Міністерство освіти і науки України
Волинський національний університет імені Лесі Українки
Навчально-науковий фізико-технологічний інститут

В. П. Муляр

ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

Лабораторний практикум

Луцьк
Вежа-друк
2022

УДК 004.438(07)
М 90

Рекомендовано до друку науково-методичною радою
Волинського національного університету імені Лесі Українки
(протокол № 3 від 16.11.2022 р.)

Рецензенти:

Яцюк С. М. – кандидат педагогічних наук, доцент кафедри загальної математики та методики навчання інформатики, декан факультету інформаційних технологій і математики Волинського національного університету імені Лесі Українки;

Багнюк Н. В. – кандидат технічних наук, доцент кафедри комп'ютерної інженерії та кібербезпеки Луцького національного технічного університету.

Муляр В. П.

М 90 Об'єктно-орієнтоване програмування: лабораторний практикум. Луцьк : Вежа-Друк, 2022. 112 с.

У лабораторному практикумі розкрито методологію об'єктно-орієнтованого програмування на мові Java. На конкретних прикладах розглянуто основні поняття і термінологію ООП та основні її принципи: інкапсуляцію, успадкування, поліморфізм та ін. Розкрито об'єктно-орієнтований підхід до створення користувацького інтерфейсу на основі технології JavaFX.

Для студентів ЗВО педагогічного та технічного спрямування, які вивчають дисципліни «Об'єктно-орієнтоване програмування», «Програмування на Java», «Проєктування і розробка користувацьких інтерфейсів», «Алгоритми і структури даних», «Комп'ютерна фізика», «Моделювання фізичних явищ і процесів», а також для тих, хто починає освоювати об'єктно-орієнтоване програмування.

УДК 004.438(07)

ЗМІСТ

Вступ

Суть парадигми об’єктно-орієнтованого програмування – сприймати всю предметну область у вигляді об’єктів. Об’єкт – це щось, що має свій стан і поведінку. Людині дуже легко мислити в категоріях об’єктів, тому що ми живемо в світі об’єктів. На відміну від функціональної парадигми, яка вимагає перебудови мислення, щоб представити всю задачу у вигляді функцій, ООП повністю відповідає звичному мисленню людини.

На відміну від процедурно-орієнтованого програмування ООП полегшує розробку та використання програм у випадку зростання коду із збільшенням розміру проєкту.

ООП забезпечує приховування даних, тоді як у процедурно-орієнтованій мові програмування до глобальних даних можна отримати доступ із будь-якого місця.

ООП дає можливість значно ефективніше імітувати події реального світу.

Лабораторний практикум спрямований на формування професійних компетентностей створення програм на мові Java на основі об’єктно-орієнтованого програмування. На конкретних прикладах розкрито основні поняття і термінологію ООП. Розглянуто технологію розробки і проєктування додатків на платформі JavaFX із використанням декларативного способу опису інтерфейсу за допомогою мови розмітки FXML, стилізації інтерфейсу за допомогою CSS та ін.

Лабораторний практикум буде корисним для студентів ЗВО педагогічного та технічного спрямування, які вивчають дисципліни «Об’єктно-орієнтоване програмування», «Програмування на Java», «Проєктування і розробка користувацьких інтерфейсів», «Алгоритми і структури даних», «Комп’ютерна фізика», «Моделювання фізичних явищ і процесів», а також для тих, хто починає освоювати об’єктно-орієнтоване програмування.

Лабораторна робота 1. Початок роботи. Синтаксис Java

Мета: набуття вмінь та навичок встановлення Java на ПК, її налаштування для Windows, створення програм у Java.

Завдання

1. Установіть Java.

На деяких ПК може бути вже встановлена Java. Щоб перевірити, чи встановлено Java на комп'ютері з ОС Windows, знайдіть Java на панелі запуску або введіть у командному рядку (cmd.exe):

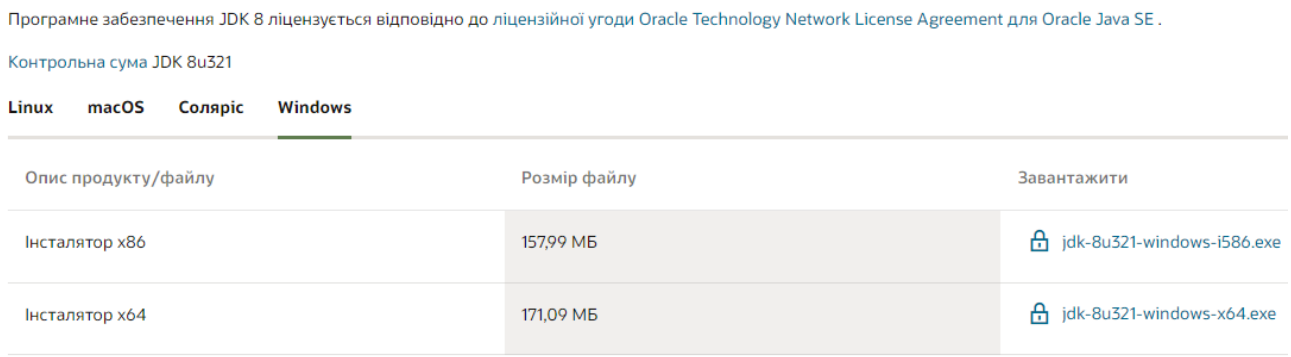
```
C:\Users\Your Name>java -version
```

Якщо встановлено Java, ви побачите щось подібне (залежно від версії):

```
java version "11.0.1" 2018-10-16 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.1+13-LTS)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.1+13-LTS, mixed mode)
```

Якщо на вашому комп'ютері не встановлено Java, ви можете безкоштовно завантажити її на oracle.com.

Важливо! У межах курсу будемо використовувати платформу Java SE Development Kit 8u321, яку можна завантажити на https://www.oracle.com/java/technologies/downloads/#java8-windows.


Програмне забезпечення JDK 8 ліцензується відповідно до ліцензійної угоди Oracle Technology Network License Agreement для Oracle Java SE .

Контрольна сума JDK 8u321

Linux macOS Солярис Windows

Опис продукту/файлу	Розмір файлу	Завантажити
Інсталятор x86	157,99 МБ	jdk-8u321-windows-i586.exe
Інсталятор x64	171,09 МБ	jdk-8u321-windows-x64.exe

Рис. 1. Вікно з інсталятором платформи Java SE Development Kit 8u321

У склад JDK 8 входить платформа JavaFX, яку будемо використовувати для проєктування користувацьких інтерфейсів. У випадку використання JDK пізніших версій потрібно буде додатково підключати бібліотеки JavaFX.

2. Налаштуйте Java на Windows.

Перейдіть до «Властивості системи» (можна знайти на панелі Керування > Система та безпека > Система > Розширені параметри системи)

Натисніть кнопку «Змінні середовища» на вкладці «Додатково».

Потім виберіть змінну «Шлях» у системних змінних і натисніть кнопку «Редагувати».

Натисніть кнопку «Новий» і додайте шлях, де встановлено Java, а потім \bin. За замовчуванням Java встановлюється в C:\Program Files\Java\jdk1.8.0_301\bin (якщо під час встановлення нічого іншого не було вказано). У цьому випадку вам доведеться додати новий шлях за допомогою: C:\Program Files\Java\jdk1.8.0_301\bin.

Потім натисніть «ОК» і збережіть налаштування.

Нарешті відкрийте командний рядок (cmd.exe) і введіть java -version, щоб перевірити, чи працює Java на вашому комп'ютері.

Покрокове встановлення Java показано нижче.

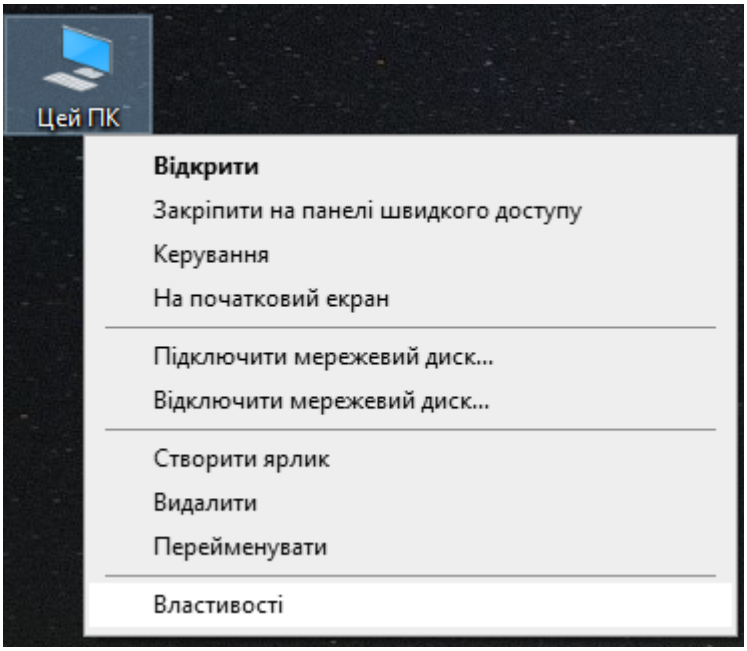


Рис. 2. Крок 1 – при натисканні правою кнопки миші на «Мій комп'ютер», вибираємо «Властивості» і з'являється вікно «Система»

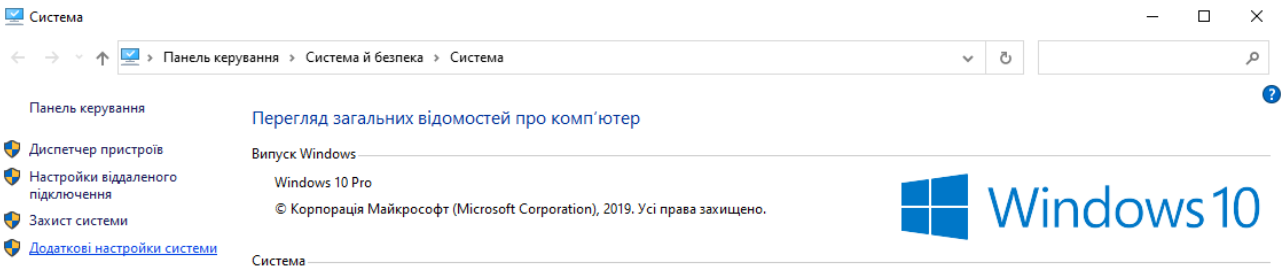


Рис. 3. Крок 2 – у вікні «Система» вибираємо «Додаткові настройки системи», з'являється вікно «Властивості системи»

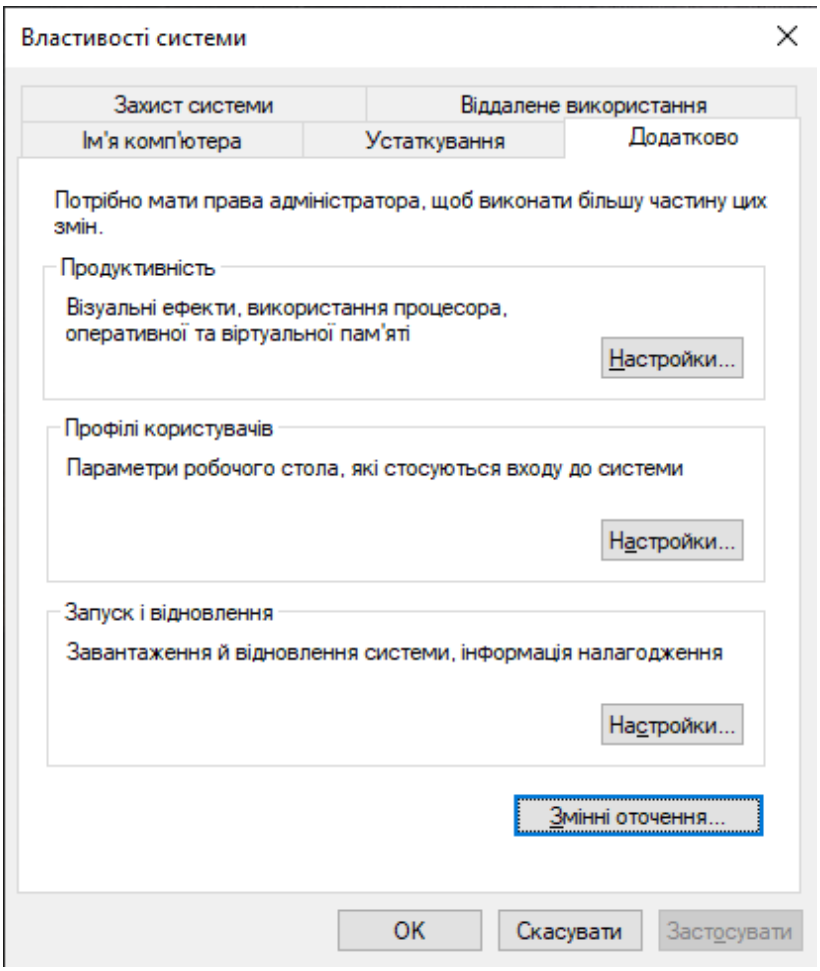


Рис. 4. Крок 3 – у вікні «Властивості системи» вибираємо «Змінні оточення»

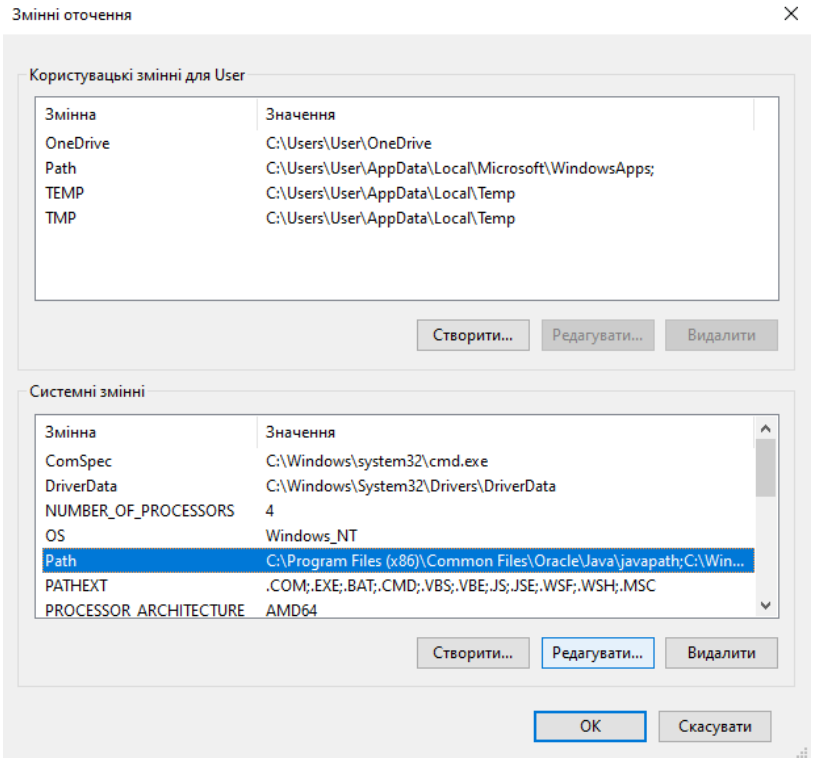


Рис. 5. Крок 4 – у вікні «Змінні оточення» в секції «Системні змінні» стаємо на рядок «Path» і вибираємо «Редагувати»

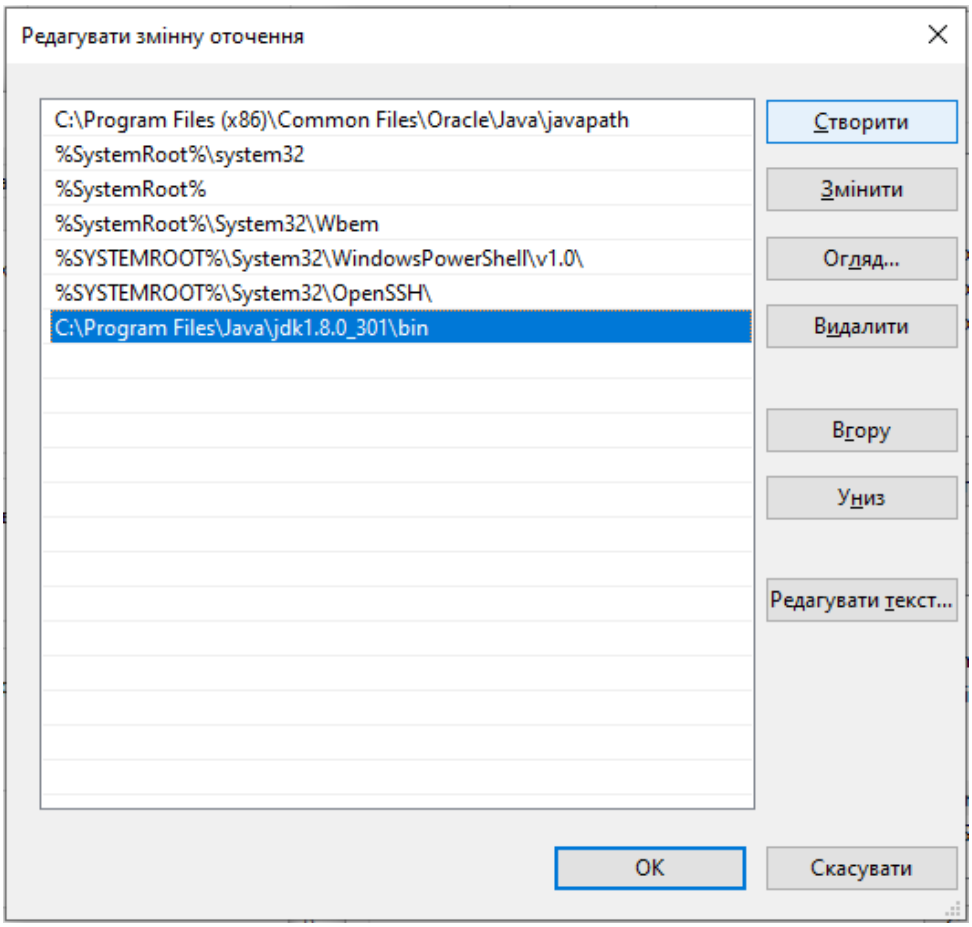


Рис. 6. Крок 5 – у вікні «Редагувати змінну оточення» вибираємо «Створити», вставляємо шлях до встановленого JDK (C:\Program Files\Java\jdk1.8.0_301\bin) та натискаємо Ok

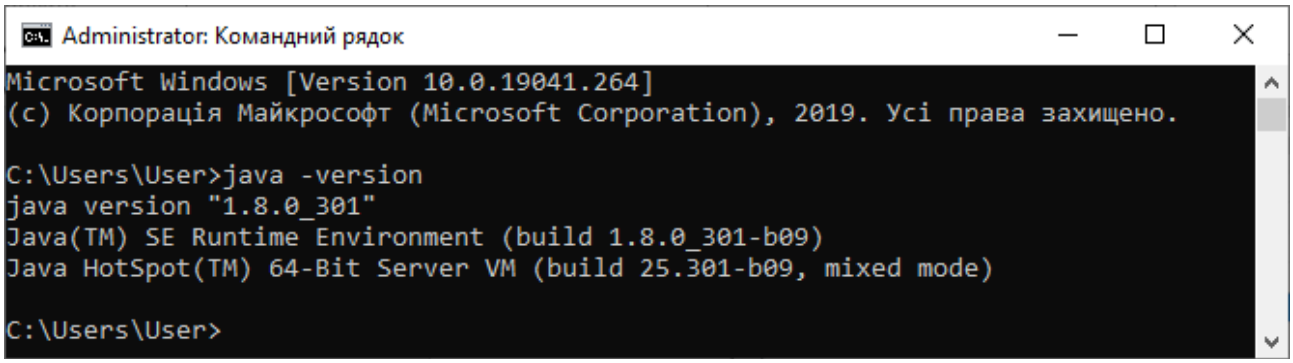


Рис. 7. Крок 6 – у командному рядку (cmd.exe) перевіряємо, чи встановлено Java (вводимо java -version і натискаємо Enter)

Примітка. У цьому курсі ми будемо писати Java-код:

– у текстовому редакторі (Блокноті), наприклад, за допомогою команди notepad Main.java можна створити файл «Main.java» в обраному каталозі;

– у вікні онлайн-компілятора https://www.onlinegdb.com/;

– в інтегрованому середовищі розробки NetBeans, яке можна завантажити на https://netbeans.apache.org/download/nb126/nb126.html (див. рис. 8).

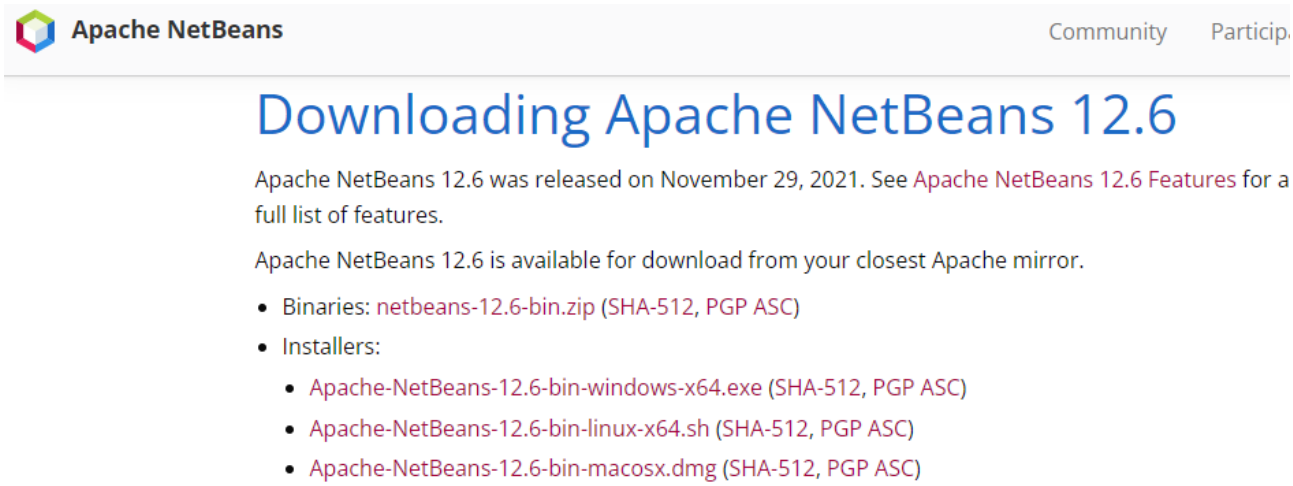


Рис. 8. Вікно з інсталятором Apache NetBeans 12.6

Крім того, писати Java-програми можна й в інших інтегрованих середовища розробки, наприклад IntelliJ IDEA або Eclipse, які використовують при створенні проєктів на мові Java.

