнопку Exit, два поля введення чисел, область для виведення повідомлень, елементи керування:

1. JCheckBox,
2. JComboBox,
3. JList,
4. JRadioButton,
5. JTabbedPane

- Написати програму, що реалізує введення двох чисел, обчислення арифметичних операцій із цими числами і виведення результату в область виведення, використовуючи перераховані елементи керування.

Теоретичні відомості:

У бібліотеці Swing визначено чотири класи кнопок – ***JButton, JToggleButton, JCheckBox*** і ***JRadioButton.*** Усі вони є підкласами класу ***AbstractButton***, який розширює клас ***JComponent***. Таким чином, у кнопок є загальні характерні риси.

```
void setDisabledIcon(Icon di)
void setPressedIcon(Icon pi)
void setSelectedIcon(Icon si)
void setRolloverIcon(Icon ri)
```

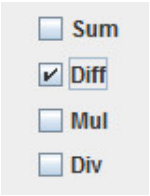
Параметри *di*, *pi*, *si* и *ri* визначають значки, використовувані для різних станів. Текст, пов'язаний з кнопкою, можна прочитати і записати за допомогою наступних методів.

```
String getText()
void setText(String рядок)
```

Тут *рядок* – це текст, який буде пов'язаний з кнопкою.

Модель, використовувана всіма кнопками, визначена інтерфейсом ***ButtonModel***. Кнопка сповіщає про подію дії, коли її натискає користувач.

Прапорці. Клас *JcheckBox*



Клас ***JCheckBox*** визначає функції прапорця. Клас ***JCheckBox*** визначає декілька конструкторів. Один з них виглядає у такий спосіб.

```
JCheckBox(String рядок)
```

Цей конструктор створює прапорець з текстом, визначеним за допомогою параметра рядок в якості мітки. Решта конструкторів дозволяють визначити вихідний стан вибору кнопки і задати значок.

Якщо користувач встановлює або скидає прапорець, створюється повідомлення про подію класу ***ItemEvent***. Визначити обраний стан простіше всього, викликавши метод ***isSelected()*** в екземплярі класу ***JCheckBox***.

Перемикачі. Клас *JradioButton*

Перемикачі являють собою групи взаємовиключних кнопок, у яких можна вибрати тільки одну. Клас ***JRadioButton*** пропонує декілька конструкторів. Один з них показаний нижче.

```
JRadioButton(String рядок)
```

Тут *рядок* – це мітка кнопки. Решта конструкторів дозволяють визначити вихідний стан кнопки і задати значок.

Перемикачі необхідно поєднувати в групу, у якій можна вибрати тільки одну із кнопок. Наприклад, якщо користувач вибирає перемикач, що знаходиться в групі, то обраний раніше перемикач у цій групі автоматично відключається.

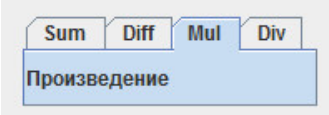
Для створення групи кнопок призначений клас ***ButtonGroup***. Для цієї мети викликається його конструктор, використовуваний за замовчуванням. Після цього можна додати в групу елементи за допомогою наступного методу.

```
void add(AbstractButton ab)
```

Тут *ab* представляє посилання на кнопку, яку необхідно додати в групу.

Клас ***JRadioButton*** створює повідомлення про подію події змін кожного разу, коли міняється обрана кнопка в перемикачі.

Клас JtabbedPane



Клас ***JTabbedPane*** визначає *панель з вкладками*. Він управляє набором компонентів, з'єднуючи їх за допомогою вкладок. При виборі вкладки компонент, пов'язаний з нею, з'являється на передньому плані. Панелі з вкладками дуже популярні в сучасних користувацьких графічних інтерфейсах.

Клас ***JTabbedPane*** визначає три конструктори. Вкладки додаються за допомогою виклику методу ***addTab()***. Нижче показана одна з його форм.

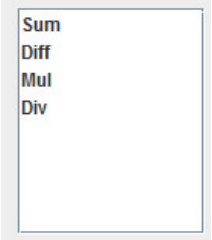
```
void addTab(String ім'я, Component comp)
```

Тут *ім'я* – це ім'я вкладки, а *comp* – це компонент, який повинна містити вкладка. Зазвичай додається компонент ***JPanel***, що містить групу зв'язаних між собою компонентів. Ця технологія дозволяє вкладці містити набір компонентів.

Загальна процедура використання панелі з вкладками виглядає в такий спосіб.

- Створюється об'єкт класу ***JTabbedPane.***
- Для додавання кожної вкладки потрібно викликати метод `addTab()`.
- Панель із вкладками переноситься в панель вмісту.

Клас JList



Базовим класом списків у бібліотеці Swing є клас ***JList***. Він підтримує вибір одного або декількох елементів зі списку. Найчастіше список складається з рядків, але можна створити список з будь-яких об'єктів, які тільки можна відобразити.

Клас ***JList*** оголошується так.

```
class JList <E>
```

Тут параметр E представляє тип елементів у списку.

Клас ***JList*** пропонує декілька конструкторів. Один з найбільш використовуваних конструкторів виглядає так.

```
JList (E [ ] елементи)
```

Цей конструктор створює список класу ***JList***, що містить елементи в масиві, що визначаються за допомогою параметра *елементи.*

Хоча клас ***JList*** здатний працювати самостійно, зазвичай його екземпляр вкладають у контейнер класу ***JScrollPane***. Завдяки цьому довгі списки автоматично прокручуються, що спрощує проектування графічного інтерфейсу. Це дозволить також спростити зміну кількості записів у списку, не змінюючи його розміри.

Клас ***JList*** створює повідомлення про подію ***ListSelectionEvent***, коли користувач вибирає елемент або змінює вибір елемента. Ця подія відбувається також тоді, коли користувач скасовує вибір елемента. Вона обробляється слухачем ***ListSelectionListener***, який визначає тільки один метод – ***valueChanged ()***.

```
void valueChanged(ListSelectionEvent le)
```

Тут параметр *le* – це посилання на об'єкт, який створив повідомлення про подію.

Подія ***ListSelectionEvent*** і слухач ***ListSelectionListener*** визначені в пакеті `javax.swing.event`.

За замовчуванням клас ***JList*** дозволяє користувачу вибирати кілька діапазонів елементів усередині списку, однак можна змінити ця поведінку, викликавши метод ***setSelectionModel()***, який визначений у класі ***JList***. Він показаний нижче.

```
void setSelectionMode(int режим)
```

Тут *режим* – це режим вибору. Він повинен бути представлений одним з наступних значень, визначених інтерфейсом ***ListSelectionModel***.

```
SINGLE_SELECTION
SINGLE_INTERVAL_SELECTION
MULTIPLE_INTERVAL_SELECTION
```

За замовчуванням використовується останнє значення, яке дозволяє користувачеві вибирати безліч діапазонів елементів усередині списку. Якщо буде заданий вибір в одиничному інтервалі `(SINGLE_INTERVAL_SELECTION)`, користувач зможе вибрати тільки один діапазон елементів.