3. Створіть першу програму Java.

У Java кожна програма починається з імені класу, і цей клас повинен відповідати імені файлу.

Давайте створимо наш перший файл Java під назвою Main.java, який можна зробити в будь-якому текстовому редакторі (наприклад, Блокноті).

Файл має містити повідомлення «Hello World», яке записується таким кодом:

```
public class Main {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

Не хвилюйтеся, якщо ви не розумієте код вище – ми детально обговоримо його в наступних лабораторних заняттях. Наразі зосередьтесь на тому, як запустити код вище.

Збережіть код у Блокноті як "Main.java". Відкрийте командний рядок (cmd.exe), перейдіть до каталогу, де ви зберегли файл, і введіть «javac Main.java»:

```
C:\Users\Your Name>javac Main.java
```

Це скомпілює ваш код. Якщо в коді немає помилок, командний рядок переведе вас до наступного рядка. Тепер введіть "java Main", щоб запустити файл:

```
C:\Users\Your Name>java Main
```

Виведення має бути таким:

```
Hello World
```

Пояснення. Кожен рядок коду, який виконується в Java, повинен бути всередині файлу class. У нашому прикладі ми назвали клас Main. Клас завжди повинен починатися з великої літери.

Примітка. Java чутлива до регістру: «MyClass» і «myclass» мають різне значення.

Ім’я файлу Java має відповідати імені класу. Під час збереження файлу збережіть його, використовуючи ім’я класу, і додайте «.java» в кінець імені файлу.

Метод main() є обов’язковим, і ви побачите його в кожній програмі Java:

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

Буде виконано будь-який код всередині методу main(). Вам не потрібно розуміти ключові слова до та після main. Ви дізнаєтеся про них пізніше.

Наразі просто пам’ятайте, що кожна програма Java має містити class з іменем, яке має відповідати імені файлу, і що кожна програма повинна містити метод main().

Усередині методу main() ми можемо використовувати метод println() для друку рядка тексту на екрані:

```
public static void main(String[] args) {
  System.out.println("Hello World");
}
```

Для звіту потрібно додати:

1) копію екрану в режимі командного рядка зі встановленою версією Java (див. рис. 7);

2) файл Main.java, який містить код програми, яка виводить у консоль «Hello World» та «Виконавець: Ваше Прізвище та ім’я»;

3) відповіді на контрольні запитання.

Контрольні запитання

1. Як перевірити, чи на вашому комп’ютері встановлена Java?
2. Де можна завантажити Java?
3. Де можна писати Java-код?

4. Які інтегровані середовища розробки використовуються для написання Java-коду?

5. Як налаштувати Java на Windows?

6. Який синтаксис Java-програми?

7. Чи чутливий Java до регістру?

8. Яке розширення файлу, який містить Java-код?

9. Яке розширення файлу із скомпільованим Java-кодом?

10. Як скомпілювати Java-файл?

11. Як запустити на виконання Java-програму?

12. З якої літери повинен починатися клас в Java?

13. Чи має ім’я файлу Java відповідати імені класу?

14. Який метод (функція) є обов’язковим для кожної Java-програми?

15. Який метод використовується для друку рядка тексту на екрані?

Лабораторна робота 2. Класи та об’єкти

Мета: набуття вмінь та навичок створення класів та об’єктів у Java.

Теоретичні відомості: матеріали лекцій 1-2.

Завдання

1. Створіть клас Main з полем *x*.

Щоб створити клас, використовуємо ключове слово class.

```
public class Main {
  int x = 5;
}
```

Пам’ятаємо, що клас завжди повинен починатися з великої літери, а ім’я файлу Java має відповідати імені класу.

2. Створіть об’єкт під назвою "myObj" і надрукуйте значення *x*.

У Java об’єкт створюється з класу. Ми вже створили клас з ім’ям Main, тож тепер ми можемо використовувати його для створення об’єктів.

Щоб створити об’єкт Main, вкажіть ім’я класу, а потім ім’я об’єкта та використовуйте ключове слово new.

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
```

```
  }
}
```

3. Створіть два об’єкти Main і виведіть їх значення *x*.

Ви можете створити кілька об’єктів одного класу.

```
public class Main {
  int x = 5;
  public static void main(String[] args) {
    Main myObj1 = new Main();   // Object 1
    Main myObj2 = new Main();   // Object 2
    System.out.println(myObj1.x);
    System.out.println(myObj2.x);
  }
}
```

4. Створити об’єкт класу та отримати доступ до нього в іншому класі.

Ви також можете створити об’єкт класу та отримати доступ до нього в іншому класі. Це часто використовується для кращої організації класів (один клас має всі атрибути та методи, а інший клас містить метод main() (код, який потрібно виконати)).

Пам’ятайте, що ім’я файлу Java має відповідати імені класу. У цьому прикладі ми створили два файли в одному каталозі/папці:

Main.java

Second.java

Main.java:

```
public class Main {
  int x = 5;
}
```

Second.java:

```
class Second {
  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

Коли обидва файли були зібрані:

```
C:\Users\>javac Main.java
C:\Users\>javac Second.java
```

Запустіть файл Second.java:

```
C:\Users\Your Name>java Second
```

І вихід буде:

```
5
```

5. Написати програму, яка ілюструє визначення класу Student з полями id (порядковий номер), name (ім’я).

Ми створюємо об’єкт класу Student за новим ключовим словом і друкуємо значення об’єкта. Тут ми створюємо метод main() всередині класу.

Файл: Student.java

```
class Student {
    int id;
    String name;
    public static void main(String args[]){
        Student s1=new Student();
        System.out.println(s1.id);
        System.out.println(s1.name);
    }
}
```

Результат:

```
0
null
```

Для звіту потрібно додати:

архів папки «ЛР2. Класи та об’єкти», яка містить папки з файлами (.java, .class) виконаних завдань 1–5, та індивідуального завдання, яке полягає у модифікації завдання 5 таким чином, щоб вивести на екран номер, прізвище та ім’я студента-виконавця, а також відповіді на контрольні запитання.

Контрольні запитання

1. Що таке об’єкт?
2. Що таке ООП?
3. Які концепції ООП ви знаєте?
4. Що таке клас?
5. Які переваги ООП над процедурно-орієнтованим програмуванням?
6. Яких правил іменування потрібно дотримуватися під час програмування на мові Java?
7. Для чого використовується ключове слово new?
8. Наведіть приклади об’єктів і класів, їх описи в Java.

Лабораторна робота 3. Атрибути класу

Мета: набуття вмінь та навичок роботи з атрибутами (полями) класу в Java.

Теоретичні відомості: матеріали лекції 2.

Завдання

1. Створіть клас під назвою "Main" з двома атрибутами: *x* і *y*.

Атрибути класу є змінними всередині класу. Інший термін для атрибутів класу – поля.

Наступний код ілюструє створення класу "Main" з двома полями: *x* і *y*.

```
public class Main {
  int x = 5;
  int y = 3;
}
```

2. Створіть об’єкт під назвою "myObj" і надрукуйте значення *x*.

Ви можете отримати доступ до атрибутів, створивши об’єкт класу та використовуючи синтаксис крапки (.).

Наступний приклад створить об’єкт класу Main з ім’ям myObj. Ми використовуємо атрибут *x* об’єкта, щоб надрукувати його значення:

```
public class Main {
  int x = 5;
  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

3. Установіть значення змінної *x* на 40.

Ви також можете змінити значення атрибутів:

```
public class Main {
  int x;
  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 40;
    System.out.println(myObj.x);
  }
}
```

4. Змініть значення *x* на 25.

```
public class Main {
```

```
  int x = 10;
  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 25; // x is now 25
    System.out.println(myObj.x);
  }
}
```

5. Проілюструйте використання ключового слова *final* для поля *x*.

Якщо ви не хочете мати можливість змінювати наявні значення, оголосіть атрибут як *final:*

```
public class Main {
  final int x = 10;
  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 25; /* генерується помилка: cannot assign a value to a final variable */
    System.out.println(myObj.x);
  }
}
```

Ключове слово *final* використовується тоді, коли ви хочете, щоб змінна завжди зберігала одне й те саме значення, наприклад PI (3.14159...).

Ключове слово *final* називається «модифікатором».

6. Змініть значення *x* на 25 для *myObj2* і залиште *x* без змін для *myObj1*.

Якщо ви створюєте кілька об’єктів одного класу, ви можете змінити значення атрибутів в одному об’єкті, не впливаючи на значення атрибутів в іншому. У нашому випадку:

```
public class Main {
  String fname = "John";
  String lname = "Doe";
  int age = 24;
  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println("Name: " + myObj.fname + " " + myObj.lname);
    System.out.println("Age: " + myObj.age);
  }
}
```

7. Проілюструйте, як можна ініціалізувати об’єкт класу *Student* за допомогою поля *name*.

Ініціалізація об’єкта означає зберігання даних у об’єкті.

Файл: TestStudent2.java

```
class Student{
  int id;
  String name;
}
class TestStudent2{
  public static void main(String args[]){
    Student s1=new Student();
    s1.id=101;
    s1.name="Sonoo";
    System.out.println(s1.id+" "+s1.name);
  }
}
```

Виведення на екран:

```
101 Sonoo
```

8. Проілюструйте, яким чином можна ініціалізувати об’єкт із використанням методу.

У цьому прикладі ми створюємо два об’єкти класу *Student* та ініціалізуємо значення для цих об’єктів, викликаючи метод *insertRecord*. Тут ми відображаємо стан (дані) об’єктів, викликаючи метод *displayInformation()*.

Файл: TestStudent4.java

```
class Student {
  int rollno;
  String name;
  void insertRecord(int r, String n){
    rollno=r;
    name=n;
  }
  void displayInformation() {
    System.out.println(rollno+" "+name);}
  }
class TestStudent4{
  public static void main(String args[]){
   Student s1=new Student();
    Student s2=new Student();
```

```
        s1.insertRecord(111,"Karan");
        s2.insertRecord(222,"Aryan");
        s1.displayInformation();
        s2.displayInformation();
    }
}
```

Виведення на екран:

```
111 Karan
222 Aryan
```

До звіту потрібно подати:

архів папки «ЛР3. Атрибути класу», яка містить папки з файлами (.java, .class) виконаних завдань 1–8, та індивідуального завдання, яке полягає у модифікації завдання 8 таким чином, щоб вивести на екран інформацію про студента-виконавця та двох його одногрупників, а також відповіді на контрольні запитання.

Контрольні запитання

1. Що атрибути класу?
2. Як отримати доступ до полів класу?
3. Як встановити (змінити) значення поля?
4. Яке ключове слово використовують для того, щоб зафіксувати значення поля? Наведіть приклади.
5. Що таке ініціалізація об’єкта?
6. Як ініціалізувати об’єкт за допомогою полів?
7. Як можна ініціалізувати об’єкт із використанням методу? Наведіть приклади.

Лабораторна робота 4. Методи класу

Мета: набуття вмінь та навичок створення методів класу та отримання до них доступу за допомогою об’єктів.

Теоретичні відомості: матеріали лекції 3.

Завдання

1. Створіть метод із назвою myMethod() в Main:

```
public class Main {
  static void myMethod() {
    System.out.println("Hello World!");
  }
}
```

Як бачимо, методи оголошуються всередині класу і використовуються для виконання певних дій. myMethod() друкує текст

(дія), коли він викликається. Щоб викликати метод, напишіть назву методу, а потім дві дужки () і крапку з комою ";".

2. Використайте myMethod() всередині main.

```
public class Main {
  static void myMethod() {
    System.out.println("Hello World!");
  }
  public static void main(String[] args) {
    myMethod();
  }
}
```

3. Продемонструйте відмінності між методами static та public.

У наведеному нижче прикладі ми створили static метод, який означає, що до нього можна отримати доступ без створення об'єкта класу, на відміну від public, до якого можуть отримати доступ лише об'єкти.

```
public class Main {
  // Static method
  static void myStaticMethod() {
    System.out.println("Static methods can be called without creating objects");
  }
  // Public method
  public void myPublicMethod() {
    System.out.println("Public methods must be called by creating objects");
  }
  // Main method
  public static void main(String[] args) {
    myStaticMethod(); // Call the static method
    // myPublicMethod(); This would compile an error
    Main myObj = new Main(); // Create an object of Main
    myObj.myPublicMethod(); // Call the public method on the object
  }
}
```

4. Створіть об'єкт класу Car з ім'ям myCar. Викличте методи fullThrottle() та speed() для об'єкта myCar, запустіть програму.

```
// Create a Main class
public class Main {
```

```
  // Create a fullThrottle() method
  public void fullThrottle() {
    System.out.println("The car is going as fast as it can!");
  }
  // Create a speed() method and add a parameter
  public void speed(int maxSpeed) {
    System.out.println("Max speed is: " + maxSpeed);
  }
  // Inside main, call the methods on the myCar object
  public static void main(String[] args) {
    Main myCar = new Main();     // Create a myCar object
    myCar.fullThrottle();         // Call the fullThrottle() method
    myCar.speed(200);               // Call the speed() method
  }
}
// The car is going as fast as it can!
// Max speed is: 200
```

Пояснення прикладу.

1) Ми створили спеціальний клас Main із ключовим словом class.

2) Ми створили методи fullThrottle() і speed() в класі Main.

3) Метод fullThrottle() і метод speed() виводять деякий текст, коли вони викликані.

4) Метод speed() приймає параметр int під назвою maxSpeed – ми будемо використовувати це в 8).

5) Щоб використовувати клас Main і його методи, нам потрібно створити об’єкт класу Main.

6) Потім перейдіть до main() методу, який, як ви вже знаєте, є вбудованим методом Java, який запускає вашу програму (виконується будь-який код всередині main).

7) За допомогою ключового слова new ми створили об’єкт з іменем myCar.

8) Потім ми викликаємо методи fullThrottle() та myCar.speed(200) для об’єкта та запускаємо програму.

5. Створити об’єкт класу та отримати доступ до нього в іншому класі.

Пам’ятайте, що ім’я файлу Java має відповідати імені класу. У цьому прикладі ми створили два файли в одному каталозі:

Main.java;

Second.java.

Файл Main.java:

```
public class Main {
  public void fullThrottle() {
    System.out.println("The car is going as fast as it can!");
  }
  public void speed(int maxSpeed) {
    System.out.println("Max speed is: " + maxSpeed);
  }
}
```

Файл Second.java:

```
class Second {
  public static void main(String[] args) {
    Main myCar = new Main();       // Create a myCar object
    myCar.fullThrottle();          // Call the fullThrottle() method
    myCar.speed(200);              // Call the speed() method
  }
}
```

Коли обидва файли були зібрані:

```
C:\Users\Your Name>javac Main.java
C:\Users\Your Name>javac Second.java
```

Запустіть файл Second.java:

```
C:\Users\Your Name>java Second
```

І отримайте результат:

```
The car is going as fast as it can!
Max speed is: 200
```

Для звіту потрібно подати:

архів папки «ЛР4. Методи класу», яка містить папки з файлами (.java, .class) виконаних завдань 1–5, та індивідуального завдання, яке полягає у модифікації завдання 5 таким чином, щоб вивести на екран інформацію про власника (студента-виконавця) автомобіля та його характеристики (потужність, швидкість), а також відповіді на контрольні запитання

Контрольні запитання

1. Що таке метод класу?
2. Як викликати метод?
3. У чому полягає відмінність між методами static та public? Наведіть приклади.
4. Яке ключове слово використовують для того, щоб зафіксувати значення поля? Наведіть приклади.

5. Чи можна створити об’єкт класу та отримати доступ до нього в іншому класі? Наведіть приклади.

Лабораторна робота 5. Конструктори

Мета: набуття вмінь та навичок створення конструкторів та їх використання для створення та ініціалізації об’єктів.

Теоретичні відомості: матеріали лекції 4.

Завдання

1. Створіть конструктор для класу Main.

Конструктор в Java – це спеціальний метод, який використовується для ініціалізації об’єктів. Конструктор викликається, коли створюється об’єкт класу. Його можна використовувати для встановлення початкових значень для атрибутів об’єкта:

```
// Create a Main class
public class Main {
  int x; // Create a class attribute
  // Create a class constructor for the Main class
  public Main() {
    x = 5; // Set the initial value for the class attribute x
  }
  public static void main(String[] args) {
    Main myObj = new Main(); // Create an object of class Main (This will call the constructor)
    System.out.println(myObj.x); // Print the value of x
  }
}
// Outputs 5
```

Зауважте, що ім’я конструктора має відповідати імені класу, і воно не може мати тип повернення (наприклад, void).

Також зауважте, що конструктор викликається під час створення об’єкта.

Усі класи мають конструктори за замовчуванням: якщо ви не створюєте конструктор класу самостійно, Java створює його для вас. Однак тоді ви не зможете встановити початкові значення для атрибутів об’єкта.

2. Створіть конструктор з параметрами.

Конструктори також можуть приймати параметри, які використовуються для ініціалізації атрибутів.

Наступний приклад додає параметр *int y* до конструктора. Усередині конструктора ми встановлюємо *x* на *y* (*x = y*). Коли ми викликаємо конструктор, ми передаємо параметр конструктору (5), який встановить значення *x* рівним 5:

```
public class Main {
  int x;
  public Main(int y) {
    x = y;
  }
  public static void main(String[] args) {
    Main myObj = new Main(5);
    System.out.println(myObj.x);
  }
}
// Outputs 5
```

3. Створіть конструктор з декількома параметрами.

```
 public class Main {
  int modelYear;
  String modelName;
  public Main(int year, String name) {
    modelYear = year;
    modelName = name;
  }
  public static void main(String[] args) {
    Main myCar = new Main(1969, "Mustang");
    System.out.println(myCar.modelYear + " " +
myCar.modelName);
  }
}
// Outputs 1969 Mustang
```

4. Проілюструйте перевантаження конструктора в Java.

Перевантаження конструктора в Java – це техніка, що містить кілька конструкторів із різними списками параметрів. Вони розташовані таким чином, що кожен конструктор виконує інше завдання. Вони відрізняються компілятором за кількістю параметрів у списку та їх типами.

Приклад перевантаження конструктора.

```
class Student5{
    int id;
```

```
    String name;
    int age;
    // створення конструктора з двома параметрами
    Student5(int i, String n){
        id = i;
        name = n;
    }
    // створення конструктора з трьома параметрами
    Student5(int i, String n, int a){
        id = i;
        name = n;
        age = a;
    }
    void display(){
        System.out.println(id+" "+name+" "+age);
    }
      public static void main(String args[]){
        Student5 s1 = new Student5(111,"Karan");
        Student5 s2 = new Student5(222,"Aryan",25);
        s1.display();
        s2.display();
   }
}
```

Виведення на екран:

```
111 Karan 0
222 Aryan 25
```

Для звіту потрібно подати:

архів папки «ЛР5. Конструктори», яка містить папки з файлами (.java, .class) виконаних завдань 1–4, та індивідуального завдання, яке передбачає створення конструктора з чотирма параметрами (номер по списку, прізвище, ім’я, вік) та його використання для виведення на екран інформації про студента-виконавця, а також відповіді на контрольні запитання.

Контрольні запитання

1. Що таке конструктор?
2. Яке призначення конструкторів в Java?
3. Чи можна використовувати конструктор для створення об’єктів? Наведіть приклади.

4. Чи можна використовувати конструктор для ініціалізації об’єктів? Наведіть приклади.

5. Яка відмінність між методом і конструктором в Java?

6. Що таке конструктор за замовчуванням?

7. Що таке конструктор з параметрами? Наведіть приклади.

8. Що таке перевантаження конструктора? Наведіть приклади.

Лабораторна робота 6. Модифікатори

Мета: набуття вмінь та навичок використання модифікаторів у Java.

Теоретичні відомості: матеріали лекцій 3-4.

Завдання

1. Створіть клас Main з використанням модифікатора *final*.

Якщо ви не хочете мати можливість змінювати наявні значення атрибутів, оголосіть атрибути як *final*:

```
public class Main {
  final int x = 10;
  final double PI = 3.14;
  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 50; // буде згенерована помилка: cannot assign a value to a final variable
    myObj.PI = 25; // буде згенерована помилка: cannot assign a value to a final variable
    System.out.println(myObj.x);
  }
}
```

2. Створіть клас *Main* з використанням модифікатора *static*.

Модифікатор *static* означає, що до нього можна отримати доступ без створення об’єкта класу, на відміну від *public*.