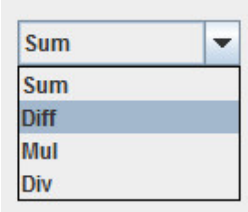
Якщо буде обране значення `SINGLE_SELECTION`, користувач зможе вибрати тільки один елемент.

Індексація починається з нуля. Тому, якщо буде обраний перший елемент, метод поверне значення 0. Якщо не буде обрано жодного елемента, буде повернуте значення -1.

Замість того щоб одержувати індекс обраного елемента, ви можете одержати значення, пов'язане з обраним елементом, викликавши метод ***getSelectedValue()***.

Метод повертає посилання на перше обране значення. Якщо не буде обрано жодного значення, він поверне значення ***null***.

Клас JComboBox



За допомогою класу ***JComboBox*** бібліотеки Swing визначається компонент, що називається комбінованим списком (комбінація текстового поля списку, що розкривається).

Як правило, комбінований список відображає один запис, але він буде також відображати список, що розкривається, дозволяє вибирати інші елементи.

Нижче показаний конструктор класу ***JComboBox***, використовуваний у цьому прикладі.

```
JComboBox (E[ ] елементи)
```

Тут *елементи* – це масив, що ініціалізує комбінований список. Крім нього, доступні також і інші конструктори.

Клас ***JComboBox*** використовує модель ***ComboBoxModel***. Змінювані комбіновані списки (списки, елементи яких можуть змінюватися) використовують модель ***MutableComboBoxModel***.

Крім передачі масиву елементів, які повинні бути відображені в списку, що розкривається, елементи можна додавати динамічно в список за допомогою методу ***addItem(),***, показаного нижче.

```
void addItem(E об'єкт)
```

Тут *об'єкт* – це об'єкт, який необхідно додати в комбінований список. Цей метод повинен використовуватися тільки в змінюваних комбінованих списках.

Клас ***JComboBox*** створює повідомлення про подію дії, коли користувач вибирає елемент зі списку.

Довідатися, який елемент списку був обраний, можна за допомогою методу ***getSelectedItem().***

```
Object getSelectedItem()
```

Потрібно буде привести повернуте значення до типу об'єкта, що зберігається в списку.

Класи подій для елементів керування:

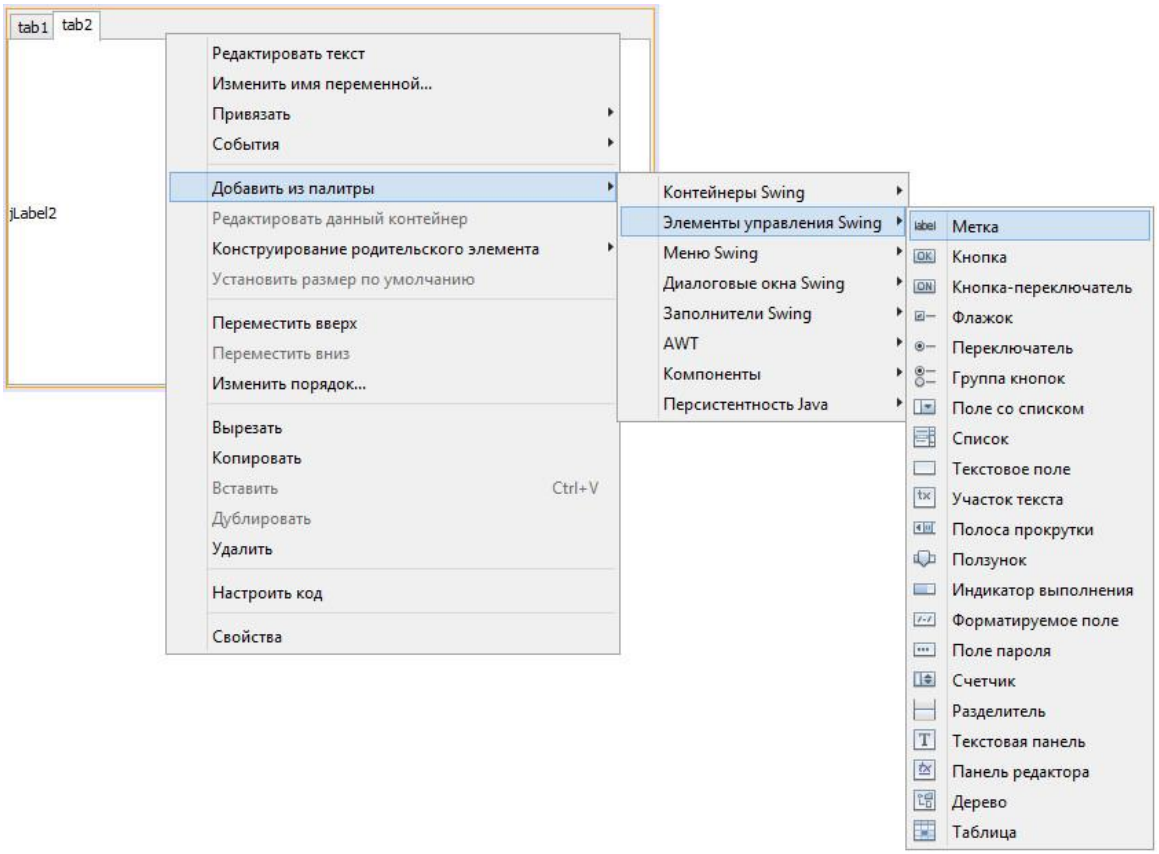
– ActionPerformed
– MousePressed

Методичні вказівки до виконання лабораторної роботи:

При створенні елемента керування JRadioButton, необхідно всі елементи включити в контейнер, який попередньо помістити у форму:

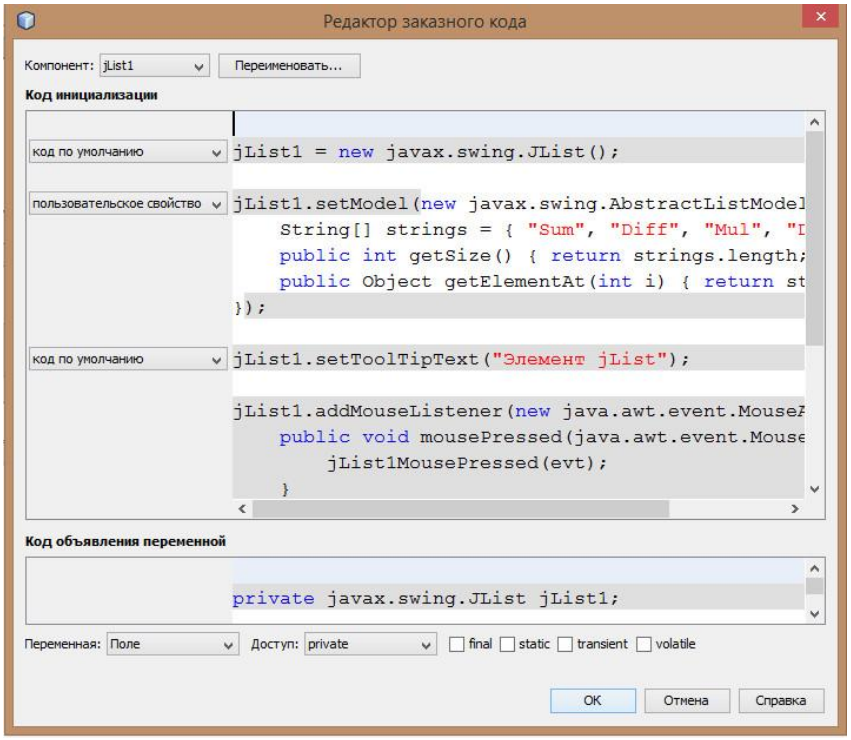
```
buttonGroup1.add(jRadioButton1);
buttonGroup1.add(jRadioButton2);
buttonGroup1.add(jRadioButton3);
buttonGroup1.add(jRadioButton4);
```

Елементи на панель вкладок додаються в такий спосіб:

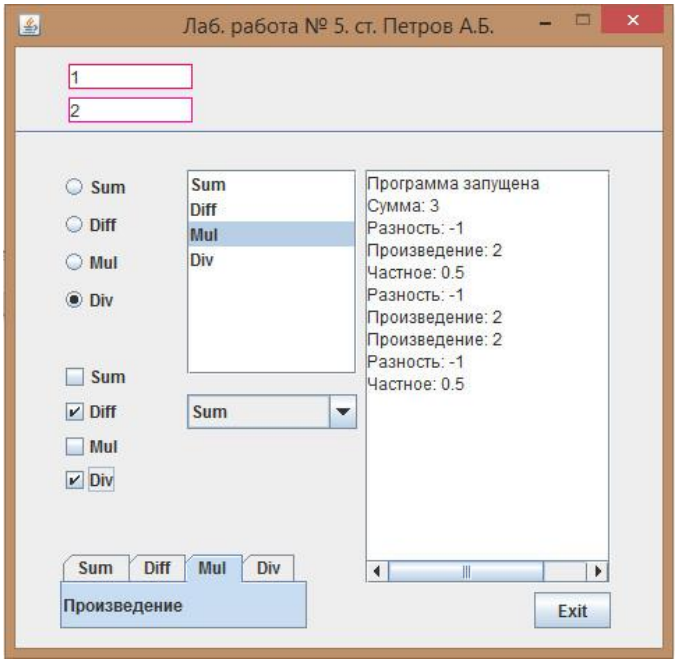


А потім редагуються властивості елемента.

Редагування коду, створеного IDE JavaBeans здійснюється за допомогою вкладки “Настроїти код”:



Рекомендований вид форми для лабораторної роботи:



Контрольні питання:

1. Призначення, створення і використання елемента керування jCheckBox.
2. Призначення, створення і використання елемента керування jList
3. Призначення, створення і використання елемента керування jRadioButton
4. Призначення, створення і використання елемента керування jTabbedPane
5. Призначення, створення і використання елемента керування jComboBox

Зміст звіту:

У звіті повинні бути представлені:

- Структурна схема програми
- Діаграма станів програми
- Лістинг програми.
- Висновки за результатами роботи.

Лабораторна робота № 6

Тема: Робота із класами JMenu, JProgressBar, JSlider і JSpinner

Ціль роботи

Одержання навичок створення, програмування і роботи з елементами керування GUI у середовищі бібліотеки Swing

Завдання:

- Намалювати структурну схему програми.
- Створити проект Java Swing.
- Створити форму, до складу якої включити:

Кнопку Exit, два поля введення чисел, область для виведення повідомлень, елементи керування:

1. JMenu,
2. JProgressBar,
3. JSlider,
4. JSpinner,
5. JLabel.

- Написати програму, що реалізує введення двох чисел, с допомогою елементів `JSlider` і `JSpinner`. Здійснити обчислення арифметичних операцій із цими числами і виведення результату в область виведення і елемент керування `JLabel`, використовуючи елемент керування `JMenu`.

Теоретичні відомості:

Лічильник JSpinner

Перебирати елементи списку часто буває зручніше за допомогою лічильника - невеликого поля, що редагується, з поточним значенням списку і двома стрілками нагору і вниз, за допомогою яких можна заміняти поточне значення попереднім або наступним.

Найпростіший лічильник створюється конструктором за замовчуванням `JSpinner()`. Його поточне значення – 0, наступне – 1, попереднє – -1, і так можна перебирати всі цілі числа.

Для створення більш складного лічильника конструктором `JSpinner(SpinnerModel)` доведеться спочатку визначати модель даних.

Вона описана інтерфейсом `SpinnerModel`, частково реалізована абстрактним класом `AbstractSpinnerModel` і повністю реалізована в трьох класах.

Клас `SpinnerDateModel` реалізує модель, що містить дати,

Клас `SpinnerListModel` – модель, що містить колекцію типу `List`, зокрема, масив довільних об'єктів типу `Object[]`.

Клас `SpinnerNumberModel` містить цілі або дійсні числа або об'єкти класу `Number`.

Наприклад, наступний рядок:

```
JSpinner sp = new JSpinner(new SpinnerNumberModel(50, 0, 100, 5));
```

створює лічильник з поточним значенням 50, діапазоном значень від 0 до 100 і кроком зміни значень 5.

У поле можна ввести будь-яке значення з зазначеного діапазону, наприклад 47, тоді попереднє значення, буде рівно 42, а наступне – 52.

За допомогою класу `SpinnerNumberModel` можна створити ще модель з дійсними числами конструктором

```
SpinnerNumberModel(double current, double min, double max, double step);
```

і числову модель загального вигляду конструктором

```
SpinnerNumberModel(Number current, Comparable min, Comparable max, Number step);
```

Значення min і max можуть бути `null`, у такому випадку нижня або верхня границі не існують.

Поточне, попереднє і наступне значення можна одержати від лічильника `JSpinner` методами

```
getValue(),
getPreviousValue() і
getNextValue() відповідно.
```

Ці методи повертають об'єкт класу `Object`. Якщо значення виходять за задані в конструкторі границі, то зазначені методи повертають `null`.

При всякій зміні поточного значення, а також закінченні введення в поле нового значення шляхом натискання клавіші `<Enter>`, у лічильнику відбувається подія класу `ChangeEvent`.