Приклад, щоб продемонструвати відмінності між методами *static* та *public*:

```
public class Main {
  // Статичний метод
  static void myStaticMethod() {
    System.out.println("Статичні методи можна викликати без створення об’єктів");
  }
  // Загальнодоступний метод
```

```
    public void myPublicMethod() {
      System.out.println("Загальнодоступні методи необхідно
викликати під час створення об’єктів");
    }
    // Основний метод
    public static void main(String[ ] args) {
      myStaticMethod(); // Викликаємо статичний метод
      // myPublicMethod(); Це виведе помилку
      Main myObj = new Main(); // Створення об’єкту Main
      myObj.myPublicMethod(); // Виклик публічного методу
    }
  }
```

3. Створіть абстрактний клас із використанням модифікатора *abstract.*

Абстрактний метод належить до абстрактного класу і не має тіла. Тіло надається підкласом:

```
// Код з файлу: Main.java
// абстрактний клас
abstract class Main {
  public String fname = "John";
  public int age = 24;
  public abstract void study(); // абстрактний метод
}
// Підклас (успадкований від Main)
class Student extends Main {
  public int graduationYear = 2018;
  public void study() {
    // тут представлено тіло абстрактного методу
    System.out.println("Studying all day long");
  }
}
// Код з файлу: Main.java
// Код з файлу: Second.java
class Second {
  public static void main(String[] args) {
    // створення об’єкту класу Student
    //(який успадковує поля та методи від Main)
    Student myObj = new Student();
    System.out.println("Name: " + myObj.fname);
```

```
        System.out.println("Age: " + myObj.age);
        System.out.println("Graduation Year: " +
myObj.graduationYear);
        myObj.study(); // виклик абстрактного методу
    }
}
```

Для звіту потрібно подати:

архів папки «ЛР6. Модифікатори» (або копії екранів чи посилання на код програм в https://www.onlinegdb.com/), яка містить папки з файлами (.java, .class) виконаних завдань 1–3, та індивідуального завдання, яке передбачає виведення на екран інформації про студента-виконавця (прізвище, ім'я, група, курс), а також відповіді на контрольні запитання.

Контрольні запитання

1. Що таке модифікатори доступу? Яке їх призначення?
2. Які модифікатори доступу можна застосовувати до класів?
3. Які модифікатори можна використовувати для атрибутів, методів і конструкторів?
4. Яке призначення модифікатора *final*?
5. Для чого використовують модифікатор *static*?
6. Що таке абстрактний метод?
7. Яке призначення модифікатора *abstract*? Наведіть приклади.

Лабораторна робота 7. Інкапсуляція

Мета: набуття вмінь та навичок використання інкапсуляції в Java.

Теоретичні відомості: матеріал лекції 7.

Завдання

1. Створіть клас *Person*, який ілюструє використання інкапсуляції в Java.

Із попередньої лабораторної роботи ви дізналися, що доступ *private* до змінних можна отримати лише в межах одного класу (зовнішній клас не має доступу до них). Однак доступ до них можна отримати, якщо ми використаємо загальнодоступні методи *get* і *set*.

Метод *get* повертає значення змінної, а метод *set* встановлює значення.

Синтаксис для обох полягає в тому, що вони починаються з *get* або *set*, за яким слідує ім'я змінної, перша літера у верхньому регістрі:

```
public class Person {
```

```
  private String name; // private = restricted access
  // Getter
  public String getName() {
    return name;
  }
  // Setter
  public void setName(String newName) {
    this.name = newName;
  }
}
```

Пояснення прикладу. Метод *get* повертає значення змінної name.

Метод *set* приймає параметр (*newName*) і призначає його змінній name. Ключове слово *this* використовується для посилання на поточний об’єкт.

Однак, оскільки змінна name оголошена як *private*, ми не можемо отримати до неї доступ за межами цього класу:

```
public class Main {
  public static void main(String[] args) {
    Person myObj = new Person();
    myObj.name = "John";   // error
    System.out.println(myObj.name); // error
  }
}
```

Якщо змінна була оголошена як *public*, ми очікували б наступного результату:

```
John
```

Однак, коли ми намагаємося отримати доступ до змінної *private*, ми отримуємо помилку:

```
MyClass.java:4: error: name has private access in Person
    myObj.name = "John";
          ^
MyClass.java:5: error: name has private access in Person
    System.out.println(myObj.name);
                       ^
2 errors
```

2. Створіть метод Main із використанням методи *getName()* та *setName()* для доступу та оновлення змінної.

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

```
        Person myObj = new Person();
        myObj.setName("John"); // Set the value of the name variable to
"John"
        System.out.println(myObj.getName());
      }
    }
    // Outputs "John"
```

3. Створіть клас Student, який має лише одне поле з методами сеттера і геттера.

Файл: Student.java

```
// Клас Java, який є повністю інкапсульованим класом.
// У нього є приватне поле даних, а також методи
// отримання та встановлення.
package com.javatpoint;
public class Student{
    // приватне поле даних
    private String name;
    // метод getter для name
    public String getName(){
        return name;
    }
    // метод setter для name
    public void setName(String name){
        this.name = name;
    }
}
```

Файл: Test.java

```
// Клас Java для перевірки інкапсульованого класу.
package com.javatpoint;
class Test{
    public static void main(String[] args){
        // створення екземпляра інкапсульованого класу
        Student s = new Student();
        // встановлення значення поля name
        s.setName("vijay");
        // отримання значення поля name
        System.out.println(s.getName());
    }
}
```

Компіляція: javac -d . Test.java

Виконання: java com.javatpoint.Test

Виведення на екран:

```
vijay
```

Для звіту потрібно подати:

архів папки «ЛР7. Інкапсуляція», яка містить папки з файлами (.java, .class) виконаних завдань 1–3, та індивідуального завдання, яке передбачає використання повністю інкапсульованого класу для виведення на екран інформації про студента-виконавця (прізвище, ім’я, назву групи), а також відповіді на контрольні запитання.

Контрольні запитання

1. Що таке інкапсуляція?
2. Які переваги інкапсуляції?
3. Що таке геттери? Наведіть приклади.
4. Що таке сеттери? Наведіть приклади.

Лабораторна робота 8. Пакети

Мета: набуття вмінь та навичок роботи з пакетами у Java.

Теоретичні відомості: матеріал лекції 8.

Завдання

1. Створіть клас *MyClass*, який передбачає використання класу Scanner для отримання інформації від користувача.

Пакет у Java використовується для групування пов’язаних класів, аналогічно папці в каталозі файлів. Ми використовуємо пакети, щоб уникнути конфліктів імен і написати краще підтримуваний код. Пакети діляться на дві категорії:

– вбудовані пакети (пакети з Java API);

– пакети, визначені користувачем (створюйте власні пакети).

Вбудовані пакети. Java API — це бібліотека попередньо написаних класів, які безкоштовні для використання, включені в середовище розробки Java.

Бібліотека містить компоненти для керування введенням даних, програмування бази даних та багато іншого. Повний список можна знайти на веб-сайті Oracle: https://docs.oracle.com/javase/8/docs/api/.

Бібліотека поділена на пакети та класи. Це означає, що ви можете імпортувати окремий клас (разом з його методами та атрибутами), або цілий пакет, який містить усі класи, що належать до вказаного пакету.

Щоб використовувати клас або пакет із бібліотеки, потрібно використовувати ключове слово *import:*

```
import package.name.Class;     // Import a single class
import package.name.*;     // Import the whole package
```

У наведеному нижче прикладі java.util – це пакет, *Scanner* – клас пакету java.util.

Щоб використовувати клас *Scanner*, створіть об’єкт класу та скористайтеся будь-яким із доступних методів, які можна знайти в документації класу *Scanner*. У нашому прикладі ми будемо використовувати метод *nextLine()*, який використовується для читання рядка.

Файл Main.java:

```
import java.util.Scanner; // import the Scanner class
class Main {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);
    String userName;
    // Enter username and press Enter
    System.out.println("Enter username");
    userName = myObj.nextLine();
    System.out.println("Username is: " + userName);
  }
}
```

Результат:

```
Enter username
Vadim
Username is: Vadim
```

Зауважимо, щоб імпортувати весь пакет, потрібно закінчити речення знаком зірочки (*). Наступний приклад імпортує всі класи в пакеті *java.util:*

```
import java.util.*;
```

2. Створіть клас *MyPackageClass* з пакетом, який визначений користувачем.

Щоб створити свій власний пакет, ви повинні розуміти, що Java використовує для їх зберігання каталог файлової системи. Як і папки на вашому комп’ютері:

```
Приклад
└── root
    └── mypack
        └── MyPackageClass.java
```

Щоб створити пакет, використовуйте ключове слово *package*.

```
package mypack;
class MyPackageClass {
  public static void main(String[] args) {
    System.out.println("This is my package!");
  }
}
```

Збережіть файл як MyPackageClass.java та скомпілюйте його:

```
C:\Users\Your Name>javac MyPackageClass.java
```

Потім компілюйте пакет:

```
C:\Users\Your Name>javac -d . MyPackageClass.java
```

Це змушує компілятор створити пакет "mypack".

Ключове слово -d вказує місце призначення для збереження файлу класу. Ви можете використовувати будь-яку назву каталогу, наприклад c:/user (windows), або, якщо ви хочете зберегти пакет у тому самому каталозі, ви можете використовувати знак крапки ".", як у прикладі вище.

Примітка. Ім’я пакета має бути написане в нижньому регістрі, щоб уникнути конфлікту з іменами класів.

Коли ми зібрали пакет у наведеному вище прикладі, була створена нова папка під назвою «mypack».

Щоб запустити файл MyPackageClass.java, напишіть наступне:

```
C:\Users\Your Name>java mypack.MyPackageClass
```

Результат:

```
This is my package!
```

Для звіту потрібно подати:

архів папки «ЛР8. Пакети», яка містить папки з файлами (.java, .class) виконаних завдань 1–2, та індивідуального завдання, яке передбачає модифікацію виконання 2 завдання для виведення на екран інформації про студента-виконавця (прізвище, ім’я, назва групи), а також відповіді на контрольні запитання.

Контрольні запитання

1. Що таке пакети Java?
2. Яке призначення пакетів Java?
3. Що таке Java API?
4. Як можна завантажити окремий клас? Наведіть приклади.
5. Як можна завантажити пакет із бібліотеки?
6. Що розуміють під пакетами, які визначені користувачем?

Лабораторна робота 9. Успадкування

Мета: набуття вмінь та навичок використання успадкування в Java.

Теоретичні відомості: матеріал лекції 9.

Завдання

1. Створити клас (підклас) *Car*, який успадковує атрибути та методи від класу *Vehicle* (суперкласу).

У Java можна успадковувати атрибути та методи від одного класу до іншого. Ми групуємо «концепцію успадкування» на дві категорії:

– підклас (дочірній) – клас, який успадковується від іншого класу;

– суперклас (батька) – клас, від якого успадковується.

Щоб успадкувати від класу, використовуйте ключове слово *extends*.

Файл Car.java:

```
class Vehicle {
  protected String brand = "Ford";
  public void honk() {
    System.out.println("Tuut, tuut!");
  }
}
class Car extends Vehicle {
  private String modelName = "Mustang";
  public static void main(String[] args) {
    Car myFastCar = new Car();
    myFastCar.honk();
    System.out.println(myFastCar.brand + " " +
myFastCar.modelName);
  }
}
```

Результат:

```
Tuut, tuut!
Ford Mustang
```

Ви помітили модифікатор protected у Vehicle?

Ми встановлюємо для атрибуту *brand* у *Vehicle* модифікатор доступу *protected*. Якщо для нього встановлено значення *private*, клас *Car* не зможе отримати до нього доступ.

Чому і коли використовувати «спадкування»?

– Це корисно для повторного використання коду: повторно використовуйте атрибути та методи існуючого класу під час створення

нового класу.

Примітка. Якщо ви не хочете, щоб інші класи успадковували клас, використовуйте ключове слово *final*.

2. Створіть клас *Dog*, який успадковує клас *Animal*.

Коли клас успадковує інший клас, це називається однорівневим *успадкуванням*.

Файл: TestInheritance.java

```
class Animal{
    void eat(){
        System.out.println("eating..."); // їсть
    }
}
class Dog extends Animal{
    void bark(){
        System.out.println("barking..."); // гавкає
    }
}
class TestInheritance{
    public static void main(String args[]){
        Dog d = new Dog();
        d.bark();
        d.eat();
    }
}
```

Результат:

```
barking...
eating...
```

3. Створіть клас *BabyDog*, що успадковує клас *Dog*, який у свою чергу успадковує клас *Animal*.

Це приклад *багаторівневого успадкування, тобто коли існує ланцюг успадкування.*

Файл: TestInheritance2.java

```
class Animal{
    void eat(){
        System.out.println("eating..."); // їсть
    }
}
class Dog extends Animal{
    void bark(){
```

```
        System.out.println("barking..."); // гавкає
    }
}
class BabyDog extends Dog{
    void weep(){
        System.out.println("weeping..."); // плаче
    }
}
class TestInheritance2{
    public static void main(String args[]){
        BabyDog d = new BabyDog();
        d.weep();
        d.bark();
        d.eat();
    }
}
```

Результат:

```
weeping...
barking...
eating...
```

4. Створіть класи *Dog* і *Cat*, які успадковують клас *Animal*.

Якщо два або більше класів успадковують один клас, це називається *ієрархічним успадкуванням*. У наведеному нижче прикладі маємо ієрархічне успадкування.

Файл: TestInheritance3.java

```
class Animal{
    void eat(){
        System.out.println("eating..."); // їсть
    }
}
class Dog extends Animal{
    void bark(){
        System.out.println("barking..."); // гавкає
    }
}
class Cat extends Animal{
    void meow(){
        System.out.println("meowing..."); // нявкає
    }
```

```
}
class TestInheritance3{
    public static void main(String args[]){
        Cat c = new Cat();
        c.meow();
        c.eat();
    //c.bark();//C.T.Error
    }
}
```

Виведення на екран:

```
meowing...
eating...
```

Для звіту потрібно подати:

архів папки «ЛР9. Спадкування», яка містить папки з файлами (.java, .class) виконаних завдань 1–4, та індивідуального завдання, яке передбачає створення класу *Parrot*, який успадковує клас *Animal*, та вимовляє прізвище господаря (студента-виконавця), а також відповіді на контрольні запитання.

Контрольні запитання

1. Що таке спадкування в Java?
2. Навіщо використовувати спадкування в Java?
3. Що таке суперклас?
4. Що таке підклас?
5. Що розуміють під повторним використанням?
6. Який синтаксис спадкування в Java?
7. Які є види спадкування в Java?
8. Що таке однорівневе спадкування?
9. Що розуміють під багаторівневим спадкуванням?
10. Що таке ієрархічне спадкування? Наведіть приклади.
11. Чи підтримується множинне спадкування в Java?

Лабораторна робота 10. Перевизначення та перевантаження

Мета: набуття вмінь та навичок перевизначення і перевантаження в Java.

Теоретичні відомості: матеріал лекцій 11–12.

Завдання

1. Створіть клас *Robot*, який описує робота, що вміє переміщатися з однієї точки в іншу.

Файл Robot.java:

```
package robot3;
public class Robot {
    private double x = 0;
    private double y = 0;
    protected double course = 0;
    public Robot(double x, double y) {
        this.x = x;
        this.y = y;
    }
    // Пересування на дистанцію distance
    public void forward(int distance) {
        x = x + distance * Math.cos (course / 180 * Math.PI);
        y = y + distance * Math.sin (course / 180 * Math.PI);
    }
    // Друк координат робота
    public void printCoordinates() {
        System.out.println(x + "," + y);
    }
    public double getX() {
        return x;
    }
    public double getY() {
        return y;
    }
    public double getCourse() {
        return course;
    }
    public void setCourse(double course) {
        this.course = course;
    }
}
```

2. Створіть клас *RobotTotal*, який обчислює пройдену відстань.

Для створення робота, який все-таки вміє обчислювати відстань, можна піти наступним шляхом: успадкуватися від класу *Robot*, доповнити новий клас змінною, наприклад *totalDistance* і при переміщенні збільшувати її на пройдену дистанцію. Тобто, нам треба перевизначити метод *forward*, у якому треба розрахувати нові координати і додати до змінної *totalDistance* величину вхідного параметра *distance*. Як бачимо все, досить логічно, за винятком одного

неприємного моменту – у нас уже є алгоритм, який вважає нові координати і не використовувати його було б дивно. Потрібен механізм, який дозволить викликати батьківські методи. І цей механізм існує.

Файл RobotTotal.java:

```
package robot3;
public class RobotTotal extends Robot {
    // Вводимо поле для зберігання пройденої дистанції
    private double totalDistance = 0;
    // перевантажений конструктор
    public RobotTotal() {
        super(0, 0);
    }
    // Конструктор теж треба перевизначити
    public RobotTotal(double x, double y) {
        super(x, y);
    }
    @Override
    public void forward(int distance) {
        // Виклик нашого методу у батьківському класі
        // Потрібно вказати зарезервоване слово super
        super.forward(distance);
        totalDistance += distance;
    }
    public double getTotalDistance() {
        return totalDistance;
    }
}
```

Пояснення. По-перше, можна побачити, що ми додали нове поле *totalDistance*. Його призначення ми вже розглядали. Цікавіший момент – це необхідність створення конструктора. Звертаємо увагу, якби ми не створили конструктор з параметрами класу *Robot*, то нам не потрібно було б створювати конструктор у нашому новому класі. Цю ситуацію ми розглянемо глибше під час розгляду перевантаження методів. Поки що ж запам’ятаємо наступне – якщо батьківському класі немає конструктора без параметрів, то клас-нащадок має визначити свій конструктор. Причому не обов’язково повторювати набір параметрів. Ви можете провести експеримент та прибрати з конструктора класу *RobotTotal* параметр *double y*. У ньому можна

замінити виклик *super(x, y);* на *super(x, 0);* І це буде цілком робочий код.

Перейдемо до перевизначеного методу *forward*. Тут бачимо спеціальну конструкцію виклику методу батьківського класу, саме зарезервоване слово *super* і через крапку виклик методу *forward*. Напевно, ось і весь механізм – треба просто використовувати слово super. Виклик батьківського методу можна здійснювати у будь-якому місці перевизначеного дочірнього методу. Можна, наприклад, спочатку збільшити змінну *totalDistance* і потім викликати метод *forward*.

Перевантаження (*overload*) методу полягає в наступному – ви створюєте метод з таким самим ім'ям, але з іншим набором параметрів.

3. Створіть клас *RobotManager*, який використовує клас *RobotTotal*, успадкований від класу *Robot*.

Файл RobotManager.java:

```
package robot3;
public class RobotManager {
    public static void main(String[] args) {
        RobotTotal robot = new RobotTotal(0, 0);
        robot.forward(20);
        robot.setCourse(90);
        robot.forward(20);
        robot.setCourse(90);
        robot.forward(50);
        // Друкувати координати
        robot.printCoordinates();
        // Надрукувати дистанцію
        System.out.println(robot.getTotalDistance());
    }
}
```

Результат:

```
20.000000000000004,70.0
90.0
```

Для звіту потрібно подати:

архів папки «ЛР10. Перевизначення та перевантаження», яка містить папки з файлами (.java, .class) виконаних завдань 1–3, та індивідуального завдання, яке передбачає виведення на екран прізвище студента-виконавця та дистанції, яку проходить робот, описуючи літеру «П», а також відповіді на контрольні запитання.

Контрольні запитання

1. Що таке перевантаження методів?
2. Які існують способи перевантаження методу? Наведіть приклади.
3. Що розуміють під перевизначенням методу в Java?
4. Яке призначення перевизначення методу?
5. Які існують правила заміни методів у Java?

Лабораторна робота 11. Відносини між класами

Мета: набуття вмінь та навичок встановлення відношень між класами в Java.

Теоретичні відомості: матеріал лекції 10.

Завдання

Маємо два класи *Robot* і *Operator*. Потрібно встановити зв'язки між роботом і оператором, який ним керує.

Між класами можна встановити асоціацію через посилання в одному класі на інший клас. Тобто, клас *Robot* має посилання на клас *Operator* і навпаки – клас *Operator* має посилання на клас *Robot*.

Файл Robot.java:

```
package robotoperator;

public class Robot {
    private double x = 0;
    private double y = 0;
    protected double course = 0;
    // Робот управляється оператором
    private Operator operator;
    public Robot(double x, double y) {
        this.x = x;
        this.y = y;
    }
    // Можна дізнатися який оператор керує роботом
    public Operator getOperator() {
        return operator;
    }
    // Можна встановити оператора для робота
    public void setOperator(Operator operator) {
        this.operator = operator;
    }
    public void forward(int distance) {
```

```
            x = x + distance * Math.cos (course / 180 * Math.PI);
            y = y + distance * Math.sin (course / 180 * Math.PI);
        }
        public double getX() {
            return x;
        }
        public double getY() {
            return y;
        }
        public double getCourse() {
            return course;
        }
        public void setCourse(double course) {
            this.course = course;
        }
        public void printCoordinates() {
            System.out.println(x + "," + y);
        }
}
```

Файл Operator.java:

```
package robotoperator;

public class Operator {
    private String firstName;
    private String lastName;
    // Оператор управляє конкретним роботом
    private Robot robot;
    public Operator(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
    public String getFirstName() {
        return firstName;
    }
    public String getLastName() {
        return lastName;
    }
    // У оператора можна запитати, яким роботом він керує
    public Robot getRobot() {
```

```
        return robot;
    }
    // Оператору можна доручити керувати роботом
    public void setRobot(Robot robot) {
        this.robot = robot;
    }
}
```

Файл RobotManager.java:

```
package robotoperator;
public class RobotManager {
    public static void main(String[] args) {
        Robot r1 = new Robot(0,0);
        r1.setCourse(0);
        System.out.print("Coordinates r1: ");
        r1.printCoordinates();
        System.out.println("course r1: "+r1.getCourse());
        Operator op1 = new Operator("Anna","Nalepa");
        r1.setOperator(op1);
        op1.setRobot(r1);
        System.out.println(r1.getOperator().getFirstName());
        System.out.println(r1.getOperator().getLastName());
        op1.getRobot().printCoordinates();
        System.out.println(op1.getRobot().getCourse());
    }
}
```

Результат:

```
Coordinates r1: 0.0,0.0
course r1: 0.0
Anna
Nalepa
0.0,0.0
0.0
```

Для звіту потрібно подати:

архів папки «ЛР11. Відносини між класами», яка містить папки з файлами (.java, .class) виконаного завдання із виведенням на екран прізвища та ім’я оператора (студента-виконавця), який керує роботом, а також відповіді на контрольні запитання.