Друга модель даних класу, `SpinnerDateModel`, дозволяє вибирати дати із заданого списку. Конструктор за замовчуванням `SpinnerDateModel()` створює лічильник з поточною датою і часом, попереднє його значення - це той же час вчорашнього дня, таке значення - той же час завтрашнього дня.

Конструктор

```
SpinnerDateModel(Date value, Comparable first, Comparable last, Date step);
```

задає довільну поточну дату `value`, діапазон значень дат від `first` до `last` і крок `step`. Якщо одне або обидва значення діапазону рівні `null`, то відповідна границя відсутня.

Крок `step` визначає також і форму представлення дати і може приймати значення однієї з констант:

```
ERA,
YEAR,
MONTH,
WEEK_OF_YEAR,
WEEK_OF_MONTH,
DAY_OF_MONTH,
DAY_OF_YEAR,
```

```
DAY_OF_WEEK,
DAY_OF_WEEK_IN_MONTH,
AM_PM,
HOUR,
HOUR_OF_DAY,
MINUTE,
SECOND,
MILLISECOND
```

класу `Calendar`.

Більш широкі можливості надає третя модель даних класу – `SpinnerListModel`.

У конструкторі `SpinnerListModel(Object[])` цього класу можна задати масив довільних об'єктів, наприклад:

```
String[] data = {"Дворник", "Прибиральниця", "Програміст", "Сторож"};
SpinnerListModel model = new SpinnerListModel(data);
JSpinner emp = new JSpinner(model);
```

В іншому конструкторові, `SpinnerListModel(List)`, задається екземпляр колекції, що реалізує інтерфейс `List`, наприклад екземпляр класу `Vector`.

Хоча в лічильник можна закласти будь-які об'єкти, у поле буде показаний тільки рядок, отриманий методом `toString()` поточного об'єкта.

Це відбувається тому, що редактор за замовчуванням, закладений у клас `JSpinner`, – це екземпляр класу `JFormattedTextField`.

Повзунок JSlider

Повзунок являє собою лінійку з покажчиком, яким можна встановити якесь значення value з діапазону min-max. Внутрішньо повзунок влаштований так само, як і смуга прокручування.

Компонент JSlider використовує модель даних класу `DefaultBoundedRangeModel` з найменшим, найбільшим і поточним значеннями, а також кроком зміни.

Втім, можна застосувати іншу модель, задавши її в конструкторі `JSlider(BoundedRangeModel)` або встановивши методом `setModel(BoundedRangeModel)`.

Основний конструктор:

```
JSlider(int orientation, int min, int max, int value);
```

В інших конструкторах відсутній той або інший параметр, при цьому встановлюється горизонтальний повзунок зі значеннями

```
min = 0, max = 100, value = (min + max)/2.
```

Поруч із лінійкою повзунка можна розмітити шкалу зі штрихами, що відстають один від одного на відстань, яка встановлюється `setMajorTickSpacing(int)`.

Спочатку ця відстань дорівнює нулю. Після визначення відстані шкала задається методом `setPaintTicks(true)`.

До штрихів можна додати числові значення методом `setPaintLabels(true)`.

Між штрихами допускається розміщення більш дрібних штрихів методом `setMinorTickSpacing(int)`.

Якщо застосувати метод `setSnapToTicks(true)`, то движок повзунка буде зупинятися тільки проти штрихів.

Основну лінійку повзунка можна забрати методом `setPaintTrack(false)`, залишивши тільки шкалу.

Числові значення в шкалі ставляться проти кожного штриха.

Методом `createStandardLabels(int incr, int start)` можна змінити це правило, задавши інший крок розміщення чисел `incr` на шкалі та інший початковий відлік `start`.

Потім цей крок треба встановити на шкалі методом `setLabelTable(Dictionary)`. Усе це зручно робити після визначень:

Зовнішній вигляд повзунка визначається абстрактним класом `SliderUI`. У нього два розширення – класи `BasicSliderUI` і `MultiSliderUI`.

При переміщенні движка в повзунку відбувається подія `ChangeEvent`. У процесі обробки цієї події можна одержати значення повзунка методом `getvalue()`.

Індикатор JProgressBar

Індикатор, часто званий "градусником", показує ступінь виконання якогось процесу, найчастіше у відсотках. Процес повинен виробляти яке-небудь ціле число. У конструкторі індикатору

```
JProgressBar(int orientation, int min, int max);
```

задаються найменше min і найбільше max значення цього числа. В інших конструкторах опущені деякі із зазначених параметрів. При цьому орієнтація вважається горизонтальною, `min = 0, max = 100.`

У міру виконання процесу він повинен передавати ступінь свого виконання в індикатор методом `setValue(int)`.

Це значення негайно відбивається в індикаторі.

Після звертання до методу `setStringPainted(true)` у вікні індикатору з'явиться ще число – відсоток виконання процесу.

Якщо час виконання процесу, пов'язаного з індикатором, не визначений, то можна перевести індикатор у *невизначений режим* (indeterminate mode). Це робиться методом `setIndeterminate(true)`.

У цьому режимі індикатор мигає, показуючи, що процес виконується.

Коли закінчення процесу визначиться, треба занести найбільше значення процесу в індикатор методом `setMaximum(int)`, значення, що тече, методом `setValue(int)` і перевести індикатор у звичайний режим методом `setIndeterminate(false)`.

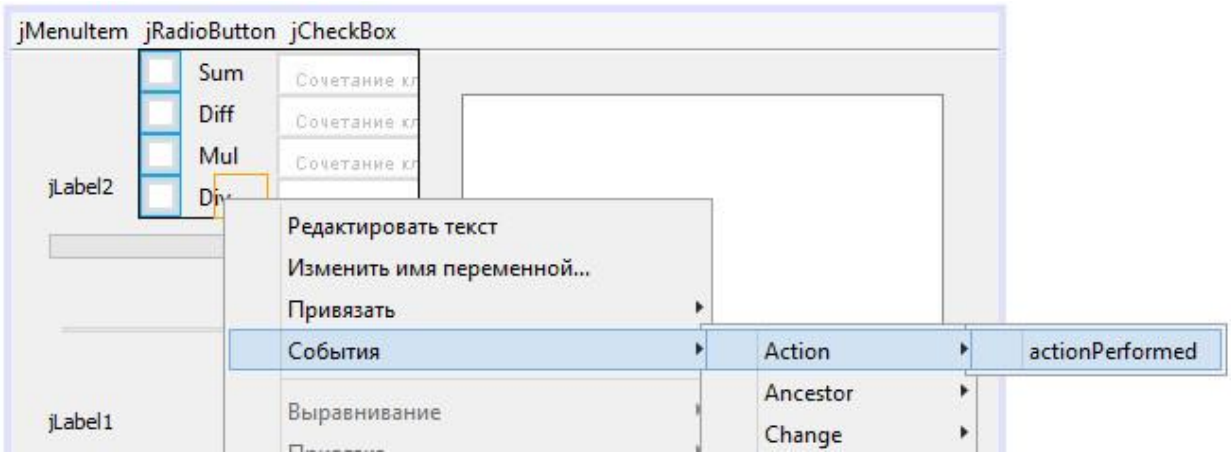
Зовнішній вигляд індикатору описується абстрактним класом `ProgressBarUI`. У нього два розширення – класи `BasicProgressBarUI` і `MultiProgressBarUI`.