Контрольні запитання

1. Що таке відношення IS-A між класами в Java? Наведіть приклади.

2. Що таке відношення HAS-A між класами в Java?
3. Що таке асоціація?
4. Що таке агрегація?
5. Що таке композиція?
6. Яка відмінність між агрегацією і композицією?

Лабораторна робота 12. Поліморфізм

Мета: набуття вмінь та навичок використання поліморфізму в Java.

Теоретичні відомості: матеріал лекції 17.

Завдання

1. Дано суперклас з назвою *Animal*, який має метод під назвою *animalSound()*. Підкласами тварин можуть бути *Свині*, *Коти*, *Собаки*, *Птахи*. Показати, що вони також мають свою власну реалізацію звуку (свиня хрюкає, а кіт нявкає тощо).

Поліморфізм означає «багато форм», і він виникає, коли у нас є багато класів, пов’язаних один з одним спадковістю.

Як було показано раніше, спадкування дозволяє нам успадковувати атрибути та методи від іншого класу. Поліморфізм використовує ці методи для виконання різних завдань. Це дозволяє нам виконувати одну дію різними способами.

Файл Main.java:

```
class Animal {
  public void animalSound() {
    System.out.println("The animal makes a sound");
  }
}
class Pig extends Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
}
class Dog extends Animal {
  public void animalSound() {
    System.out.println("The dog says: bow wow");
  }
}
class Main {
  public static void main(String[] args) {
```

```
        Animal myAnimal = new Animal();
        Animal myPig = new Pig();
        Animal myDog = new Dog();
        myAnimal.animalSound();
        myPig.animalSound();
        myDog.animalSound();
    }
}
```

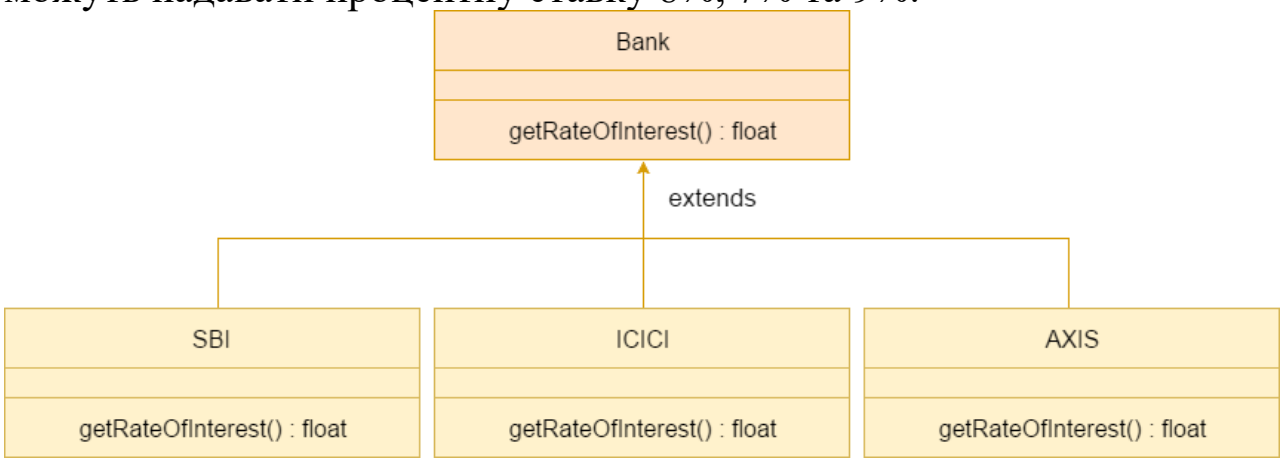
Результат:

```
The animal makes a sound
The pig says: wee wee
The dog says: bow wow
```

Пригадаймо, що ми використовуємо ключове слово *extends* для успадкування від класу.

2. Написати програму Java для демонстрації реального сценарію заміни методу Java, де три класи замінюють метод батьківського класу.

Розглянемо сценарій, коли банк – це клас, який забезпечує функціональність для отримання процентної ставки. Однак процентна ставка залежить від банків. Наприклад, банки SBI, ICICI та AXIS можуть надавати процентну ставку 8%, 7% та 9%.



Примітка. Перевизначення методу Java в основному використовується в поліморфізмі середовища виконання.

Файл Test2:

```
// Створення батьківського класу.
class Bank{
    int getRateOfInterest(){
        return 0;
    }
}
```

```
// Створення дочірніх класів.
class SBI extends Bank{
    int getRateOfInterest(){
        return 8;
    }
}
class ICICI extends Bank{
    int getRateOfInterest(){
        return 7;
    }
}
class AXIS extends Bank{
    int getRateOfInterest(){
        return 9;
    }
}
// Тестування класу для створення об’єктів та виклику методів
class Test2{
    public static void main(String args[]){
        SBI s = new SBI();
        ICICI i = new ICICI();
        AXIS a = new AXIS();
        System.out.println("SBI Rate of Interest:
  "+s.getRateOfInterest());
        System.out.println("ICICI Rate of Interest:
  "+i.getRateOfInterest());
        System.out.println("AXIS Rate of Interest:
  "+a.getRateOfInterest());
    }
}
```

Результат:

```
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9
```

Для звіту потрібно подати:

архів папки «ЛР12. Поліморфізм», яка містить папки з файлами (.java, .class) виконаних завдань 1–2 та індивідуального завдання, яке передбачає створення дочірнього класу PRIVATBANK та виведення на екран його процентної ставки (10 %) і прізвища студента-

виконавця, а також відповіді на контрольні запитання.

Контрольні запитання

1. Що таке поліморфізм?
2. Навіщо використовувати поліморфізм в Java?
3. Що таке перевизначення методу?
4. Що таке перезавантаження методу?
5. Яка відмінність між перевизначенням і перезавантаженням методу? Наведіть приклади.

Лабораторна робота 13. Внутрішні класи

Мета: набуття вмінь та навичок використання внутрішніх (вкладених) класів у Java.

Теоретичні відомості: матеріал лекцій 9–10.

Завдання

1. Створити клас Main, який ілюструє використання внутрішніх класів у Java.

У Java також можна вкладати класи (клас у класі). Метою вкладених класів є групування класів, що робить ваш код більш читабельним і зручним для підтримки.

Щоб отримати доступ до внутрішнього класу, створіть об’єкт зовнішнього класу, а потім створіть об’єкт внутрішнього класу.

Файл Main.java:

```
class OuterClass {
  int x = 10;
  class InnerClass {
    int y = 5;
  }
}
public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.y + myOuter.x);
  }
}
```

Результат:

```
15
```

2. Створити клас *Main*, який ілюструє використання приватних внутрішніх класів у Java.

На відміну від "звичайного" класу, внутрішній клас може бути *private* або *protected*. Якщо ви не хочете, щоб зовнішні об’єкти отримували доступ до внутрішнього класу, оголосіть клас як *private*.

Файл Main.java:

```
class OuterClass {
  int x = 10;
  private class InnerClass {
    int y = 5;
  }
}
public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.y + myOuter.x);
  }
}
```

Результат:

```
Main.java:13: error: OuterClass.InnerClass has private access in OuterClass
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
                ^
Main.java:13: error: OuterClass.InnerClass has private access in OuterClass
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
                                                ^
2 errors
```

Виникає помилка, бо ми намагаємось отримати доступ до приватного внутрішнього класу із зовнішнього класу.

3. Створити клас *Main*, який ілюструє використання статичних внутрішніх класів у Java.

Внутрішній клас також може бути *static*, що означає, що ви можете отримати до нього доступ, не створюючи об’єкт зовнішнього класу.

```
class OuterClass {
  int x = 10;

  static class InnerClass {
    int y = 5;
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    OuterClass.InnerClass myInner = new OuterClass.InnerClass();
    System.out.println(myInner.y);
  }
}
```

Результат:

```
5
```

Примітка: як і статичні атрибути та методи, так і внутрішній клас не має доступу до членів зовнішнього класу.

4. Створити клас *Main*, який ілюструє доступ до зовнішнього класу з внутрішнього класу.

Однією з переваг внутрішніх класів є те, що вони можуть отримати доступ до атрибутів і методів зовнішнього класу.

```
class OuterClass {
  int x = 10;
  class InnerClass {
    public int myInnerMethod() {
      return x;
    }
  }
}
public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.myInnerMethod());
  }
}
```

Результат:

```
10
```

Для звіту потрібно подати:

архів папки «ЛР13. Внутрішні класи», яка містить папки з файлами (.java, .class) виконаних завдань 1–4 та індивідуального завдання, яке ілюструє доступ до зовнішнього класу *OuterStudent*, який містить поля *Прізвище* та *Назву групи*, з внутрішнього класу *InnerStudent* та виводить на екран прізвище студента-виконавця та групи, в якій він навчається, а також відповіді на контрольні запитання.

Контрольні запитання

1. Що таке внутрішній клас?

2. Як отримати доступ до внутрішнього класу? Наведіть приклад.

3. Як діяти, коли ви не хочете, щоб зовнішні об’єкти отримували доступ до внутрішнього класу?

4. Чи може бути внутрішній клас статичним?

5. Як отримати доступ до статичного внутрішнього класу? Наведіть приклад.

6. Чи можна отримати доступ до зовнішнього класу з внутрішнього класу? Наведіть приклади.

Лабораторна робота 14. Абстракція класів та методів

Мета: набуття вмінь та навичок створення абстрактних класів та методів у Java.

Теоретичні відомості: матеріал лекції 20.

Завдання

1. Перетворити клас *Animal* в абстрактний клас.

Пригадаймо, що абстракція даних – це процес приховування певних деталей і показу користувачеві лише важливої інформації.

Абстракції можна досягти за допомогою абстрактних класів або інтерфейсів (про які ви дізнаєтеся більше в наступному розділі).

Ключове слово *abstract* є модифікатором без доступу, який використовується для класів і методів:

– абстрактний клас – це обмежений клас, який не можна використовувати для створення об’єктів (щоб отримати до нього доступ, він повинен бути успадкований від іншого класу).

– абстрактний метод може використовуватися тільки в абстрактному класі, і він не має тіла. Тіло надається підкласом (успадкованим від іншого класу).

Абстрактний клас може мати як абстрактні, так і неабстрактні методи:

```
abstract class Animal {
  public abstract void animalSound();
  public void sleep() {
    System.out.println("Zzz");
  }
}
```

Із наведеного вище прикладу неможливо створити об’єкт класу *Animal*:

```
Animal myObj = new Animal(); // will generate an error
```

Щоб отримати доступ до абстрактного класу, він повинен бути успадкований від іншого класу.

Давайте перетворимо клас Animal, який ми використовували під час розгляду поліморфізму, в абстрактний клас.

Файл Main.java:

```
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}
// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}
class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Результат:

```
The pig says: wee wee
Zzz
```

2. Створити абстрактний клас *Shape*, класи *Rectangle* та *Circle*, які забезпечують його реалізацію.

У цьому прикладі, якщо ви створите екземпляр класу *Rectangle*, буде викликано метод *draw()* класу *Rectangle*.

Файл: TestAbstraction1.java

```
abstract class Shape{
    abstract void draw();
}
```

```
// У реальному сценарії реалізацію забезпечують інші,
// тобто невідома кінцевим користувачем
class Rectangle extends Shape{
    void draw(){
        // креслення прямокутника
        System.out.println("drawing rectangle");
    }
}
class Circle1 extends Shape{
    void draw(){
        // креслення кола
        System.out.println("drawing circle");
    }
}
// У реальному сценарії метод викликається програмістом
// або користувачем
class TestAbstraction1{
    public static void main(String args[]){
        // У реальному сценарії об’єкт надається через метод,
        // наприклад, метод getShape()
        Shape s = new Circle1();
        s.draw();
    }
}
```

Виведення на екран:

```
drawing circle
```

3. Написати програму, яка ілюструє створення абстрактного класу, який має абстрактні та неабстрактні методи.

Абстрактний клас може мати поле, абстрактний метод, тіло методу (неабстрактний метод), конструктор і навіть метод main().

Файл: TestAbstraction2.java

```
abstract class Bike{
    Bike(){
        // велосипед створено
        System.out.println("bike is created");
    }
    abstract void run();
    void changeGear(){
        // передача змінена
```

```
        System.out.println("gear changed");
    }
}
// Створення дочірнього класу, який успадковує
// абстрактний клас
class Honda extends Bike{
    void run(){
        // безпечно працює..
        System.out.println("running safely..");
    }
}
// Створення класу TestAbstraction2, який викликає абстрактні
// та неабстрактні методи
class TestAbstraction2{
    public static void main(String args[]){
        Bike obj = new Honda();
        obj.run();
        obj.changeGear();
    }
}
```

Результат:

```
bike is created
running safely..
gear changed
```

Отже, якщо в класі є абстрактний метод, цей клас має бути абстрактним.

Для звіту потрібно подати:

архів папки «ЛР14. Абстракція класів та методів», яка містить папки з файлами (.java, .class) виконаних завдань 1–3 та індивідуального завдання, яке полягає у створенні дочірнього класу *Yamaha*, який успадковує абстрактний клас *Bike*, і виводить на екран "*running Yamaha*" і власника транспортного засобу (прізвище, ім'я студента-виконавця). Також потрібно дати відповіді на контрольні запитання.

Контрольні запитання

1. Що таке абстракція даних?
2. Що таке абстрактний клас? Наведіть приклад.
3. Що таке абстрактний метод?
4. Чи може мати абстрактний клас неабстрактні методи?

5. З якою метою використовують абстрактні класи і методи?

Лабораторна робота 15. Інтерфейси

Мета: набуття вмінь та навичок створення інтерфейсів у Java.

Теоретичні відомості: матеріал лекції 21.

Завдання

1. Створити інтерфейс *Animal* та отримати доступ до його методів.

Відомо, що іншим способом досягнення абстракції в Java є інтерфейси.

Інтерфейс – це повністю "абстрактний клас", який використовується для групування пов’язаних методів з порожнім тілом, наприклад:

```
// interface
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void run(); // interface method (does not have a body)
}
```

Іншими словами, поля інтерфейсу є загальнодоступними, статичними та остаточними за замовчуванням, а методи є відкритими та абстрактними.

Щоб отримати доступ до методів інтерфейсу, інтерфейс повинен бути «реалізований» (як би успадкований) іншим класом з ключовим словом *implements* (замість *extends*). Тіло методу інтерфейсу надається класом "*implement*".

Файл Main.java:

```
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void sleep(); // interface method (does not have a body)
}
class Pig implements Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    System.out.println("Zzz");
  }
}
class Main {
```

```
  public static void main(String[] args) {
    Pig myPig = new Pig();
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Результат:

```
The pig says: wee wee
Zzz
```

Пояснення щодо інтерфейсів:

– як і абстрактні класи, інтерфейси не можна використовувати для створення об’єктів (у наведеному вище прикладі неможливо створити об’єкт «Animal» у MyMainClass);

– методи інтерфейсу не мають тіла – тіло надається класом "*implement*";

– при реалізації інтерфейсу ви повинні перевизначити всі його методи;

– методи інтерфейсу за замовчуванням *abstract* і *public*;

– атрибутами інтерфейсу за замовчуванням є *public*, *static* і *final*;

– інтерфейс не може містити конструктор (оскільки його не можна використовувати для створення об’єктів).

2. Створити інтерфейс *Drawable*, який має лише один метод, а його реалізація забезпечується класами *Rectangle* і *Circle*.

У реальному сценарії інтерфейс визначається кимось іншим, але його реалізацію забезпечують різні постачальники впровадження. Більше того, ним користується хтось інший. Частина реалізації прихована користувачем, який використовує інтерфейс.

Файл: TestInterface1.java

```
// Оголошення інтерфейсу: першим користувачем
interface Drawable{
    void draw();
}
// Реалізація: другим користувачем
class Rectangle implements Drawable{
    public void draw(){
        System.out.println("drawing rectangle");
    }
}
class Circle implements Drawable{
```

```
    public void draw(){
        System.out.println("drawing circle");
    }
}
// Використання інтерфейсу: третім користувачем
class TestInterface1{
    public static void main(String args[]){
        // У реальному сценарії об’єкт надається методом,
        // наприклад getDrawable()
        Drawable d = new Circle();
        d.draw();
    }
}
```

Виведення на екран:

```
drawing circle
```

Чому і коли використовувати інтерфейси?

1) Щоб забезпечити безпеку – приховайте певні деталі та покажіть лише важливі деталі об’єкта (інтерфейсу).

2) Java не підтримує «множинну спадковість» (клас може успадковувати лише один суперклас). Однак цього можна досягти за допомогою інтерфейсів, оскільки клас може реалізувати декілька інтерфейсів.

Примітка. Щоб реалізувати декілька інтерфейсів, розділіть їх комою (див. приклад нижче).

3. Створити програму, яка ілюструє реалізацію множинного успадкування за допомогою інтерфейсу.

Якщо клас реалізує декілька інтерфейсів або інтерфейс розширює декілька інтерфейсів, це називається *множинним успадкуванням*.

Файл A7.java:

```
interface Printable{
    void print();
}
interface Showable{
    void show();
}
class A7 implements Printable, Showable{
    public void print(){
        System.out.println("Hello");
    }
```

```
    public void show(){
        System.out.println("Welcome");
    }
    public static void main(String args[]){
        A7 obj = new A7();
        obj.print();
        obj.show();
    }
}
```

Результат:

```
Hello
Welcome
```

Для звіту потрібно подати:

архів папки «ЛР15. Інтерфейси», яка містить папки з файлами (.java, .class) виконаних завдань 1–3 та індивідуального завдання, яке полягає у створенні та реалізації інтерфейсів для виведення на екран інформації про студента-виконавця (прізвища, імені та групи), а також відповіді на контрольні запитання.

Контрольні запитання

1. Що таке інтерфейс у Java?
2. Яке призначення інтерфейсу?
3. Яка відмінність між абстрактним класом й інтерфейсом?
4. Як отримати доступ до методів інтерфейсу?
5. Чи можна реалізувати множинне успадкування за допомогою інтерфейсу?

Лабораторна робота 16. Архітектура JavaFX. GUI-компонент Button

Мета: набуття вмінь та навичок створення JavaFX-додатків, які містять компонент Button.

Теоретичні відомості

Модель програмування додатків платформи JavaFX 2.0. Один і той же код JavaFX-додатку може запускатися як настільний додаток, який розгортається на клієнтському комп’ютері автономно, може розгортатися як додаток *Java Web Start* або відображатися у Web-браузері як JavaFX-аплет, вбудований в HTML-сторінку.

Точкою входу в JavaFX-додаток служить Java-клас, що розширює абстрактний клас javafx.application.Application і метод main():

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

```
        launch(args);
    }
    public void init(){
        //Ініціалізація додатку
        . . .
    }
    @Override
    public void start(Stage primaryStage) {
        //Установка параметрів сцени
        . . .
        primaryStage.setScene(scene);
        primaryStage.setVisible(true);
    }
    public void stop(){
        //Звільнення ресурсів додатку
        . . .
    }
}
```

У методі main() головного класу JavaFX-додатку викликає метод launch() класу Application, що відповідає за завантаження JavaFX-додатку. Крім того, головний клас JavaFX-додатку повинен перевизначити абстрактний метод start() класу Application, що забезпечує створення та відображення сцени JavaFX-додатку.

Методи init() і stop() класу Application можуть використовуватись для ініціалізації даних та звільнення ресурсів JavaFX-додатку.

Оскільки метод init() викликається перед створенням головного потоку додатку JavaFX Application Thread, то ініціалізація JavaFX-додатку в методі init() за участю вузлів графа сцени має здійснюватися із застосуванням статичного методу javafx.application.Platform.runLater().

Для виконання JavaScript-коду на Web-сторінці, що містить JavaFX-додаток, головний клас JavaFX-додатку може використовувати метод getHostServices() класу Application та об’єкт netscape.javascript.JSObject.

Обробка вхідних аргументів або параметрів у головному класі JavaFX-додатку може бути здійснена за допомогою виклику методу getParameters() класу Application.

Поліпшити відображення та обробку процесу запуску та завантаження JavaFX-додатку можна декількома способами. Перший спосіб це використання обробника onGetSplash JavaScript-бібліотеки

Deployment Toolkit API для створення заставки запуску JavaFX-аплету, вбудованого в Web-сторінку. Інший спосіб – це застосування CSS-стилів до Preloader-завантажувача за замовчуванням. І нарешті, можна створити свій клас передзавантажувача, що розширює абстрактний клас javafx.application.Preloader, і послатися на нього в JNLP-дескрипторі розгортання JavaFX-додатку. При цьому для зв'язку головного класу JavaFX-додатка із завантажувачем можна використовувати метод notifyPreloader() класу Application.

Метод start() класу Application містить як параметр об'єкт javafx.stage.Stage, що представляє графічний контейнер головного вікна JavaFX-додатку. Цей об'єкт Stage створюється середовищем виконання під час запуску JavaFX-додатку і передається в метод start() головного класу JavaFX-додатку, що дозволяє використовувати методи об'єкта Stage для встановлення та відображення сцени JavaFX-додатку. Замість об'єкта Stage, аргументу методу start(), розробник може створити свій екземпляр класу Stage для відображення сцени JavaFX-додатку.

Перед встановленням та відображенням сцени у графічному контейнері Stage головного вікна JavaFX-додатку необхідно створити граф сцени, що складається з кореневого вузла та його дочірніх елементів, та на його основі створити об'єкт javafx.scene.Scene сцени.

Як правило, як кореневий сайт використовується об'єкт javafx.scene.Group, який створюється за допомогою конструктора і виступає як аргумент конструктора при створенні об'єкта javafx.scene.Scene.