Стандартний вигляд Java L&F забезпечується класом `MetalProgressBarUI`. При необхідності зміни зовнішнього вигляду індикатору слід розширити один із цих класів і встановити новий вид методом `setUI(ProgressBarUI)`.

Методичні вказівки до виконання лабораторної роботи:

Створити форму, у яку помістити елементи керування, зазначені в завданні.

Визначити оброблювачі подій для елементів керування:

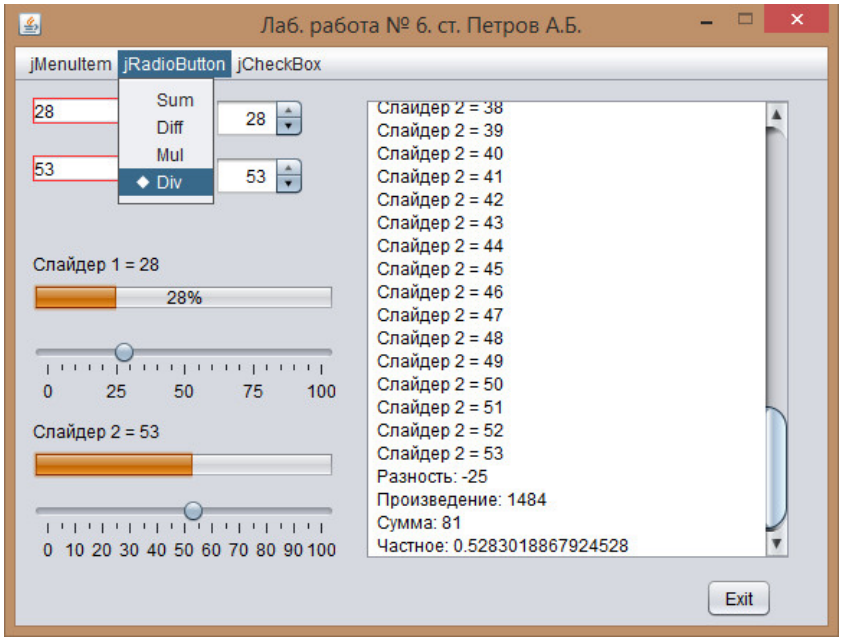


Основним елементом керування є `JSlider`. Прочитати значення `JSlider` і встановити їх в елемент `JProgressBar`, `JLabel`, `JTextField` і `JTextArea`.

Змінюючи стан елемента керування `JSpinner`, змінювати стан елемента `JProgressBar`.

Вибираючи відповідний пункт меню, виконати арифметичні операції із числами в полях елементів `JTextField`.

Рекомендований вигляд форми :



Контрольні питання:

- Призначення, створення і використання елемента керування JMenu.
- Призначення, створення і використання елемента керування JProgressBar.
- Призначення, створення і використання елемента керування JSlider.
- Призначення, створення і використання елемента керування JSpinner.
- Призначення моделей елементів керування.

Зміст звіту:

У звіті повинні бути представлені:

- Структурна схема програми
- Діаграма станів програми
- Лістинг програми.
- Висновки за результатами роботи.

Лабораторна робота № 7

Тема: Робота з файловою системою. Робота із класом RandomAccessFile

Ціль роботи

Одержання навичок роботи з файлами в режимі довільного доступу до даних

Завдання:

- Намалювати структурну схему програми.
- Створити проект Java Swing.
- Створити форму, до складу якої включити:

Кнопку Exit, поле для введення імені файлу, поле для введення даних, поле для позиції файлового покажчика, область для виведення повідомлень і кнопки для виконання операцій:

1. Створити | Відкрити файл.
2. Установити позицію покажчика у файлі.
3. Додати дані в кінець файлу.
4. Вставити дані в позиції покажчика файлу.
5. Подивитися вміст файлу за допомогою програми Notepad.exe.
6. Прочитати вміст файлу і відобразити в області перегляду.
7. Написати програму, що реалізує файлові операції.

Теоретичні відомості:

Клас `java.io.RandomAccessFile` забезпечує читання і запис даних у будь-якому місці файлу. Відомо, що потоки даних з мережних з'єднань дозволяють одержати дані послідовним шляхом. На відміну від мережних з'єднань, файли, записані на дисках, мають довільний доступ. Даний спосіб читання і запису необхідний у тих випадках, коли потрібно одержати доступ до даних, не читаючи дані з початку файлу. Крім того, файли з довільним доступом можуть відкриватися одночасно по читанню і запису, а також можуть відкриватися тільки по читанню.

У файлі з довільним доступом присутній покажчик файлу (file pointer), який завжди вказує на положення нового запису. Метод `seek` дозволяє довільно встановлювати покажчик на будь-який байт усередині файлу. Аргумент методу `seek` має тип `long` і приймає значення від 0 до максимальної довжини файлу, вираженої в байтах. А за допомогою методу `getFilePointer` можна одержати поточну позицію покажчика файлу.

Найбільш вживані конструктори і методи класу:

`RandomAccessFile(String name, String mode)`

`name` – ім'я файлу, що залежить від системи.

`mode` – режим відкриття файлу, може приймати значення `"r","rw","rws","rwd"`.

`RandomAccessFile(File file, String mode)`

`file` – об'єкт класу File, інкапсулюючий системно-залежне ім'я файлу.

`mode` – режим відкриття файлу, може приймати значення `"r","rw","rws","rwd"`.

Значення режимів відкриття файлу:

`"r"`	Відкриває файл тільки по читанню. Запуск будь-яких методів записи даних приведе до викиду виключення IOException.
`"rw"`	Відкриває файл по читанню і запису. Якщо файл ще не створений, то здійснюється спроба створити його.
`"rws"`	Відкриває файл по читанню і запису подібно "rw", і також вимагає системі при кожній зміні вмісту файлу або метаданих синхронно записувати ці зміни на основний носій.
`"rwd"`	Відкриває файл по читанню і запису подібно "rws", але примушує систему синхронно записувати зміни на основний носій тільки при кожній зміні вмісту файлу. Якщо змінюються метадані, синхронний запис не здійснюється.

`long getFilePointer()`

Повертає поточну позицію покажчика файлу.

```
void seek(long pos)
```

Переміщує покажчик файлу на `pos` байтів від початку файлу. Починаючи із цієї позиції, буде проводитися наступна операція читання або запису. Зсув може бути встановлений за межі файлу. У цьому випадку довжина файлу не зміниться. Довжина файлу зміниться тільки у випадку запису, при якому зсув установлений за межі файлу.

```
int skipBytes(int n)
```

Здійснює спробу пропустити n байт. Цей метод може пропустити набагато менше байт, можливо навіть 0 байт. Подібна ситуація може виникнути у випадку досягнення кінця файлу перше ніж n байт буде пропущено – це одна з можливостей. Цей метод ніколи не викидає `EOFException`. Вернеться дійсна кількість пропущених байтів. Якщо n негативно, то байти не будуть пропускатися.

```
long length()
```

Повертає довжину файлу, виражену в байтах.

```
void setLength(long newLength)
```

Установлює довжину заданого файлу.

Якщо поточна довжина файлу, що вертається методом `length` більше чим аргумент `newLength`, то файл буде обрізаний. У цьому випадку, якщо зсув, який вертається методом `getFilePointer` більше чому аргумент `newLength`, то після цього методу зсув буде рівним `newLength`. Якщо поточна довжина файлу, що вертається методом `length` менше ніж аргумент `newLength`, то файл буде розширений. У цьому випадку вміст розширюваної порції файлу не визначений, хоча часто заповнюється символом з кодом 0.

Метод для роботи з унікальним файловим каналом, асоційованим із заданим файлом:

```
FileChannel getChannel();
```

Методи для виконання звичайного і форматованого введення з файлу:

```
int read();
int read(byte b[]);
int read(byte b[],int off,int len);
final boolean readBoolean();
final byte readByte();
final char readChar();
final double readDouble();
final float readFloat();
final void readFully(byte b[]);
final void readFully(byte b[], int off, int len);
final int readInt();
final String readLine();
final long readLong();
final short readShort();
final int readUnsignedBytee();
final int readUnsignedShort();
final String readUTF();
```

Існують також методи для звичайної або форматованого записи у файл із прямим доступом:

```
void write(byte b[]);
void write(byte b[],int off,int len);
void write(int b);
final void writeBoolean(boolean v);
final void writeBytee(int v);
final void writeBytes(String s);
final void writeChar(int v);
final void writeChars(String s);
final void writeDouble(double v);
final void writeFloat(float v);
final void writeInt(int v);
final void writeLong(long v);
final void writeShort(int v);
final void writeUTF(String str);
```

Спроба відкрити неіснуючий файл тільки на читання приведе до виключення `FileNotFoundException`. При відкритті на читання і запис він буде створений відразу

(або ж буде викинуте виключення `FileNotFoundException`, якщо це неможливо здійснити).

`"rws"` і `"rwd"` режими гарантують при запуску методів, що всі зміни файлу будуть відразу ж записуватися на носій. Цей метод придатний для впевненості, що критично важлива інформація не буде загублена під час збою системи. Розглянемо докладніше процеси, що відбуваються при використанні даних режимів.

Дискові операції пов'язані з механічними переміщеннями деталей дисководу для позиціонування по диску. На подібні переміщення потрібний певний час, який сповільнює процес запису. Чим більше операцій читання-запису проводиться на диску, тим більший час віднімається на позиціонування.

Крім того, коли позиціонування відбулося, зручніше записати один великий блок даних однією операцією, чим робити запис декількох маленьких блоків. Зазвичай, операційна система буферизує операції запису в основній пам'яті і при накопиченні певної кількості даних, розмірних з великими блоками запису, робить фізичний запис даних на диск.

Таким чином, дані не посилають прямо в дисковий пристрій. Замість цього, результати операцій записи фіксуються в стабільній пам'яті і підтверджуються як завершені. Це набагато швидше, чим очікування закінчення механічної операції записи даних на диск. Через деякий час дані фіксуються на диску.

Даний принцип вигідний у тому випадку, якщо потрібна висока швидкість операцій читання/запису. Однак, у випадку збою системи виникає ризик втрати блоків файлу, які ще фізично не переписалися з пам'яті на диск.

Якщо необхідно по максимуму уникати подібних ситуацій, то використовується синхронний метод запису. У цьому випадку будь-яка операція, що приводить до зміни вмісту файлу або його метаданих, буде фіксуватися відразу на фізичний носій.

Синхронний принцип операцій запису буде виконуватися набагато повільніше, в основному через операції позиціонування на фізичному носії. Синхронний принцип виправданий у випадках роботи з базами даних, серверами, системами з транзакційною схемою обробки даних, а також системами, де втрата навіть декількох байт даних приводить до глобальних помилок.

При використанні синхронного принципу зі збереженням не тільки вмісту файлу, але і його метаданих необхідно розуміти значення самих метаданих. Метадані – це інформація, що описує безпосередньо файлову систему. Метадані файлової системи можуть містити інформацію про каталог файлу, дату останньої модифікації, індексні дескриптори і ще ряд специфічних елементів. Наприклад, у файловій системі NTFS метадані файлової системи є частиною файлу. При цьому слід мати на увазі, що не всяка файлова система може підтримувати метадані і, отже, збереження метаданих може бути не потрібно взагалі.

На практиці синхронний і асинхронний методи записи мають місце, і розробникам доводиться самостійно робити вибір між ефективністю і надійністю.

По закінченню роботи з файлом його слід закрити, викликавши метод `close()`.

Створення файлів.

Якщо при створенні екземпляра класу `RandomAccessFile` у конструкторі вказати ім'я файлу, який відсутній у файловій системі, то він буде автоматично створений з нульовою довжиною.

Автоматичне створення файлу гарантоване в тому випадку, якщо користувач має права на створення файлів у заданому каталозі. Якщо такі права відсутні, то буде викинуте виключення. Перед тем як воно буде викинуто, слід закрити потік, звільнивши системні ресурси, пов'язані із цим потоком.

Зовнішні `try catch` блоки в цьому випадку присутні через те, що використання методу `close()` у блоці `finally` вимагає або внесення в список виключень, які будуть збуджуватися методом за допомогою ключового слова `throws`, або перехоплення виключення.

У деяких випадках краще перехоплювати виключення за допомогою `try catch` блоків, чим вносити до списку виключень методу, оскільки подальше використання подібного методу усередині інших методів також вимагає перехоплення або внесення в список виключень.