Дочірні вузли графа сцени, що представляють графіку, елементи контролю GUI-інтерфейсу, медіаконтент, додаються до кореневого вузол за допомогою методу getChildren().add() або методу getChildren().addAll(). При цьому дочірні вузли можуть мати візуальні ефекти, режими накладання, CSS-стилі, прозорість, трансформації, обробники подій, брати участь в анімації за ключовими кадрами, анімації та ін.

Програмний інтерфейс JavaFX API. Програмний інтерфейс JavaFX API дає можливість розробляти Rich Client Application (RIA) додатки з насиченим графічним інтерфейсом, код яких поєднує широкі можливості платформи Java з багатою графікою та медіафункціональністю платформи JavaFX.

Компоненти графічного інтерфейсу користувача платформи JavaFX представлені такими пакетами JavaFX API, як javafx.scene.control,

javafx.scene.chart, javafx.scene.image, javafx.scene.layout, javafx.scene.media, javafx.scene.shape, javafx.scene.text, javafx.scene.web та javafx.stage.

Усі компоненти GUI-інтерфейсу є об'єктами Node вузлів графа сцени та характеризуються ідентифікатором, CSS-стилем, межами, візуальними ефектами, прозорістю, трансформаціями, обробниками подій, станом, режимом накладання та участю в анімації.

Пакет javafx.scene.control надає такі GUI-компоненти, як панель Accordion, кнопку Button, прапорець CheckBox, список ChoiceBox, контекстне меню ContextMenu, гіперпосилання Hyperlink, мітку Label, список ListView, меню Menu, панель MenuBar, кнопку MenuButton, поле введення пароля PasswordField, індикатор ProgressBar, індикатор ProgressIndicator, перемикач RadioButton, панель ScrollPane, прокрутку ScrollBar, роздільник Separator, бігунок Slider, кнопку SplitMenuButton, панель SplitPane, таблицю TableView, панель із вкладками TabPane, багаторядкове поле TextArea, поле введення TextField, панель TitledPane, кнопку ToggleButton, групу ToggleGroup, панель ToolBar, вікно підказки Tooltip, дерево TreeView.

Пакет javafx.scene.chart забезпечує створення діаграм PieChart, AreaChart, BarChart, BubbleChart, LineChart та ScatterChart.

Пакет javafx.scene.image містить GUI-компонент зображення ImageView.

Пакет javafx.scene.layout надає панелі компонування AnchorPane, BorderPane, FlowPane, GridPane, HBox, StackPane, TilePane, VBox.

Пакет javafx.scene.media містить GUI-компоненти медіаконтенту MediaView та аудіоконтенту AudioClip.

Пакет javafx.scene.shape забезпечує малювання геометричних форм за допомогою таких GUI-компонентів, як Arc, Circle, CubicCurve, Ellipse, Line, Path, Polygon, Polyline, QuadCurve, Rectangle, SVGPath та Path.

Пакет javafx.scene.text містить GUI-компонент тексту Text.

Пакет javafx.scene.web забезпечує відображення HTML-контенту за допомогою GUI-компонента WebView та редагування HTML-контенту за допомогою GUI-компонента HTMLEditor.

Пакет javafx.scene містить групу Group та сцену Scene.

Пакет javafx.stage надає GUI-компоненти вікон Stage, Popup та FileChooser.

Кнопка Button. Компонент Button представлений класом javafx.scene.control.Button, екземпляр якого може бути створений за допомогою класу-фабрики ButtonBuilder або за допомогою

конструктора:

Button btn = New Button ();

Клас Button має:

– успадковані від класу javafx.scene.Node властивості: blendMode, boundsInLocal, boundsInParent, cacheHint, cache, clip, cursor, depthTest, disabled, disable, effect, eventDispatcher, focused, focusTraversable, hover, id, inputMethodRequests, layoutBounds, layout, layout, managed, mouseTransparent, onDragDetected, onDragDone, onDragDropped, onDragEntered, onDragExited, onDragOver, onInputMethodTextChanged, onKeyPressed, onKeyReleased, onKeyTyped, onMouseClicked, onMouseDragged, onMouseEntered, onMouseExited, onMouseMoved, onMousePressed, onMouseReleased, opacity, parent, pickOnBounds, pressed, rotate, rotationAxis, scaleX, scaleY, scaleZ, scene, style, translateX, translateY, translateZ, visible;

– успадкована від класу javafx.scene.Parent властивість needsLayout;

– успадковані від класу javafx.scene.control.Control властивості: contextMenu, height, maxHeight, maxWidth, minHeight, minWidth, prefHeight, prefWidth, skin, tooltip, width;

– успадковані від класу javafx.scene.control.Labeled властивості: alignment, contentDisplay, font, graphic, graphicTextGap, labelPadding, mnemonicParsing, textAlignment, textFill, textOverrun, text, underline, wrapText;

– успадковані від класу javafx.scene.control.ButtonBase властивості armed та onAction;

– власні властивості cancelButton та defaultButton.

Завдання

1. Для створення JavaFX-додатку з GUI-інтерфейсом, що містить кнопки *Button*, відкрийте середовище *NetBeans* з підтримкою платформи JavaFX 2.0 та в меню *Файл* оберіть *Створити проект* | *Java with Ant* | *JavaFX* | *JavaFX Application*, натисніть кнопку *Далі*, введіть ім’я проекту *JavaFXApplicationButton*, залишивши прапорець *Create Main Class*, та натисніть кнопку *Готово*.

У вікні редактора середовища NetBeans з’явиться код згенерованого класу *JavaFXApplicationButton*, який розширює клас Application.

```
package javafxapplicationbutton;
import javafx.application.Application;
import javafx.event.ActionEvent;
```

```
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class JavaFXApplicationButton extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say ‘Hello World’");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

2. Змінюємо заголовок основного вікна JavaFX-додатку:

```
primaryStage.setTitle("Тестування GUI-компонентів");
```

3. Змінюємо кореневий вузол графа сцени та на його основі екземпляр сцени:

```
Group root = new Group ();
Scene scene = new Scene(root, 400, 400, Color.LIGHTGREEN);
```

4. Змінюємо властивості кнопки *Button*, яка поміщається в лівий верхній кут основного вікна JavaFX-додатку, встановлюємо текст кнопки:

```
Button btn;
btn = new Button();
btn.setLayoutX(20);
```

```
btn.setLayoutY(20);
btn.setText("Тестувати властивості");
```

5. Клацаємо ліворуч від обробника подій кнопки, вибираємо «використовувати лямда-вираз» та змінюємо його:

```
btn.setOnAction(e -> {
System.out.println("Властивості, успадковані від класу Node:");
System.out.println("Властивість blendMode:
"+btn.blendModeProperty().getValue());
 System.out.println("Властивість boundsInLocal:
"+btn.boundsInLocalProperty().getValue());
});
```

6. Створюємо другу кнопку, яка розміщується нижче за першу. Для другої кнопки встановлюємо текст, стиль, розміри:

```
Button btnON = new Button();
  btnON.setLayoutX(20);
  btnON.setLayoutY(150);
  btnON.setText("Встановити властивості");
  btnON.setStyle("-fx-font: bold italic 12pt Arial;-fx-text-fill:
#660000;"
  + "-fx-background-color: #ff99ff; -fx-border-width: 3px; "
  + "-fx-border-radius: 30;-fx-background-radius: 30;"
  + "-fx-border-color: # 660066; ");
btnON.setPrefSize(250,30);
```

7. Створюємо обробник подій для другої кнопки, який встановлює такі властивості першої кнопки, як режим накладання, маска, курсор миші, візуальний ефект, управління компонуванням, прозорість, обертання, переміщення, масштабування, розміри, підказка, значок, стиль, вирівнювання, підкреслення тексту, перенесення рядків, прив’язка до клавіші активації, задній план:

```
btnON.setOnAction(e->{
  btn.setBlendMode(BlendMode.DARKEN);
  javafx.scene.shape.Circle clip = new
javafx.scene.shape.Circle(75,53,80);
  //btn.setClip(clip);
  btn.setCursor(Cursor.CLOSED_HAND);
  DropShadow effect=new DropShadow();
  effect.setOffsetX(10);
  effect.setOffsetY(10);
  btn.setEffect(effect);
```

```
        //btn.setManaged(false);
        //btn.setMouseTransparent(true);
        btn.setOpacity(0.5);
        btn.setRotate(10);
        btn.setLayoutX(80);
        btn.setScaleX(1.8);
        btn.setLayoutY(170);
        btn.setTranslateZ(-50);
        btn.setPrefSize(150,100);
        btn.setTooltip(new Tooltip("Це кнопка тестування властивостей
класу Button"));
        Image im = new
Image(this.getClass().getResource("image.png").toString());
        ImageView imv = new ImageView(im);
        imv.setFitHeight(50);
        imv.setFitWidth(50);
        btn.setGraphic(imv);
        btn.setStyle("-fx-font: bold italic 8pt Helvetica;");
        //btn.setFont(Font.font("Helvetica", FontWeight.BOLD,
        // FontPosture.ITALIC, 10));
        btn.setAlignment(Pos.CENTER);
        btn.setContentDisplay(ContentDisplay.RIGHT);
        btn.setUnderline(true);
        btn.setWrapText(true);
        //btn.setCancelButton(true);
        //btn.toBack();
    });
```

8. Скачуємо з інтернету рисунок із зображенням смайлика, зберігаємо його як *image.png* у папці, де розміщений файл JavaFXApplicationButton.java.

9. Додаємо другу кнопку в кореневий вузол:

```
root.getChildren().add(btnON);
```

10. Запускаємо створений JavaFX-додаток, клацнувши на піктограмі *Виконати*.

В результаті можна побачити дві кнопки (рис. 1), натискання однієї з яких дозволить вивести в консоль значення властивостей цієї кнопки, а натискання іншої кнопки призведе до зміни першої кнопки (рис. 2).

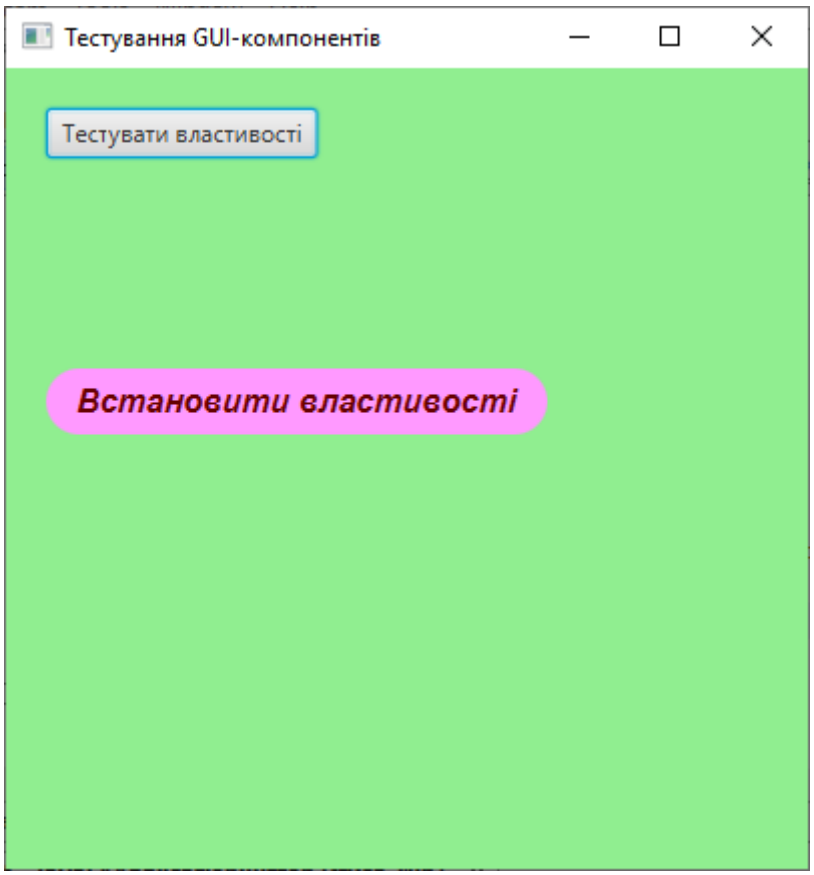


Рис. 1. JavaFX-додаток з GUI-інтерфейсом, що містить дві кнопки

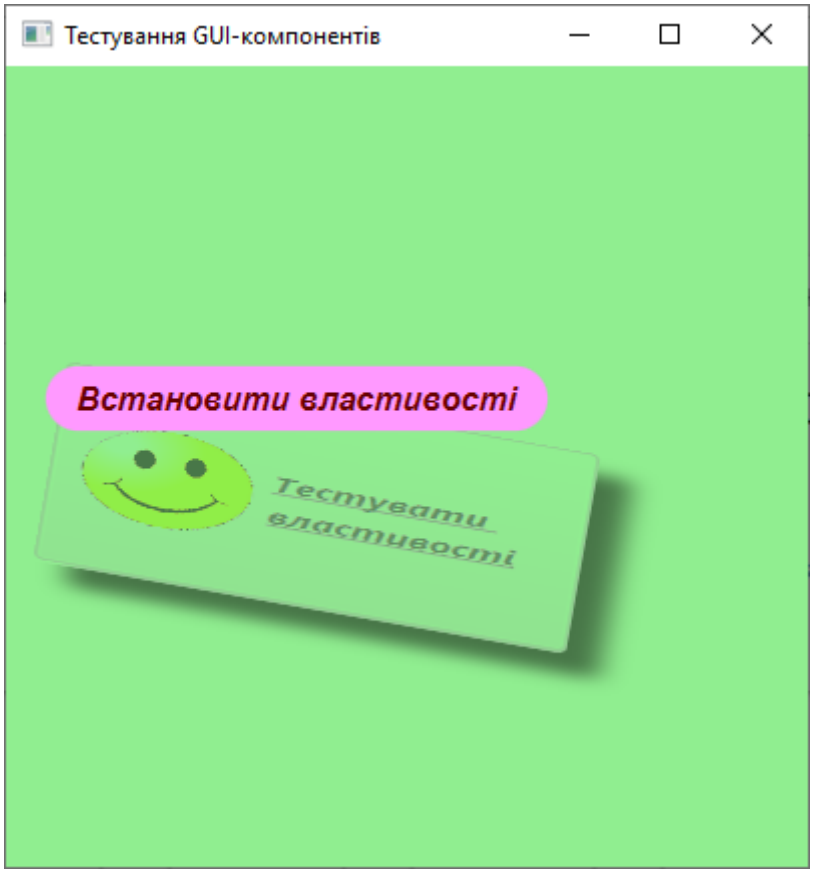


Рис. 2. Зміна властивостей першої кнопки за допомогою натискання другої кнопки

Натиснувши кнопку *Тестувати властивості* до її зміни та після зміни кнопки, у вікні *Виведення* середовища NetBeans можна побачити початкові та змінені значення властивостей кнопки.

Для звіту потрібно подати:

zip-файл, який містить проект *JavaFXApplicationButton*.

Примітка. Заголовок основного вікна повинен містити прізвище та ім’я студента-виконавця.

Контрольні запитання

1. Яка модель програмування додатків на платформі JavaFX?
2. Яке призначення інтерфейсу JavaFX API?
3. Яке призначення компоненту *Button*?
4. Які основні властивості притаманні компоненту *Button*?

Лабораторна робота 17. GUI-компонент CheckBox

Мета: набуття вмінь та навичок створення JavaFX-додатків, які містять компонент *CheckBox*.

Теоретичні відомості

Компонент *CheckBox* представлений класом javafx.scene.control.CheckBox, екземпляр якого може бути створений за допомогою класу-фабрики *CheckBoxBuilder* або за допомогою одного з конструкторів:

```
CheckBox ckb = new CheckBox();
CheckBox ckb = new CheckBox("[текст]");
```

Клас CheckBox має:

– успадковані від класу javafx.scene.Node властивості: blendMode, boundsInLocal, boundsInParent, cacheHint, cache, clip, cursor, depthTest, disabled, disable, effect, eventDispatcher, focused, focusTraversable, hover, id, inputMethodRequests, layoutBounds, layoutX, layoutY, managed, mouseTransparent, onDragDetected, onDragDone, onDragDropped, onDragEntered, onDragExited, onDragOver, onInputMethodTextChanged, onKeyPressed, onKeyReleased, onKeyTyped, onMouseClicked, onMouseDragged, onMouseEntered, onMouseExited, onMouseMoved, onMousePressed, onMouseReleased, opacity, parent, pickOnBounds, pressed, rotate, rotationAxis, scaleX, scaleY, scaleZ, scene, style, translateX, translateY, translateZ, visible;

– успадкована від класу javafx.scene.Parent властивість needsLayout;

– успадковані від класу javafx.scene.control.Control властивості: contextMenu, height, maxHeight, maxWidth, minHeight, minWidth, prefHeight, prefWidth, skin, tooltip, width;

– успадковані від класу javafx.scene.control.Labeled властивості: alignment, contentDisplay, font, graphic, graphicTextGap, labelPadding, mnemonicParsing, textAlignment, textFill, textOverrun, text, underline, wrapText;

– успадковані від класу javafx.scene.control.ButtonBase властивості armed та onAction;

– власні властивості indeterminate, selected і allowIndeterminate.

Завдання

1. Для створення JavaFX-додатку з GUI-інтерфейсом, що містить прапорець *CheckBox*, відкрийте середовище *NetBeans* з підтримкою платформи JavaFX 2.0 та в меню *Файл* оберіть *Створити проект* | *Java with Ant* | *JavaFX* | *JavaFX Application*, натисніть кнопку *Далі*, введіть ім’я проекту *JavaFXApplicationCheckBox*, залишивши прапорець *Create Main Class*, та натисніть кнопку *Готово*.

У вікні редактора середовища NetBeans з’явиться код згенерованого класу *JavaFXApplicationCheckBox*, який розширює клас Application.

```
package javafxapplicationbutton;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class JavaFXApplicationCheckBox extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say ‘Hello World’");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });
```

```
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

2. У методі *start* вилучіть весь код.

3. Установіть заголовок основного вікна JavaFX-додатку:

```
primaryStage.setTitle("Тестування GUI-компонентів: CheckBox");
```

4. Установіть кореневий вузол графа сцени та на його основі екземпляр сцени:

```
Group root = new Group();
Scene scene = new Scene(root, 400, 400, Color.LIGHTGREEN);
```

5. Створіть екземпляр прапорця CheckBox із текстом. Прапорець розміщується в лівому верхньому куті основного вікна JavaFX-додатку. Для нього визначаємо обробник подій натискання кнопки миші таким чином, що при натисканні кнопкою миші на прапорці в консоль виводяться значення властивостей CheckBox:

```
CheckBox ckb = new CheckBox("Тестувати властивості");
// ckb=CheckBoxBuilder.create().build();
ckb.setLayoutX(20);
ckb.setLayoutY(20);
ckb.setOnMousePressed(e->{
System.out.println("Властивості, успадковані від класу Node:");
  System.out.println("Властивість blendMode: "+
ckb.blendModeProperty().getValue());
  System.out.println("Властивість boundsInLocal: "+
ckb.boundsInLocalProperty().getValue());
});
```

6. Створіть кнопку, яка розміщується нижче за прапорець, при натисканні якої змінюються властивості прапорця CheckBox. Для кнопки встановлюється текст, стиль, переважні розміри:

```
Button btnON = new Button();
btnON.setLayoutX(20);
```

```
        btnON.setLayoutY(100);
    btnON.setText("Установити властивості");
    btnON.setStyle("-fx-font: bold italic 12pt Arial;"
        + "-fx-text-fill: white;-fxbackground-color: #0000cc;"
        + "-fx-border-width: 3px; "
        + "-fx-border-color:#6699ff #000066 #000066 #6699ff;" );
    btnON.setPrefSize(200,30);
```

7. Для кнопки встановлюємо обробник подій. В ньому, якщо встановлено прапорець CheckBox, визначаються такі властивості CheckBox, як режим накладання, маска, курсор миші, візуальний ефект, управління компонуванням, активація мишею, прозорість, переміщення, масштабування, переважні розміри, підказка, значок, стиль, вирівнювання, підкреслення тексту, перенесення рядків, встановлення трьох станів прапорця, задній план та поворот:

```
    btnON.setOnAction(e->{
      if(ckb.selectedProperty().getValue()==true){
        ckb.setBlendMode(BlendMode.HARD_LIGHT);
        Rectangle clip = new Rectangle(0,15,15,20);
        //ckb.setClip(clip);
        ckb.setCursor(Cursor.CROSSHAIR);
        DropShadow effect=new DropShadow();
        effect.setOffsetX(5);
        effect.setOffsetY(10);
        ckb.setEffect(effect);
        //ckb.setManaged(false);
        //ckb.setMouseTransparent(true);
        //ckb.setOpacity(0.5);
        ckb.setLayoutX(50);
        ckb.setTranslateZ(-50);
        ckb.setScaleX(1.8);
        ckb.setPrefSize(150,50);
        ckb.setTooltip(new Tooltip ("Це перемикач тестування
властивостей класу CheckBox"));
        Image im = new
Image(this.getClass().getResource("image.png").toString());
        ImageView imv = new ImageView(im);
        imv.setFitHeight(50);
        imv.setFitWidth(50);
        ckb.setGraphic(imv);
```

```
            ckb.setStyle("-fx-font: bold italic 10pt Helvetica;");
            ckb.setAlignment(Pos.CENTER);
            ckb.setContentDisplay(ContentDisplay.RIGHT);
            ckb.setUnderline(true);
            ckb.setWrapText(true);
            ckb.setAllowIndeterminate(true);
            //ckb.toBack();
            //ckb.setTranslateY(50);
        }
    });
```

8. Скачайте з інтернету рисунок із зображенням смайлика та збережіть його як *image.png* у папці, де розміщений файл JavaFXApplicationCheckBox.java.