Крім того, блоками `try catch` зручно управляти виведенням інформації про перехоплене виключення і обходу виключень для подальшого виконання програми.

```
Файл CreateFileDemo.java
import java.io.RandomAccessFile;
import java.io.IOException;

public class CreateFileDemo
{
   public static void main(String[] args)
   {
          RandomAccessFile raf = null;
          try
          {
                 // Create a new file
                 raf = new RandomAccessFile("output.dat", "rw");
                 raf.close();
          }catch(IOException e){
                 e.printStackTrace();
          }
   }
}
```

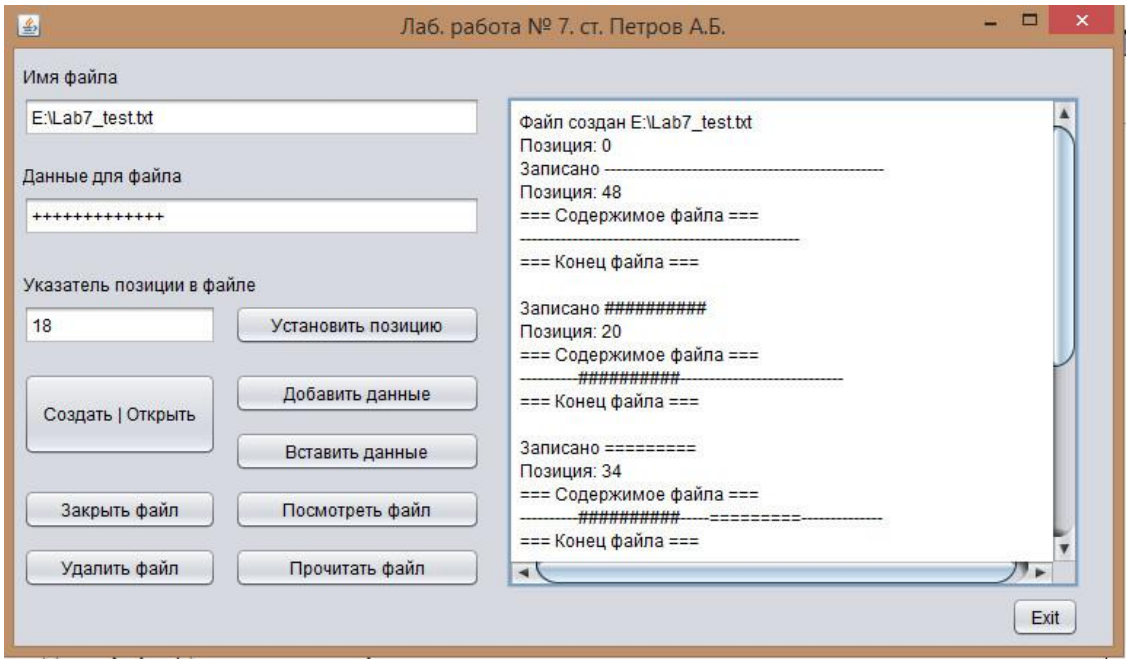
При створенні потоків можуть виникати виключення `FileNotFoundException`, `SecurityException`, `IOException`. Виключення `FileNotFoundException` виникає при спробі відкрити вхідний потік даних для неіснуючого файлу, тобто коли файл не знайдений.

Виключення `SecurityException` виникає при спробі відкрити файл, для якого заборонений доступ. Наприклад, якщо файл можна тільки читати, а він відкривається для запису, виникне виключення `SecurityException`. Якщо файл не може бути відкритий для запису по яких-небудь інших причинах, то виникає виключення `IOException`.

Методичні вказівки до виконання лабораторної роботи:

Створити форму, у яку помістити елементи керування, зазначені в завданні. Визначити оброблювачі подій для елементів керування.

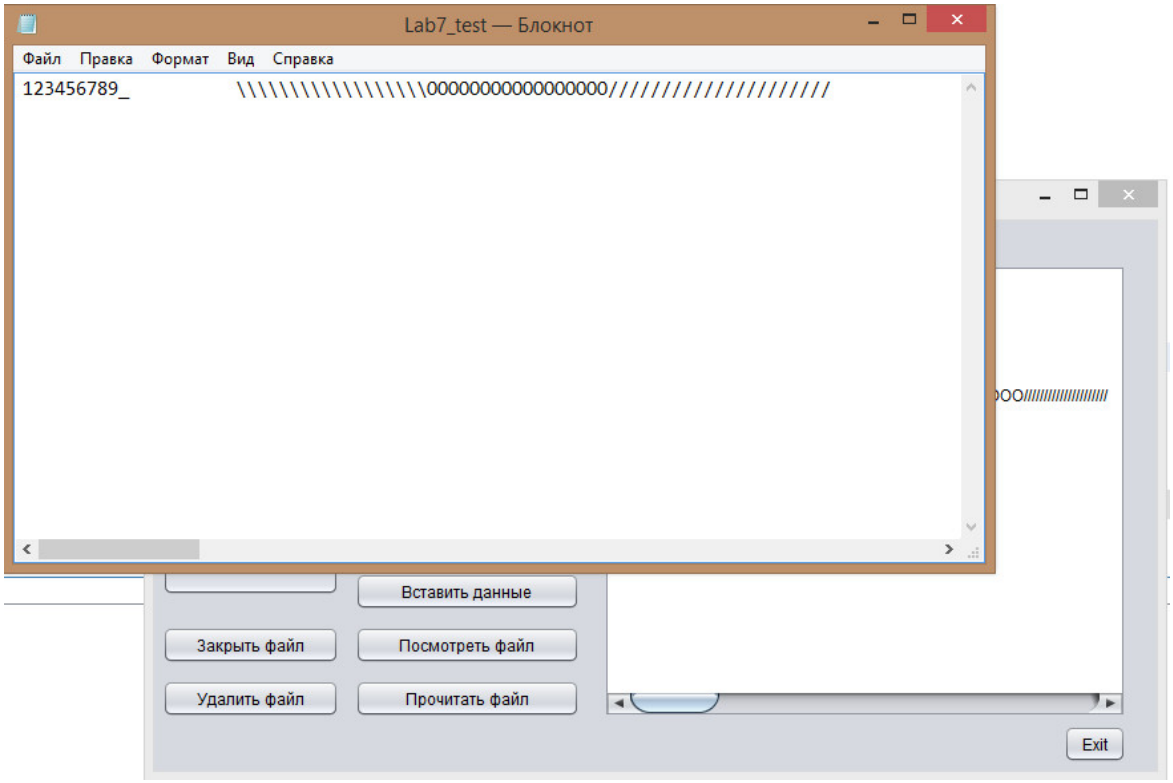
Рекомендований вигляд форми :



Виконати наступні операції:

- Створити | Відкрити файл.
- Встановити позицію покажчика у файлі.
- Додати дані в кінець файлу.
- Вставити дані в позиції покажчика файлу.
- Подивитися вміст файлу за допомогою програми Notepad.exe.
- Для перегляду використовувати класи Runtime і Process.
- Прочитати вміст файлу і відобразити в області перегляду

Перегляд вмісту файлу:



Контрольні питання:

1. Призначення класу RandomAccessFile.
2. Керування покажчиком файлу.
3. Конструктори класу RandomAccessFile.
4. Методи класу RandomAccessFile для запису даних.
5. Методи класу RandomAccessFile для читання даних.

Зміст звіту:

У звіті повинні бути представлені:

- Структурна схема програми
- Діаграма станів програми
- Лістинг програми.
- Висновки за результатами роботи.

Лабораторна робота № 8

Тема: Багатопотокове програмування

Ціль роботи

Одержання навичок програмування багатопотокових додатків

Завдання:

- Намалювати структурну схему програми.
- Створити проект Java Swing.
- Створити форму, до складу якої включити:

Кнопку Exit, прапорці для вибору потоків, поле для введення значення затримки, область для виведення повідомлень і кнопки для виконання операцій:

1. Створити потік.
2. Затримка виконання потоку.
3. Видалення потоку.

- Написати програму, що реалізує багатопотоковість для виведення імені активного потоку.

Теоретичні відомості:

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

Кожна частина такої програми називається потоком (thread), і кожний потік задає окремий шлях 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 завдань одночасно. Наприклад, текстовий редактор може форматувати текст у той же час,

коли виконується його друк, доти, поки ці дві дії виконуються двома окремими потоками.

Багатопотоковість дозволяє писати ефективні програми, які по максимуму використовують доступну міць процесора системи.

Ще однією перевагою багатопотоковості є відомість 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.

Наприклад, швидкість передачі даних по мережі набагато нижча, чим швидкість, з якої комп'ютер може їх обробляти. Навіть читання і запис ресурсів локальної файлової системи набагато повільніший, чим темп їх обробки в процесорі. І, звичайно, користувач набагато повільніше вводить дані із клавіатури, чому їх може обробити комп'ютер.

В однопотокових середовищах ваша програма змушена очікувати закінчення таких завдань, 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.

Пріоритети потоків

Java привласнює кожному потоку пріоритет, який визначає поведінка даного потоку відносно інших.

Пріоритети потоків задаються цілими числами, що визначають відносний пріоритет одного потоку в порівнянні з іншими.

Значення пріоритету саме по собі ніякого значення не має – більш високопріоритетний потік не виконується швидше, чим низькопріоритетний, коли він є єдиним виконуваним потоком у цей момент.

Замість цього пріоритет потоку використовується для ухвалення рішення при перемиканні від одного потоку, що виконується, до іншого. Це називається перемиканням контексту. Правила, які визначають, коли повинно відбуватися перемикання контексту, досить прості.

Потік може добровільно поступитися керування. Для цього можна явно поступитися черзі виконання, призупинити потік або блокувати на час очікування введення-виведення. При такому сценарії всі інші потоки перевіряються, і ресурси процесора передаються потоку з максимальним пріоритетом, який готовий до виконання.

Потік може бути перерваний іншим, більш пріоритетним потоком. У цьому випадку низькопріоритетний потік, який не займає процесор, просто припиняється високопріоритетним потоком, незалежно від того, що він робить.

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. Деякі з них перераховані в табл. 8.1.

Таблиця 8.1. Методи керування потоками класу Thread

	Призначення
`getName`	Одержати ім'я потоку
`getPriority`	Одержати пріоритет потоку
`isAlive`	Визначити, чи виконується потік
`join`	Очікувати завершення потоку
`run`	Вхідна крапка потоку
`sleep`	Призупинити виконання потоку на заданий час
`start`	Запустити потік викликом його методу run ()

Потік може знаходитися в одному із шести станів, позначених наступними константами класу-перерахування (enum) Thread.State, статично вкладеного в клас Thread:

Константи станів. (об'єкти класу Thread.State.)

- NEW – підпроцес створений, але ще не запущений;
- RUNNABLE – підпроцес виконується;
- BLOCKED – підпроцес блокований;
- WAITING – підпроцес чекає закінчення роботи іншого підпроцесу;
- TIMED_WAITING – підпроцес чекає якийсь час закінчення іншого підпроцесу;
- TERMINATED – підпроцес закінчений.

Клас Thread

У класі Thread вісім конструкторів. Основний з них,

```
Thread(ThreadGroup group, Runnable target, String name, long stackSize);
```

створює подпроцесс із 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 підпроцес 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 підпроцесу name не має ніякого значення, воно не використовується віртуальною машиною Java і застосовується тільки для розрізнення підпроцесів у програмі.

Після створення пІдпроцесу його треба запустити методом `start()`. Віртуальна машина

Java почне виконувати метод `run()` цього об'єкта-підпроцеса.

Підпроцес завершить роботу після виконання методу `run()`.

Подпроцесс, що виконується можна призупинити статичним методом

```
sleep(long ms);
```

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

```
sleep(long ms, int nanosec);
```

Головний потік

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

Створення потоку

В Java для цього визначено два способи.

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

Реалізація інтерфейсу Runnable

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

Можна створити потік з будь-якого об'єкта, що 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, що реалізує інтерфейс Runnable, треба створити об'єкт типу Thread із цього класу. У класі Thread визначено декілька конструкторів.

Той, який повинен використовуватися в цьому випадку, виглядає в такий спосіб.

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

У цьому конструкторові ***об'єкт_потоку*** – це екземпляр класу, що реалізує інтерфейс ***Runnable.*** Він визначає, де почнеться виконання потоку. Ім'я нового потоку передається в параметрі ***ім'я_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:

Створити форму, у яку помістити елементи керування, зазначені в завданні.

Визначити оброблювачі подій для елементів керування.

Рекомендований вигляд форми :

Виконати наступні операції:

- Створити потоки (4) і привласнити їм номера. Запустити на виконання
- Затримати виконання обраного потоку.
- Видалити обраний потік.

Контрольні питання:

- Призначення класу Thread.
- Призначення інтерфейсу Runnable.
- Процедура створення потоку за допомогою класу Thread.
- Процедура створення потоку використанням інтерфейсу Runnable.
- Методи класу Thread.

Зміст звіту:

У звіті повинні бути представлені:

- Структурна схема програми
- Діаграма станів програми
- Лістинг програми.

Висновки за результатами роботи.

Рекомендована література

1. Шилдт Герберт Java. Полное руководство, 8-е изд. / Шилдт Герберт – М.: ООО "И.Д. Вильямс", 2012. – 1104 с.

2. Шилдт Герберт Swing: руководство для начинающих / Шилдт Герберт – М.: ООО "И.Д. Вильямс", 2007. – 704 с.

3. Эккель Брюс Философия Java. Библиотека программиста, 4-е изд / Эккель Брюс – СПБ.: Питер, 2009. – 640 с.

4. Хорстманн Кей С., Java. Библиотека профессионала / Том 1. Основы. 9-е изд. / Хорстманн Кей С., Корнелл Гари – М.: ООО "И.Д. Вильямс", 2014. – 864 с.

5. Хорстманн Кей С., Java. Библиотека профессионала / Том 2. Расширенные средства. 9-е изд. / Хорстманн Кей С., Корнелл Гари – М.: ООО "И.Д. Вильямс", 2014. – 1008 с.