9. Прапорець і кнопку додайте в кореневий вузол графа сцени, для об'єкта Stage встановіть створену сцену, і об'єкт Stage встановіть видимим:

```
root.getChildren().add(btnON);
root.getChildren().add(ckb);
primaryStage.setScene(scene);
primaryStage.show();
```

10. Запустіть створений JavaFX-додаток, клацнувши правою кнопкою миші на піктограмі *Очистити та побудувати проект* (Shift+F11), а потім – *Виконати проект* (F6).

У результаті можна побачити прапорець та кнопку (рис. 3.3). При клацанні мишею на прапорці буде змінено його стан і в консоль буде виведено значення властивостей прапорця, а при натисканні кнопки стан прапорця зміниться (рис. 3.4).

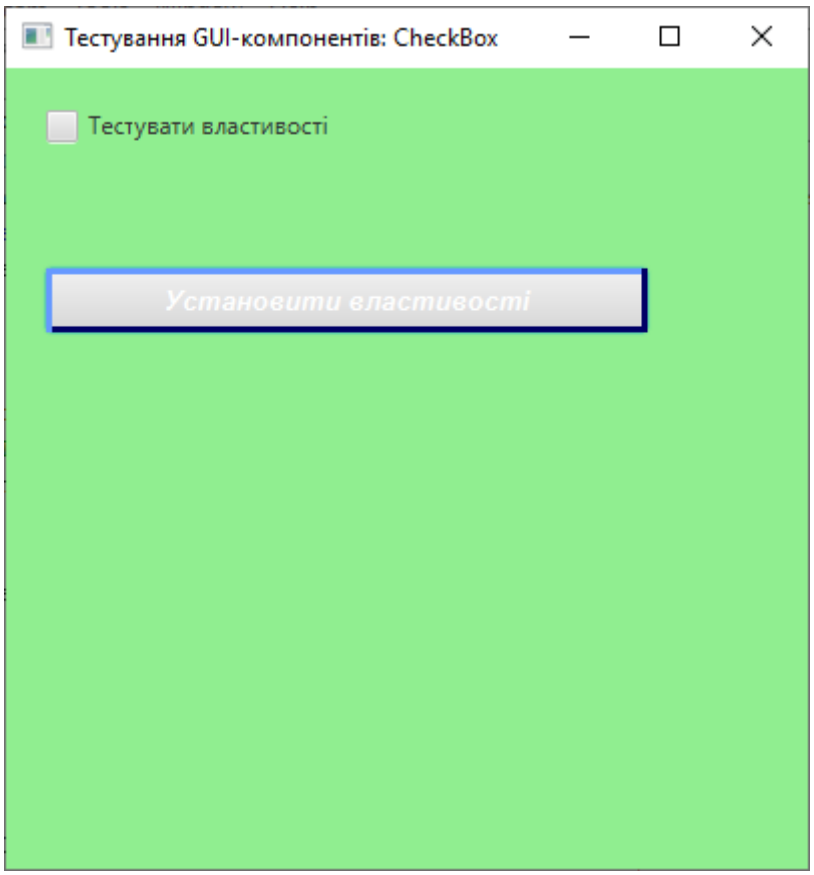


Рис. 3.3. JavaFX-прикладання з GUI-інтерфейсом, що містить прапорець і кнопку

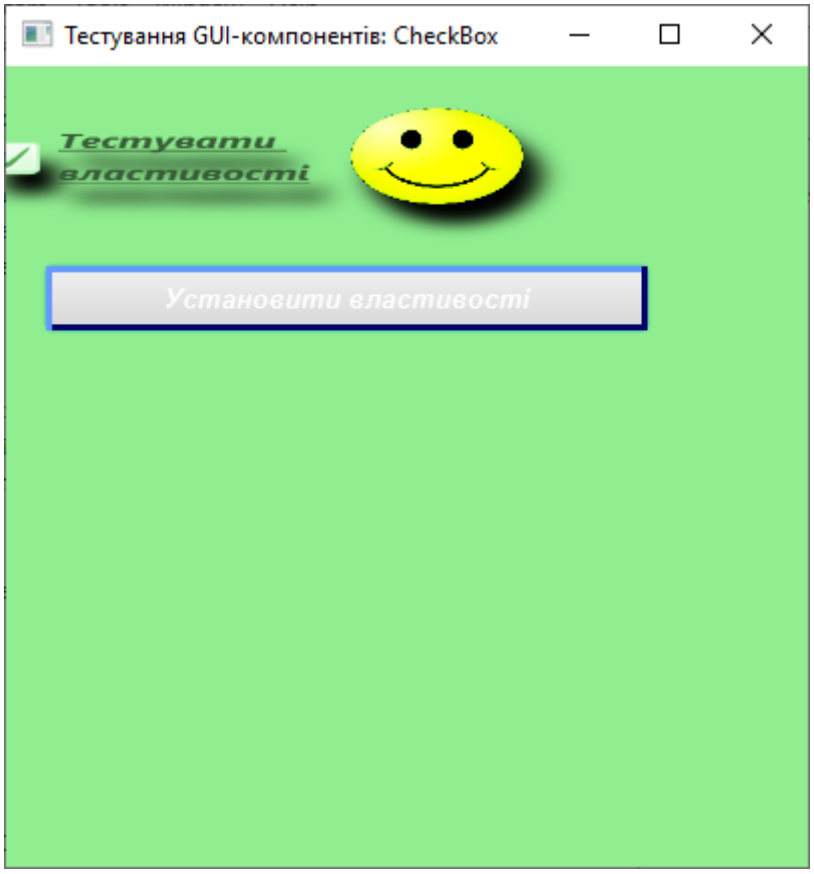


Рис. 3.4. Зміна властивостей прапорця за допомогою натискання

кнопки

Властивості компонента *CheckBox*, успадковані від класів *ButtonBase*, *Labeled*, *Control*, *Parent* і *Node*, аналогічні тим самим властивостям, які має компонент *Button*.

Окрім методу setOnAction() для визначення обробника подій компонента можна використовувати методи setOnMousePressed(), setOnMouseClicked() та setOnMouseReleased().

При використанні методу setOnAction() властивість armed набуває значення true, а властивість pressed – значення false. При використанні методу setOnMousePressed() властивості armed та pressed приймають значення true, а при застосуванні методів setOnMouseClicked() і setOnMouseReleased() ці властивості набувають значення false.

Крім того, обробку подій зміни стану перемикача можна здійснити, використовуючи JavaFX Beans-властивість компонента CheckBox:

```
ckb.selectedProperty().addListener(new ChangeListener<Boolean>()
{
public void changed(ObservableValue<? extends Boolean> ov,
Boolean old_val, Boolean new_val) {
. . .
} });
```

Відмінність набору властивостей компонента *CheckBox* від набору властивостей компонента *Button* полягає в наявності властивостей самого класу javafx.scene.control.CheckBox *indeterminate*, *selected* і *allowIndeterminate*.

Властивість selected приймає значення *true* чи *false* залежно від того, чи встановлено прапорець чи ні.

Якщо властивість *allowIndeterminate* встановити рівним true, тоді замість галочки можна поставити межу біля прапорця. У цьому властивість *indeterminate* прийме значення *true*, а властивість *selected* – значення *false*.

Для звіту потрібно подати:

zip-файл, який містить проект *JavaFXApplicationCheckBox*.

Примітка. Заголовок основного вікна повинен містити прізвище та ім’я студента-виконавця.

Контрольні запитання

1. Яке призначення компоненту *CheckBox*?

2. Як створити екземпляр компоненту *CheckBox*?
3. Які основні властивості має компонент *CheckBox?*

Лабораторна робота 18. GUI-компонент RadioButton

Мета: набуття вмінь та навичок створення JavaFX-додатків, які містять компонент *RadioButton*.

Теоретичні відомості

Компонент RadioButton представлений класом javafx.scene.control.RadioButton, екземпляр якого може бути створений за допомогою класу-фабрики RadioButtonBuilder або за допомогою одного з конструкторів:

RadioButton btn = New RadioButton();

RadioButton btn = new RadioButton("[текст]");

Клас RadioButton має:

– успадковані від класу javafx.scene.Node властивості: blendMode, boundsInLocal, boundsInParent, cacheHint, cache, clip, cursor, depthTest, disabled, disable, effect, eventDispatcher, focused, focusTraversable, hover, id, inputMethodRequests, layoutBounds, layoutX, layoutY, managed, mouseTransparent, onDragDetected, onDragDone, onDragDropped, onDragEntered, onDragExited, onDragOver, onInputMethodTextChanged, onKeyPressed, onKeyReleased, onKeyTyped, onMouseClicked, onMouseDragged, onMouseEntered, onMouseExited, onMouseMoved, onMousePressed, onMouseReleased, opacity, parent, pickOnBounds, pressed, rotate, rotationAxis, scaleX, scaleY, scaleZ, scene, style, translateX, translateY, translateZ, visible;

– успадкована від класу javafx.scene.Parent властивість needsLayout;

– успадковані від класу javafx.scene.control.Control властивості: contextMenu, height, maxHeight, maxWidth, minHeight, minWidth, prefHeight, prefWidth, skin, tooltip, width;

– успадковані від класу javafx.scene.control.Labeled властивості: alignment, contentDisplay, font, graphic, graphicTextGap, labelPadding, mnemonicParsing, textAlignment, textFill, textOverrun, text, underline, wrapText;

– успадковані від класу javafx.scene.control.ButtonBase властивості armed та onAction;

– успадковані від класу javafx.scene.control.ToggleButton властивості selected та toggleGroup.

Властивості компонента RadioButton аналогічні властивостям

компонента ToggleButton.

Об'єднання перемикачів RadioButton у групу ToggleGroup відрізняється від об'єднання кнопок ToggleButton у групу ToggleGroup.

У групі перемикачів RadioButton щонайменше один елемент повинен бути у вибраному стані, і вибір іншого перемикача скасовує вибір решти. Таким чином, у групі перемикачів RadioButton постійно вибрано лише один елемент, і його не можна перевести у невибраний стан.

У групі кнопок ToggleButton при натисканні однієї з кнопок решта всіх кнопок групи автоматично віджимається, однак натиснуту кнопку можна віджати.

Екземпляр класу ToggleGroup можна створити за допомогою класу-фабрики ToggleGroupBuilder або за допомогою конструктора:

ToggleGroup tgroup = New ToggleGroup();

Клас ToggleGroup має властивість selectedToggle, що вказує на обраний в даний момент елемент групи.

Завдання

1. Відкрийте середовище *NetBeans* з підтримкою платформи JavaFX 2.0. Для створення JavaFX-додатку з GUI-інтерфейсом, що містить перемикач *RadioButton*, в меню *Файл* оберіть *Створити проект | Java with Ant | JavaFX | JavaFX Application*, натисніть кнопку *Далі*, введіть ім’я проекту *JavaFXApplicationRadioButton*, залишивши прапорець *Create Main Class*, та натисніть кнопку *Готово*.

У вікні редактора середовища NetBeans з’явиться код згенерованого класу *JavaFXApplicationRadioButton*, який розширює клас Application.

```
package javafxapplicationbutton;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class JavaFXApplicationRadioButton extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say ‘Hello World’");
```

```
            btn.setOnAction(new EventHandler<ActionEvent>() {
                @Override
                public void handle(ActionEvent event) {
                    System.out.println("Hello World!");
                }
            });
            StackPane root = new StackPane();
            root.getChildren().add(btn);
            Scene scene = new Scene(root, 300, 250);
            primaryStage.setTitle("Hello World!");
            primaryStage.setScene(scene);
            primaryStage.show();
        }
        public static void main(String[] args) {
            launch(args);
        }
    }
```

2. У методі *start* вилучіть весь код.

3. Створіть кореневий вузол графа сцени та на його основі екземпляр сцени, який встановлюється для об’єкта Stage. Установіть заголовок основного вікна JavaFX-додатку “Тестування GUI-компонентів: RadioButton” та зробіть видимим об’єкт Stage:

```
Group root = new Group();
Scene scene = new Scene(root, 400, 400, Color.LIGHTGREEN);
primaryStage.setScene(scene);
primaryStage.setTitle("Тестування GUI-компонентів:
RadioButton");
primaryStage.show();
```

4. Створіть групу ToggleGroup та перший екземпляр кнопки RadioButton з текстом. Кнопку помістіть у лівий верхній кут основного вікна JavaFX-програми. Для неї визначте такі властивості, як режим накладання, курсор миші, переважні розміри, спливаюча підказка, значок, стиль, вирівнювання, перенесення рядків, групу, фокус та обробник зміни значення selected кнопки, в якому змінюється загальний колір сцени:

```
ToggleGroup tgroup=new ToggleGroup();
RadioButton btnOn;
btnOn = new RadioButton("Mozilla Firefox");
btnOn.setLayoutX(20);
```

```
        btnOn.setLayoutY(20);
        btnOn.setBlendMode(BlendMode.MULTIPLY);
        btnOn.setCursor(Cursor.CLOSED_HAND);
        btnOn.setPrefSize(200,80);
        btnOn.setTooltip(new Tooltip("Це кнопка вибору Mozilla
Firefox"));
        Image imOn = new
Image(this.getClass().getResource("imageOn.jpg").toString());
        ImageView imvOn = new ImageView(imOn);
        imvOn.setFitHeight(50);
        imvOn.setFitWidth(50);
        btnOn.setGraphic(imvOn);
        btnOn.setStyle("-fx-font: bold italic 12pt Georgia;");
        btnOn.setAlignment(Pos.CENTER);
        btnOn.setContentDisplay(ContentDisplay.RIGHT);
        btnOn.setTextAlignment(TextAlignment.CENTER);
        btnOn.setGraphicTextGap(10);
        btnOn.setWrapText(true);
        btnOn.setToggleGroup(tgroup);
        btnOn.requestFocus();
        btnOn.selectedProperty().addListener((javafx.beans.value.Observabl
eValue<? extends Boolean> ov, Boolean old_val, Boolean new_val) -> {
            if (new_val.equals(Boolean.TRUE))
                scene.setFill(Color.web("#fff8dc"));
        });
```

5. Створіть другий екземпляр кнопки RadioButton із текстом. Другу кнопку помістіть під першою кнопкою, і для неї визначте також такі властивості, як режим накладання, курсор миші, переважні розміри, підказка, значок, стиль, вирівнювання, перенесення рядків, група та обробник зміни значення властивості selected кнопки, в якому змінюється загальний колір сцени:

```
        RadioButton btnOf;
        btnOf = new RadioButton("Internet Explorer");
        btnOf.setLayoutX(20);
        btnOf.setLayoutY(100);
        btnOf.setBlendMode(BlendMode.MULTIPLY);
        btnOf.setCursor(Cursor.CLOSED_HAND);
        btnOf.setPrefSize(200,80);
        btnOf.setTooltip(new Tooltip("Це кнопка вибору Internet
```

```
Explorer"));
        Image imOf = new
Image(this.getClass().getResource("imageOf.jpg").toString());
        ImageView imvOf = new ImageView(imOf);
        imvOf.setFitHeight(50);
        imvOf.setFitWidth(50);
        btnOf.setGraphic(imvOf);
        btnOf.setStyle("-fx-font: bold italic 12pt Georgia;" );
        btnOf.setAlignment(Pos.CENTER);
        btnOf.setContentDisplay(ContentDisplay.RIGHT);
        btnOf.setTextAlignment(TextAlignment.CENTER);
        btnOf.setGraphicTextGap(10);
        btnOf.setWrapText(true);
        btnOf.setToggleGroup(tgroup);
        btnOf.selectedProperty().addListener((javafx.beans.value.Observable
Value<? extends Boolean> ov, Boolean old_val, Boolean new_val) -> {
            if (new_val.equals(Boolean.TRUE))
                scene.setFill(Color.web("#99ffff"));
        });
```

6. Скачайте з інтернету рисунки, назвавши як *imageOn.jpg* та *imageOf.jpg* і збережіть їх у папці, де розміщений файл JavaFXApplicationRadioButton.java.

7. Створені кнопки додайте в кореневий вузол графа сцени:

```
root.getChildren().add(btnOn);
root.getChildren().add(btnOf);
```

8. Запустіть створений JavaFX-додаток, клацнувши правою кнопкою миші на піктограмі *Очистити та побудувати проект* (Shift+F11), а потім – *Виконати проект* (F6).

У результаті можна буде побачити дві кнопки RadioButton (рис. 1), при виборі яких змінюється колір сцени. Спочатку при відкритті вікна програми жодна з кнопок не вибрана, однак фокус наведено на першу кнопку. Після вибору однієї з кнопок вже не можна перевести сцену у вихідний стан із вихідним кольором та двома невибраними кнопками.

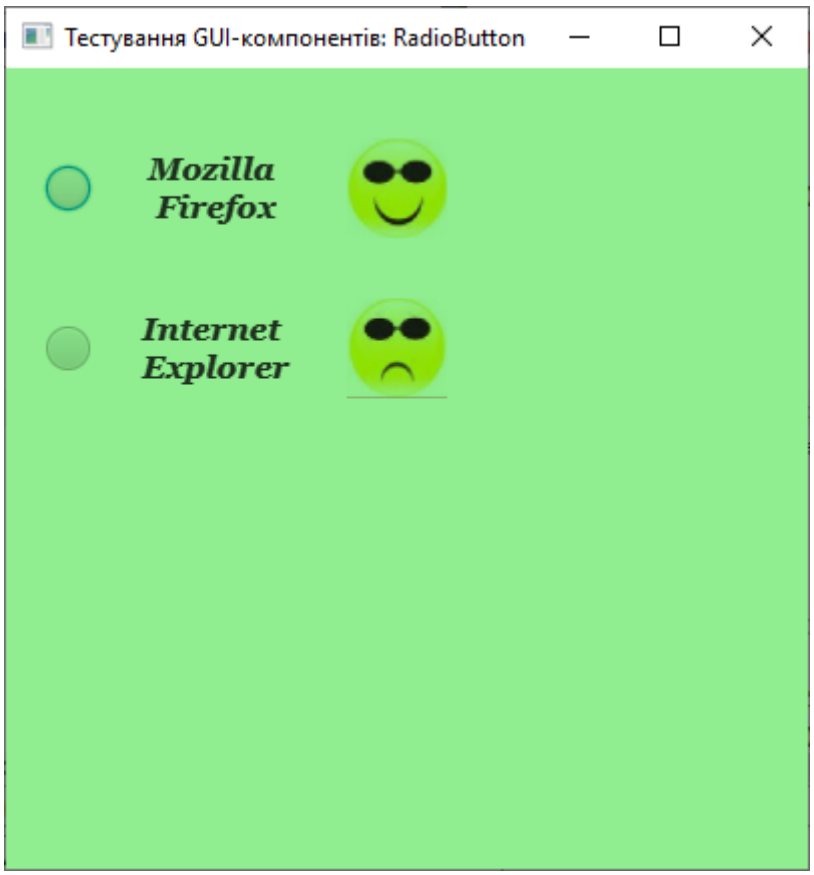


Рис. 1. JavaFX-додаток з GUI-інтерфейсом, що містить кнопки RadioButton

Контрольні запитання

1. Яке призначення компонента *RadioButton*?
2. Як створити екземпляр компонента *RadioButton*?
3. Які основні властивості має компонент *RadioButton?*

Лабораторна робота 19. Діаграми. Кругова діаграма PieChart

Мета: набуття вмінь та навичок створення JavaFX-додатків, які містять кругову діаграму *PieChart*.

Теоретичні відомості

Платформа JavaFX 2.0 забезпечує створення двовимірних діаграм за допомогою базового класу javafx.scene.chart.Chart та його підкласів PieChart, AreaChart, BarChart, BubbleChart, LineChart та ScatterChart.

Клас Chart має такі властивості:

– успадковані від класу javafx.scene.Node властивості: blendMode, boundsInLocal, boundsInParent, cacheHint, cache, clip, cursor, depthTest, disabled, disable, effect, eventDispatcher, focused, focusTraversable, hover, id, inputMethodRequests, layoutBounds, layoutX, layoutY, managed, mouseTransparent, onDragDetected, onDragDone,

onDragDropped, onDragEntered, onDragExited, onDragOver, onInputMethodTextChanged, onKeyPressed, onKeyReleased, onKeyTyped, onMouseClicked, onMouseDragged, onMouseEntered, onMouseExited, onMouseMoved, onMousePressed, onMouseReleased, opacity, parent, pickOnBounds, pressed, rotate, rotationAxis, scaleX, scaleY, scaleZ, scene, style, translateX, translateY, translateZ, visible;

– успадковану від класу javafx.scene.Parent властивість needsLayout;

– успадковані від класу javafx.scene.layout.Region властивості: height, maxHeight, maxWidth, minHeight, minWidth, padding, prefHeight, prefWidth, snapToPixel, width;

– власні властивості: animated, legend, legendSide, legendVisible, title, titleSide.

За допомогою властивості title встановлюється заголовок діаграми, а за допомогою властивості titleSide – розташування заголовку вгорі, внизу, праворуч або ліворуч самої діаграми.

Властивості legendSide та legendVisible визначають розташування та видимість панелі пояснень до діаграми.

Якщо властивість animated встановити зі значенням true, тоді діаграма динамічно реагуватиме на зміни своїх властивостей.

У той час як клас PieChart безпосередньо розширює клас Chart, класи AreaChart, BarChart, BubbleChart, LineChart та ScatterChart є підкласами класу javafx.scene.chart.XYChart<X,Y>, що розширює клас Chart.

Клас XYChart є базовим класом для класів, що представляють діаграми з двома осями, і має, крім успадкованих від класу Chart властивостей, власні властивості: data, verticalGridLinesVisible, horizontalGridLinesVisible, alternativeColumnFillVisible, alternativeRowFillVisible, verticalZeroLineVisible, horizontalZeroLineVisible.

Властивості alternativeColumnFillVisible і alternativeRowFillVisible визначають виділення через один у сітці діаграми стовпців і рядків, якщо обидві властивості встановлені зі значенням true, сітка діаграми схожа на шахівницю.

Властивості verticalGridLinesVisible та horizontalGridLinesVisible визначають відображення вертикальних та горизонтальних ліній сітки діаграми, а властивості verticalZeroLineVisible та horizontalZeroLineVisible – відображення додаткових вертикальних та горизонтальних ліній нульових позначок за умови відображення

вертикальних та горизонтальних ліній сітки діаграми.

Властивість data визначає набір javafx.collections.ObservableList<XYChart.Series<X,Y>> даних, з яких формується сама діаграма, розмітка осей діаграми, набір підписів до міток осей діаграми та вміст легенди. Поповнити набір ObservableList<XYChart.Series<X,Y>> даних діаграми можна за допомогою методу getData().addAll() класу javafx.scene.chart.XYChart<X,Y>.

Дані діаграми XYChart є класом javafx.scene.chart.XYChart.Series<X,Y>, екземпляр якого можна створити за допомогою одного з конструкторів:

XYChart.Series series= new XYChart.Series();

XYChart.Series series = new XYChart.Series(ObservableList<XYChart.Data<X,Y>> data);

XYChart.Series series = new XYChart.Series(java.lang.String name, ObservableList<XYChart.Data<X,Y>> data);

Клас XYChart.Series представляє серію даних діаграми XYChart і має властивості: chart, data, name, node.

Властивість name визначає назву серії, що відображається у легенді до діаграми.

Після того, як серія додається в набір даних діаграми XYChart, формується вузол Node, який відповідає за відображення серії, і за допомогою властивості node можна отримати доступ до властивостей цього вузла.

Властивість data визначає набір javafx.collections.ObservableList<XYChart.

Data<X,Y>> даних серії XYChart.Series, поповнити який можна за допомогою методу getData().addAll() класу javafx.scene.chart.XYChart.Series<X,Y>.

Дані серії XYChart.Series представлені класом javafx.scene.chart.XYChart.Data<X,Y>, екземпляр якого можна створити за допомогою одного з конструкторів:

XYChart.Data data = new XYChart.Data();

XYChart.Data data = new XYChart.Data (X xValue, Y yValue);

XYChart.Data data= new XYChart.Data(X xValue, Y yValue,

java.lang.Object extraValue);

Клас XYChart.Data представляє дані серії XYChart.Series та має властивості: extraValue, node, xValue, yValue.

Властивості xValue та yValue визначають значення елемента

даних по осі x та y, властивість extraValue — додаткове значення, наприклад радіус для бульбашкової діаграми.

Після того, як елемент даних додається в набір даних діаграми XYChart, формується вузол Node, який відповідає за відображення елемента даних, і за допомогою властивості node можна отримати доступ до властивостей цього вузла. До додавання елемента даних до набору даних діаграми XYChart можна визначити свій вузол Node, який відповідає за відображення елемента даних.

Осі діаграми XYChart представлені базовим класом javafx.scene.chart.Axis<T> та його підкласами CategoryAxis та NumberAxis.

Клас Axis має наступні властивості:

 успадковані від класу javafx.scene.Node властивості: blendMode, boundsInLocal, boundsInParent, cacheHint, cache, clip, cursor, depthTest, disabled, disable, effect, eventDispatcher, focused, focusTraversable, hover, id, inputMethodRequests, layoutBounds, layoutX, layoutY, managed, mouseTransparent, onDragDetected, onDragDone, onDragDropped, onDragEntered, onDragExited, onDragOver, onInputMethodTextChanged, onKeyPressed, onKeyReleased, onKeyTyped, onMouseClicked, onMouseDragged, onMouseEntered, onMouseExited, onMouseMoved, onMousePressed, onMouseReleased, opacity, parent, pickOnBounds, pressed, rotate, rotationAxis, scaleX, scaleY, scaleZ, scene, style, translateX, translateY, translateZ, visible;

успадкована від класу javafx.scene.Parent властивість needsLayout;

успадковані від класу javafx.scene.layout.Region властивості: height, maxHeight, maxWidth, minHeight, minWidth, padding, prefHeight, prefWidth, snapToPixel, width;

власні властивості: animated, autoRanging, label, side, tickLabelFill, tickLabelFont, tickLabelGap, tickLabelRotation, tickLabelsVisible, tickLength, tickMarkVisible.

Властивість label визначає підпис осі, а властивість side – сторону відображення осі.

За допомогою характеристики autoRanging встановлюється автоматичний вибір діапазону значень осі на основі набору даних діаграми.

Якщо властивість animated встановити рівним true, тоді вісь динамічно реагуватиме на зміни своїх властивостей.

Властивості tickLabelFill, tickLabelFont, tickLabelGap,

tickLabelRotation, tickLabelsVisible, tickLength та tickMarkVisible визначають колір підписів до міток осі, шрифт підписів до міток осі, інтервал між лініями міток на осі та підписами до міток осі, поворот підписів до міток осі, відображення підписів міток осі, довжину ліній міток на осі та відображення міток осі.

Клас CategoryAxis є вісь, що відображає дискретні рядкові значення окремих категорій. Примірник класу CategoryAxis може бути створений за допомогою класу-фабрики CategoryAxisBuilder або за допомогою одного з конструкторів:

CategoryAxis axis = new CategoryAxis();

CategoryAxis axis = new
CategoryAxis(ObservableList<java.lang.String> categories);

Набір рядкових значень ObservableList<java.lang.String> осі CategoryAxis може бути створений за допомогою статичного методу observableArrayList() класу javafx.collections.FXCollections.

Клас CategoryAxis має, окрім успадкованих від класу Axis<T> властивостей, власні властивості category початком осі та першою міткою.

Клас NumberAxis представляє числову вісь діаграми XYChart.

Примірник класу NumberAxis може бути створений за допомогою класу-фабрики NumberAxisBuilder або за допомогою одного з конструкторів:

NumberAxis = NumberAxis();

NumberAxis = NumberAxis(double lowerBound, double upperBound, double tickUnit);

NumberAxis axis = new NumberAxis(java.lang.String axisLabel, double lowerBound, double upperBound, double tickUnit);

Клас NumberAxis має, крім успадкованих від класу Axis<T> властивостей, такі властивості:

– успадковані від класу javafx.scene.chart.ValueAxis<T> властивості: lowerBound, minorTickCount, minorTickLength, minorTickVisible, scale, tickLabelFormatter, upperBound;

– власні властивості forceZeroInRange та tickUnit.

Властивості lowerBound, minorTickCount, minorTickLength, minorTickVisible, tickLabelFormatter та upperBound дозволяють визначити мінімальне значення осі, кількість допоміжних міток, довжину допоміжних міток, відображення допоміжних міток, форматування підписів до міток осі та максимальне

Властивості forceZeroInRange та tickUnit визначають включення

нульової мітки у видимий діапазон при його автоналаштуванні та інтервал між головними мітками осі.

Кругова діаграма PieChart

Діаграма PieChart представлена класом javafx.scene.chart.PieChart, екземпляр якого можна створити за допомогою класу-фабрики PieChartBuilder або за допомогою одного з конструкторів:

PieChart chart = new PieChart();

PieChart chart = new PieChart(ObservableList<PieChart.Data> data);

Клас PieChart представляє кругову (секторну) діаграму і має, крім успадкованих від класу Chart властивостей, власні властивості: clockwise, data, labelLineLength, labelsVisible, startAngle.

Властивість data визначає набір javafx.collections.ObservableList<PieChart.Data> даних, з яких формуються сектори діаграми, підписи до них та вміст панелі пояснень до діаграми.

Набір даних ObservableList<PieChart.Data> діаграми PieChart може бути створений за допомогою статичного методу observableArrayList() класу javafx.collections.FXCollections. При цьому дані діаграми PieChart є класом javafx.scene.chart.PieChart.Data, екземпляр якого може бути створений за допомогою конструктора public PieChart.Data(java.lang.String name, double value), де name – це підпис до сектора діаграми, а value – частка сектора діаграми.

Отримати доступ до властивостей вузла Node, що є сектором PieChart.Data, дозволяє метод getNode() класу PieChart.Data, а до властивостей діаграми PieChart, до якої належить сектор – метод getChart() класу PieChart.Data.

За допомогою властивості clockwise класу PieChart встановлюється розташування секторів діаграми за годинниковою стрілкою, а за допомогою властивості startAngle – кут початку першого сектора діаграми.

Властивості labelsVisible і labelLineLength визначають відображення підписів до секторів діаграми та довжину лінії від сектора діаграми до підпису до сектора.

Завдання

1. Відкрийте середовище *NetBeans* з підтримкою платформи JavaFX 2.0. Для створення JavaFX-додатку з GUI-інтерфейсом, що містить кругову діаграму *PieChart*, в меню *Файл* оберіть *Створити проект* | *Java with Ant* | *JavaFX* | *JavaFX Application*, натисніть кнопку

Далі, введіть ім’я проекту *JavaFXApplicationPieChart*, залишивши прапорець *Create Main Class*, та натисніть кнопку *Готово*.

У вікні редактора середовища NetBeans з’явиться код згенерованого класу *JavaFXApplicationPieChart*, який розширює клас Application.

```
package javafxapplicationbutton;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class JavaFXApplicationPieChart extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say ‘Hello World’");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

2. У методі *start()* вилучіть весь код.

3. Створіть кореневий вузол графа сцени та на його основі екземпляр сцени, який встановлюється для об’єкта Stage. Установіть заголовок основного вікна JavaFX-додатку “Тестування GUI-

компонентів: PieChart” та зробіть видимим об’єкт Stage:

```
Group root = new Group();
Scene scene = new Scene(root, 600, 500, Color.BEIGE);
primaryStage.setScene(scene);
primaryStage.setTitle("Тестування GUI-компонентів: PieChart") ;
primaryStage.show();
```

4. Створіть кругову діаграму PieChart. Діаграму помістіть у лівий верхній кут, і для неї встановіть набір даних і такі властивості, як курсор миші, стиль, переважні розміри, динамічне реагування на зміну властивостей, заголовок та його розташування, відображення та розташування панелі пояснень до діаграми, розташування секторів за годинною стрілки, відображення підписів і довжина лінії до підпису, кут початку першого сектора, обробник клацання кнопкою миші на секторі діаграми, який відображає значення сектора, та обробник перетягування мишею, що повертає діаграму навколо своєї осі. Такий поворот відображення діаграми навколо своєї осі можливий, тому що властивість animated діаграми встановлено рівним true:

```
final ObservableList<PieChart.Data> pieChartData =
        FXCollections.observableArrayList(new
PieChart.Data("Долар США", 45.9),
        new PieChart.Data("Євро", 42.5),
        new PieChart.Data("Японська єна", 1.6),
        new PieChart.Data("Фунт стерлінгів", 9.2),
        new PieChart.Data("Канадський долар", 0.8));
        final PieChart chart = new PieChart(pieChartData);
        chart.setLayoutX(50);
        chart.setLayoutY(10);
        chart.setCursor(Cursor.CROSSHAIR);
        chart.setStyle("-fx-font:bold 14 Arial; -fx-text-fill:brown;");
        chart.setPrefSize(500, 400);
        chart.setAnimated(true);
        chart.setTitle("Розподіл валютних активів\n Ощадбанку по
валюті у 2022 р.");
        chart.setTitleSide(Side.TOP);
        chart.setLegendVisible(true);
        chart.setLegendSide(Side.BOTTOM);
        chart.setClockwise(true);
        chart.setLabelsVisible(true);
        chart.setLabelLineLength(20);
```

```
            chart.setStartAngle(150);
            final Popup popup = new Popup();
            popup.setAutoHide(true);
            final Label label = new Label("");
            label.setStyle("-fx-font: bold 20 Arial;-fx-text-fill:brown");
            popup.getContent().addAll(label);
            chart.getData().forEach(data -> {
        data.getNode().addEventHandler(MouseEvent.MOUSE_PRESSED,
(MouseEvent e) -> {
                    label.setText(String.valueOf(data.getPieValue())     +
"%");
                    popup.setX(e.getScreenX());
                    popup.setY(e.getScreenY());
                    popup.show(primaryStage);
                });
            });
            chart.addEventHandler(MouseEvent.DRAG_DETECTED,
(MouseEvent e) -> {
                chart.setStartAngle(chart.getStartAngle()+2);
            });
```

5. Додайте кругову діаграму в кореневий вузол графа сцени:

```
root.getChildren().add(chart);
```

6. Запустіть створений JavaFX-додаток, клацнувши правою кнопкою миші на піктограмі *Очистити та побудувати проект* (Shift+F11), а потім – *Виконати проект* (F6).

У результаті можна буде побачити кругову діаграму (рис. 1), у якій відображено розподіл валютних активів Ощадбанку по валюті у 2022 році. При натисканні лівою кнопкою миші на секторі з’являється числове значення у валюті.

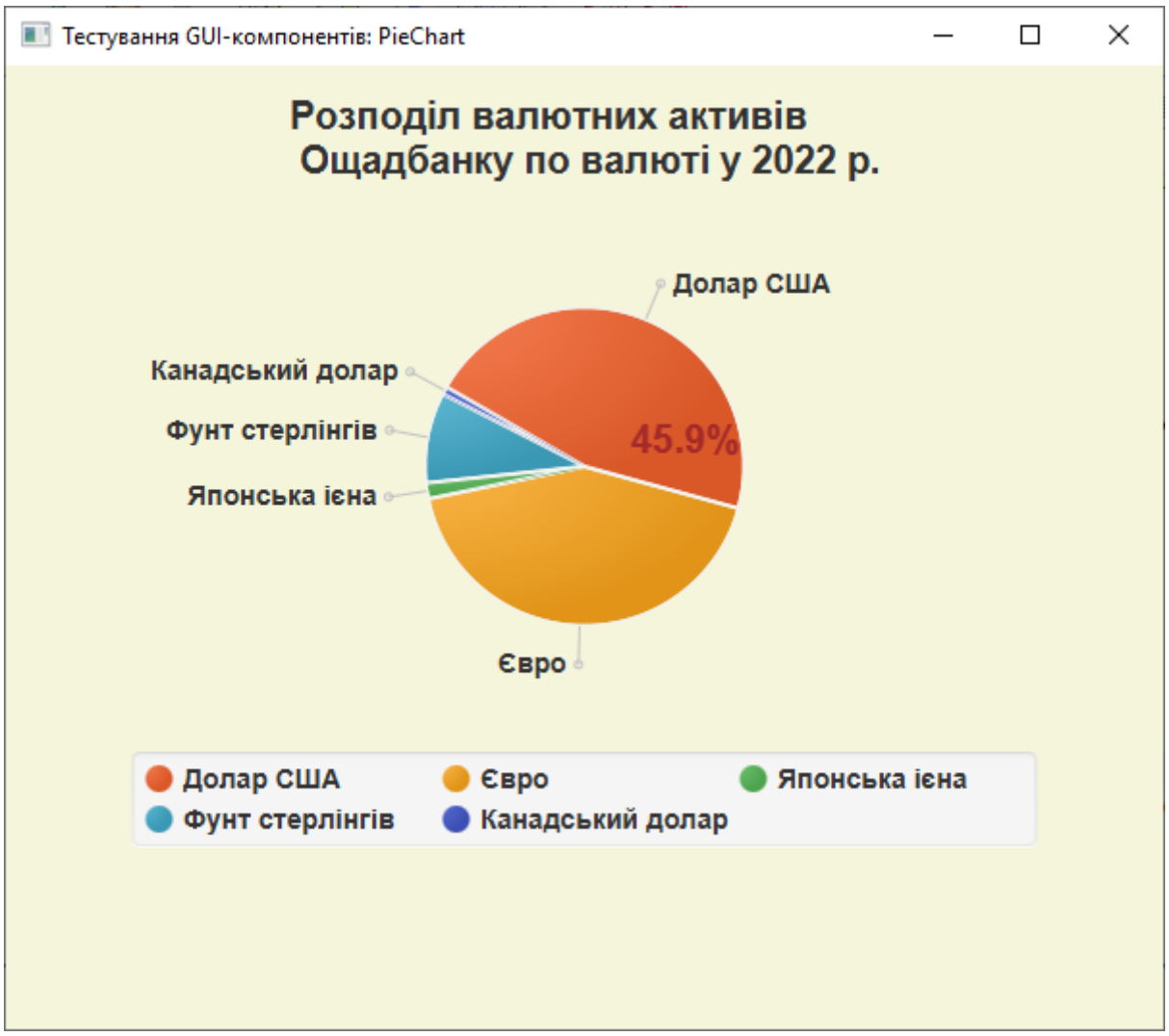


Рис. 1. JavaFX-додаток з GUI-інтерфейсом, що містить кругову діаграму

Контрольні запитання

1. Яке призначення компонента *PieChart*?
2. Як створити екземпляр компонента *PieChart*?
3. Які основні властивості має компонент *PieChart?*

Лабораторна робота 20. Візуальні ефекти Blend, Bloom, Glow

Мета: набуття вмінь та навичок створення JavaFX-додатків, які містять візуальні ефекти: змішування Blend, свічення Bloom та Glow.

Теоретичні відомості

Візуальні ефекти платформи JavaFX 2.0 представлені javafx.scene.effect. При цьому базовим класом всіх ефектів пакета javafx.scene.effect є абстрактний клас Effect, що має реалізації у вигляді класів Blend, Bloom, BoxBlur, ColorAdjust, ColorInput, DisplacementMap, DropShadow, GaussianBlur, Glow, ImageInput,

InnerShadow, Lighting, MotionBlur, PerspectiveTransform, Reflection, SepiaTone і Shadow.

Ефект, представлений Effect-об'єктом, пов'язується з вузлом графа сцени за допомогою методу setEffect() класу javafx.scene.Node та забезпечує створення нового зображення Node-вузла, отриманого в результаті модифікації вихідного графічного Node-вузла.

Ефект змішування Blend

Візуальний ефект змішування представлений у технології JavaFX 2.0 класом javafx.scene.effect.Blend.

Примірник класу Blend може бути створений за допомогою класу-фабрики javafx.scene.effect.BlendBuilder або за допомогою конструктора:

Blend blend = new Blend ();

Blend-ефект бере як один вхід зображення вузла Node, до якого ефект приєднаний за допомогою методу setEffect() класу Node, і змішує його з іншим ефектом Effect, який виступає як інший вхід Blend-ефекту. У цьому режим змішування визначається властивістю mode класу Blend.

Другий вхід Blend-ефекту, що містить об'єкт Effect, може бути двох типів – це може бути нижній або верхній вхід операції змішування. Effect-об'єкт встановлюється як нижній вхід за допомогою методу setBottomInput() класу Blend або як верхній вхід за допомогою методу setTopInput() класу Blend.

Режим змішування зображення вузла Node з Effect-об'єктом встановлюється методом setMode() класу Blend, що приймає як аргумент поле SRC_OVER, SRC_IN, SRC_OUT, SRC_ATOP, ADD, MULTIPLY, SCREEN, OVERLAY, DARKEN, LIGHTEN, COLOR_DODGE, COLOR_BURN, HARD_LIGHT, SOFT_LIGHT, DIFFERENCE, EXCLUSION, RED, GREEN і BLUE перерахування javafx.scene.effect.BlendMode.

Крім властивостей mode, bottomInput та topInput клас Blend має властивість opacity, що визначає прозорість верхнього введення перед змішуванням, значення якого встановлюється за допомогою методу setOpacity() класу Blend.

Ефект світіння Bloom

Візуальний ефект світіння Bloom представлений у технології JavaFX 2.0 класом javafx.scene.effect.Bloom.

Примірник класу Bloom може бути створений за допомогою фабрики javafx.scene.effect.BloomBuilder або за допомогою

конструктора:

Bloom bloom = new bloom();

Bloom-ефект бере як вхід зображення вузла Node, до якого ефект приєднаний за допомогою методу setEffect() класу Node, і засвічує яскраві ділянки графічного вмісту вузла Node, ґрунтуючись на значенні властивості threshold.

Властивість threshold класу Bloom визначає поріг яскравості пікселів, після якого вони будуть світитися, і може набувати значення від 0.0 до 1.0 (за замовчуванням 0.3).

Інша властивість input класу Bloom, значення якого встановлюється за допомогою методу setInput(), може визначати як вхід інший ефект Effect, створюючи, таким чином, ланцюжок ефектів.

Ефект світіння Glow

Візуальний ефект світіння Glow представлений у технології JavaFX 2.0 класом javafx.scene.effect.Glow.

Примірник класу Glow може бути створений за допомогою фабрики javafx.scene.effect.GlowBuilder або за допомогою конструктора:

Glow glow = new Glow();

Glow-ефект бере як вхід зображення вузла Node, до якого ефект приєднаний за допомогою методу setEffect() класу Node, і засвічує графічний вміст вузла Node, ґрунтуючись на значенні властивості level.

Властивість рівня класу Glow визначає інтенсивність світіння і може набувати значення від 0.0 до 1.0 (за замовчуванням 0.3).

Інша властивість input класу Glow, значення якого встановлюється за допомогою методу setInput(), може визначати як вхід інший ефект Effect, створюючи, таким чином, ланцюжок ефектів.

Дія Glow-ефекту подібна до дії Bloom-ефекту, відрізняючись тим, що Glow-ефект засвічує зображення більш рівномірно.

Завдання

1. Відкрийте середовище *NetBeans* з підтримкою платформи JavaFX 2.0. Для створення JavaFX-додатку з GUI-інтерфейсом, що містить візуальний ефект *Blend*, в меню *Файл* оберіть *Створити проект | Java with Ant | JavaFX | JavaFX Application*, натисніть кнопку *Далі*, введіть ім’я проекту *JavaFXApplicationBlendBloomGlow*, залишивши прапорець *Create Main Class*, та натисніть кнопку *Готово*.

У вікні редактора середовища NetBeans з’явиться код згенерованого класу *JavaFXApplicationBlendBloomGlow*, який розширює клас Application.

```
package javafxapplicationbutton;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class JavaFXApplicationBlendBloomGlow extends
Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say ‘Hello World’");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

2. У методі *start()* вилучіть весь код.

3. Створіть кореневий вузол графа сцени та на його основі екземпляр сцени, який встановлюється для об’єкта Stage. Установіть заголовок основного вікна JavaFX-додатку “Тестування візуальних ефектів: Blend, Bloom, Glow” та зробіть видимим об’єкт Stage:

```
FlowPane root = new FlowPane();
root.setOrientation(Orientation.VERTICAL);
root.setAlignment(Pos.CENTER);
root.setVgap(8);
Scene scene = new Scene(root, 500, 300, Color.GRAY);
primaryStage.setScene(scene);
primaryStage.setTitle("Тестування візуальних ефектів: Blend, Bloom, Glow ");
primaryStage.show();
```

4. Створіть кнопку Button. При проходженні курсора миші через неї графічний вміст кнопки Button має поєднуватись із лінійним градієнтом за допомогою Blend-ефекту. В якості нижнього входу Blend-ефекту використайте ефект ColorInput, що забезпечує як вхід для іншого ефекту прямокутник, заповнений певним кольором:

```
final Button btn = new Button();
btn.setLayoutX(100);
btn.setLayoutY(100);
btn.setText("Blend");
btn.setPrefSize(100,50);
btn.setStyle("-fx-font: italic bold 14pt Georgia; "
                + "-fx-text-fill: white; -fxbackground-color: black;");
Stop[] stops = new Stop[] {
    new Stop(0, Color.RED), new Stop(1, Color.YELLOW)
};
LinearGradient lg = new LinearGradient(0, 0, 0.25, 0.25, true, CycleMethod.REFLECT, stops);
ColorInput colorInput = new ColorInput();
colorInput.setWidth(100);
colorInput.setHeight(50);
colorInput.setPaint(lg);
final Blend blend = new Blend();
blend.setMode(BlendMode.LIGHTEN);
blend.setBottomInput(colorInput);
btn.setOnMouseEntered((MouseEvent event) -> {
    btn.setEffect(blend);
});
btn.setOnMouseExited((MouseEvent event) -> {
    btn.setEffect(null);
});
```

5. Створіть кнопку Button, через яку при проходженні курсора миші яскраві ділянки графічного вмісту кнопки Button засвічуються, створюючи ілюзію освітлення яскравим джерелом світла:

```
final Button btn1 = new Button();
btn1.setLayoutX(100);
btn1.setLayoutY(100);
btn1.setText("Bloom");
btn1.setPrefSize(100,30);
btn1.setStyle("-fx-font: bold italic 14pt Georgia; "
    + "-fx-text-fill: white; -fx-background-color: #a0522d;"
    + "-fx-border-width: 3px; "
    + "-fx-border-color: #f4a460#800000 #800000 #f4a460;");
final Bloom bloom = new Bloom();
bloom.setThreshold(0.3);
btn1.setOnMouseEntered((MouseEvent event) -> {
    btn1.setEffect(bloom);
});
btn1.setOnMouseExited((MouseEvent event) -> {
    btn1.setEffect(null);
});
```

6. Створіть кнопку Button, що має ореол світіння, що створюється за допомогою світіння прямокутника, розташованого під кнопкою. При цьому інтенсивність світіння має змінюватися з часом із використанням анімації:

```
Button btn2 = new Button();
btn2.setText("Glow");
btn2.setPrefSize(100,30);
btn2.setStyle("-fx-font: bold italic 14pt Georgia;"
    + "-fx-text-fill: white; "
    + "-fx-background-color: #a0522d;"
    + "-fx-border-width: 3px; "
    + "-fx-border-color:#f4a460 #800000 #800000 #f4a460;");
Glow glow = new Glow();
glow.setLevel(0.0);
Timeline timeline = new Timeline();
timeline.setCycleCount(Timeline.INDEFINITE);
timeline.setAutoReverse(true);
KeyValue kv = new KeyValue(glow.levelProperty(), 1.0);
KeyFrame kf = new KeyFrame(Duration.millis(2000), kv);
```

```
timeline.getKeyFrames().add(kf);
timeline.play();
Rectangle r = new Rectangle();
r.setWidth(115);
r.setHeight(45);
r.setArcHeight(20);
r.setArcWidth(20);
r.setFill(Color.web("#f4a460"));
r.setStroke(Color.WHITE);
r.setStrokeWidth(5);
r.setStrokeType(StrokeType.OUTSIDE);
r.setEffect(glow);
```

4. Створіть екземпляр компонування StackPane із відповідними властивостями, додайте до нього об’єкти r та btn2.

```
StackPane pane = new StackPane();
pane.getChildren().addAll(r,btn2);
pane.setLayoutX(100);
pane.setLayoutY(100);
```

5. Створені об’єкти btn, btn1 та pane додайте в кореневий вузол графа сцени:

```
root.getChildren().addAll(btn, btn1, pane);
```

6. Запустіть створений JavaFX-додаток, клацнувши правою кнопкою миші на піктограмі *Очистити та побудувати проект* (Shift+F11), а потім – *Виконати проект* (F6).

У результаті можна буде побачити кнопки із візуальними ефектами Blend, Bloom і Glow (рис. 1).

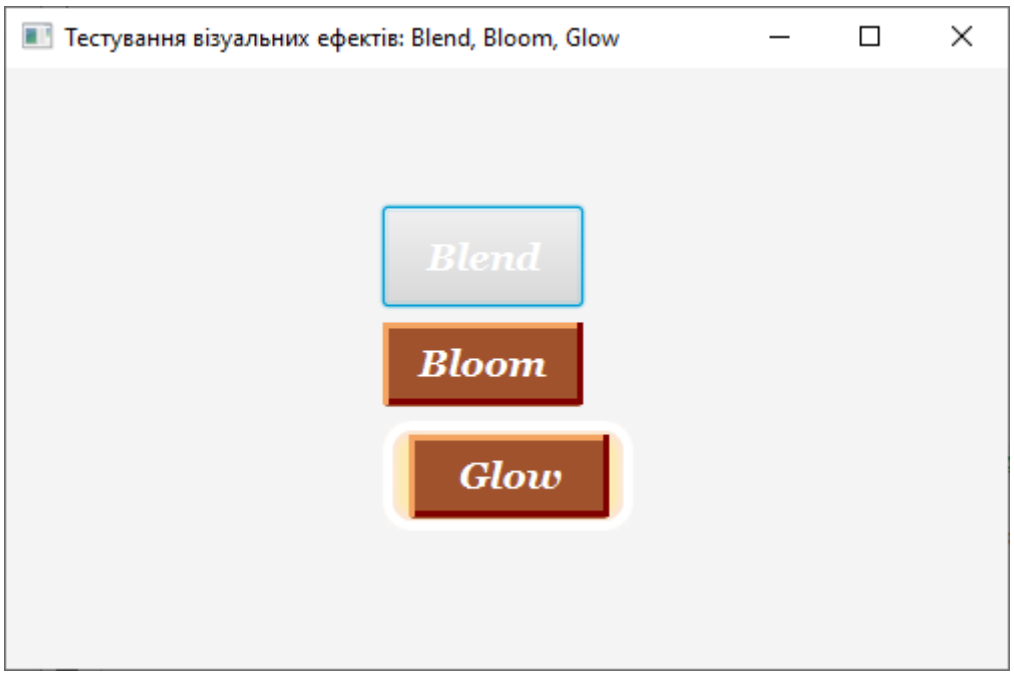


Рис. 1. JavaFX-додаток, що містить кнопки з візуальними ефектами Blend, Bloom і Glow

Контрольні запитання

1. Яке призначення візуальних ефектів *Blend*, *Bloom* і *Glow*?
2. Як створити екземпляри ефектів *Blend*, *Bloom* і *Glow*?
3. Які основні властивості мають візуальні ефекти *Blend*, *Bloom* і *Glow*?

Лабораторна робота 21. Трансформація і анімація

Мета: набуття вмінь та навичок створення JavaFX-додатків, які містять трансформацію та анімацію.

Теоретичні відомості

Платформа *JavaFX* 2.0 забезпечує створення двох видів анімації – анімацію по ключовим кадрам і анімацію з вбудованою тимчасової шкалою.

JavaFX-анімацію представляє пакет javafx.animation, базовим класом якого є клас *Animation*. Клас *Animation* розширюється класами *Timeline* і *Transition*, при цьому клас *Timeline* представляє анімацію по ключовим кадрам, а клас *Transition* – анімацію з вбудованою тимчасовою шкалою.

Клас *Animation* має набір властивостей, що дозволяють управляти швидкістю і напрямком анімації, затримкою і кількістю циклів анімації, встановлювати автореверс анімації, зчитувати статус анімації, обробляти завершення анімації та ін.

Швидкість і напрямок анімації можна встановити за допомогою методу *setRate(double value)*, затримку анімації – за допомогою методу *setDelay(Duration value)*, кількість циклів анімації – методом *setCycleCount(int value),* автореверс анімації – методом *setAutoReverse(boolean value)*, зчитати статус *Animation.Status.PAUSED, Animation.Status.RUNNING* або *Animation.Status.STOPPED* анімації – методом *getStatus()*, встановити обробник завершення анімації – методом *setOnFinished(EventHandler <ActionEvent> value)*.

Також клас *Animation* надає методи управління життєвим циклом анімації:

jumpTo(Duration time) – перехід анімації до зазначеної позиції на часовій шкалі;

playFrom(Duration time) – запуск анімації, починаючи з вказаної позиції на часовій шкалі;

play() – запуск анімації з поточної позиції на часовій шкалі;

playFromStart() – запуск анімації з початкової позиції на часовій шкалі;

stop() – зупинка анімації;

pause() – пауза анімації.

Анімація по ключовим кадрам дозволяє створити видиму зміну значення будь-якої *JavaFX*-властивості за певний проміжок часу за допомогою класу *Timeline*.

Примірник класу *Timeline* можна створити за допомогою класу-фабрики *TimelineBuilder* або за допомогою одного з конструкторів, що дозволяють встановити частоту кадрів і набір ключових кадрів анімації:

public Timeline(double targetFramerate)

public Timeline(double targetFramerate, KeyFrame ... keyFrames)

public Timeline(KeyFrame ... keyFrames) і public Timeline()

Набір ключових кадрів *Timeline*-анімації можна поповнити методом *getKeyFrames().AddAll()*, а зупинити *Timeline*-анімацію і повернути її до початкової позиції – методом *stop()*.

Ключовий кадр *Timeline*-анімації представлений класом *javafx.animation.KeyFrame* і визначає зміни значень *JavaFX*-властивостей за певний проміжок часу.

Примірник класу *KeyFrame* можна створити за допомогою набору конструкторів, що дозволяють встановити час відтворення ключового кадру, ім’я ключового кадру, обробник закінчення ключового кадру і

набір змін значень *JavaFX*-властивостей:

public KeyFrame(Duration time, java.lang.String name, EventHandler <ActionEvent> onFinished, java.util.Collection <KeyValue <? >> values)

public KeyFrame(Duration time, java.lang.String name, EventHandler <ActionEvent> onFinished, KeyValue <?> ... values)

public KeyFrame(Duration time, EventHandler <ActionEvent> onFinished, KeyValue <?> ... values)

public KeyFrame(Duration time, java.lang.String name, KeyValue <?> ... values)

public KeyFrame(Duration time, KeyValue <?> ... values)

Зміну значення *JavaFX*-властивості представлено класом *javafx.animation.KeyValue*, екземпляр якого можна створити за допомогою конструкторів, що дозволяють встановити змінювану *JavaFX*-властивість, її кінцеве значення в результаті анімації і спосіб її зміни протягом анімації:

public KeyValue(WritableValue <T> target, T endValue, Interpolator <? super T> interpolator)

public KeyValue(WritableValue <T> target, T endValue)

Спосіб зміни значення *JavaFX*-властивості протягом анімації представлений класом *javafx.animation.Interpolator*, який має статичні поля:

Interpolator.DISCRETE – дискретна зміна значення *JavaFX*-властивості, при якому значення залишається початковим до закінчення тимчасового інтервалу, коли значення стає кінцевим;

Interpolator.LINEAR(за замовчуванням) – лінійна зміна значення *JavaFX*-властивості, при якому значення визначається за формулою

startValue +(endValue – startValue) fraction;

Interpolator.EASE_BOTH – використовується величина 0.2 для приросту і зменшення значення *JavaFX*-властивості;

Interpolator.EASE_IN – використовується величина 0.2 для приросту значення *JavaFX*-властивості;

Interpolator.EASE_OUT – використовується величина 0.2 для зменшення значення *JavaFX*-властивості.

Крім того, можна створити окремий клас *Interpolator* з перевизначенням його методів, що описують зміну значення *JavaFX*-властивості.

Transition-анімація з вбудованою тимчасової шкалою також використовує об’єкт *Interpolator* як значення властивості *interpolator*

класу *Transition*.

Таким чином, анімацію зміни значень декількох *JavaFX*-властивостей можна створити двома способами. Перший спосіб – це створення одного ключового кадру *KeyFrame* і додавання в нього декількох об’єктів *KeyValue*. Інший спосіб – це створення окремих ключових кадрів *KeyFrame* для кожного з об’єктів *KeyValue* і додавання їх в *Timeline*-анімацію.

Transition-анімація з вбудованою тимчасової шкалою, на відміну від *Timeline*-анімації, описує зміну в часі обмеженого набору *JavaFX*-властивостей, таких як прозорість, просторове положення, обертання і масштабування вузла графа сцени, а також колір заповнення і колір контуру форми *Shape*.

Анімація прозорості вузла графа сцени створюється за допомогою класу *FadeTransition*, що має властивості:

byValue – крок зміни властивості прозорості;

duration – тривалість анімації;

fromValue – початкове значення прозорості;

node – цільовий вузол графа сцени даної анімації;

toValue – кінцеве значення прозорості.

Примірник класу *FadeTransition* створюється за допомогою класу-фабрики *FadeTransitionBuilder* або за допомогою конструкторів

public FadeTransition(Duration duration, Node node)

public FadeTransition(Duration duration)

public FadeTransition()

Анімація просторового положення вузла графа сцени створюється за допомогою класів *PathTransition* і *TranslateTransition*.

Клас *PathTransition* дозволяє створювати переміщення графічного об’єкта уздовж кривої за допомогою властивостей:

duration – тривалість анімації;

orientation – орієнтація, якщо *PathTransition.OrientationType.NONE* – орієнтація графічного об’єкта не змінюється, якщо *PathTransition.OrientationType.ORTHOGONAL_TO_TANGENT* – графічний об’єкт перпендикулярний відносно кривої свого переміщення;

path – об’єкт *javafx.scene.shape.Shape*, що представляє криву переміщення.

Примірник класу *PathTransition* створюється за допомогою класу-фабрики *PathTransitionBuilder* або за допомогою конструкторів:

public PathTransition()

public PathTransition(Duration duration, Shape path, Node node)

public PathTransition(Duration duration, Shape path)

Клас *TranslateTransition* дозволяє створювати переміщення графічного об’єкта з однієї 3D-точки в іншу 3D-точку за допомогою властивостей:

node – цільової вузол для анімації;

duration – тривалість анімації;

fromX – початкова координата переміщення по осі *x*;

fromY – початкова координата переміщення по осі *y*;

fromZ – початкова координата переміщення по осі *z*;

toX – кінцева координата переміщення по осі *x*;

toY – кінцева координата переміщення по осі *y*;

toZ – кінцева координата переміщення по осі *z*;

byX – крок переміщення по осі *x*;

byY – крок переміщення по осі *y*;

byZ – крок переміщення по осі *z*.

Примірник класу *TranslateTransition* можна створити за допомогою класу-фабрики *TranslateTransitionBuilder* або за допомогою конструкторів:

public TranslateTransition(Duration duration, Node node)

public TranslateTransition(Duration duration)

public TranslateTransition()

Анімація обертання вузла графа сцени створюється за допомогою класу *RotateTransition*, які мають властивості:

node – цільової вузол анімації;

duration – тривалість анімації;

axis – вісь обертання *javafx.geometry.Point3D*;

fromAngle – початковий кут обертання;

toAngle – кінцевий кут обертання;

byAngle – крок обертання.

Примірник класу *RotateTransition* створюється за допомогою класу-фабрики *RotateTransitionBuilder* або за допомогою конструкторів:

public RotateTransition(Duration duration, Node node)

public RotateTransition(Duration duration)

public RotateTransition()

Анімація масштабування вузла графа сцени створюється за допомогою класу *ScaleTransition*, що має властивості:

node – цільової вузол анімації;

duration – тривалість анімації;

fromX – початкове значення масштабування по осі *x*;

fromY – початкове значення масштабування по осі *y*;

fromZ – початкове значення масштабування по осі *z*;

toX – кінцеве значення масштабування по осі *x*;

toY – кінцеве значення масштабування по осі *y*;

toZ – кінцеве значення масштабування по осі *z*;

byX – крок масштабування по осі *x*;

byY – крок масштабування по осі *y*;

byZ – крок масштабування по осі *z*.

Примірник класу *ScaleTransition* можна створити за допомогою класу-фабрики

ScaleTransitionBuilder або за допомогою конструкторів:

public ScaleTransition(Duration duration, Node node)

public ScaleTransition(Duration duration)

public ScaleTransition()

Анімація кольору заповнення форми *Shape* створюється за допомогою класу *FillTransition*, що має властивості:

duration – тривалість анімації;

fromValue – початкове значення кольору;

shape – цільовий об’єкт *javafx.scene.shape.Shape* анімації;

toValue – кінцеве значення кольору.

Примірник класу *FillTransition* створюється за допомогою класу-фабрики *FillTransitionBuilder* або за допомогою конструкторів:

public FillTransition(Duration duration, Shape shape, Color fromValue, Color toValue)

public FillTransition(Duration duration, Color fromValue, Color toValue)

public FillTransition(Duration duration, Shape shape)

public FillTransition(Duration duration)

public FillTransition()

Анімація кольору контуру форми *Shape* створюється за допомогою класу *StrokeTransition*, що має властивості:

shape – цільовий об’єкт *javafx.scene.shape.Shape* для анімації;

duration – тривалість анімації;

fromValue – початковий колір контуру;

toValue – кінцевий колір контуру.

Примірник класу *StrokeTransition* створюється за допомогою

класу-фабрики *StrokeTransitionBuilder* або за допомогою конструкторів:

public StrokeTransition(Duration duration, Shape shape, Color fromValue, Color toValue)

public StrokeTransition(Duration duration, Color fromValue, Color toValue)

public StrokeTransition(Duration duration, Shape shape)

public StrokeTransition(Duration duration)

public StrokeTransition()

Класи *ParallelTransition* і *SequentialTransition* дають можливість групувати анімації в паралельне і послідовне виконання.

Клас *ParallelTransition* має властивість *node*(цільової вузол графа сцени для анімації) і конструктори:

public ParallelTransition()

public ParallelTransition(Node node, Animation ... children)

public ParallelTransition(Animation ... children)

public ParallelTransition(Node node)

також клас-фабрику *ParallelTransitionBuilder*. Метод *getChildren().AddAll* дозволяє поповнити список дочірніх паралельних анімацій.

Клас *SequentialTransition* має властивість *node*(цільової вузол графа сцени для анімації) і конструктори:

public SequentialTransition()

public SequentialTransition(Node node, Animation ... children)

public SequentialTransition(Animation ... children)

public SequentialTransition(Node node)

також клас-фабрику *SequentialTransitionBuilder*. Метод *getChildren().AddAll* дозволяє поповнити список дочірніх послідовних анімацій.

Клас *PauseTransition* дозволяє зробити паузу в послідовності анімацій. Клас *PauseTransition* має властивість *duration*(тривалість паузи) і конструктори *public PauseTransition*(*Duration duration*) і *public PauseTransition*(), а також клас-фабрику *PauseTransitionBuilder*.

Клас *AnimationTimer* дозволяє створювати таймер, що викликається в кожному кадрі анімації. Створити таймер можна розширивши абстрактний клас *AnimationTimer* з перевизначенням його методу *handle*(*long now*), що викликається в кожному кадрі. Для управління таймером клас *AnimationTimer* пропонує методи *start*() і *stop*().

Пакет *javafx.scene.transform* платформи *JavaFX* 2.0 забезпечує трансформації вузлів графа сцени, що складаються з афінних перетворень: обертань, переміщень, масштабування і зсуву.

На відміну від анімації, трансформації графічних об'єктів не мають плавного видимого переходу від початкової точки до кінцевої точки протягом певного проміжку часу, а виконуються відразу.

Базовим класом *JavaFX*-трансформацій є клас *javafx.scene.transform.Transform*, що має реалізації у вигляді класів *Affine*, *Rotate*, *Scale*, *Shear* і *Translate*.

Застосувати *JavaFX*-трансформації до вузла графа сцени можна двома способами. Перший спосіб – це використовувати метод *getTransforms*() класу *javafx.scene.Node*, який повертає список *ObservableList<Transform>* об'єктів *Transform*, поповнити який можна методом *addAll()*. Інший спосіб – це застосування методів *setRotate()*, *setRotationAxis()*, *setScaleX()*, *setScaleY()*, *setScaleZ()*, *setTranslateX()*, *setTranslateY()* і *setTranslateZ()* класу *javafx.scene.Node*, що забезпечують трансформації обертання, масштабування і переміщення для вузла графа сцени.

Клас *Affine* представляє афінне перетворення матриці:

mxx mxy mxz tx

myx myy myz ty

mzx mzy mzz tz

за допомогою властивостей:

mxx – *X*-множник матриці;

mxy – *XY*-множник матриці;

mxz – *XZ*-множник матриці;

tx – зсув по осі *x*;

myx – *YX*-множник матриці;

myy – *Y*-множник матриці;

myz – *YZ*-множник матриці;

ty – зсув по осі *y*;

mzx – *ZX*-множник матриці;

mzy – *ZY*-множник матриці;

mzz – *Z*-множник матриці;

tz – зсув по осі *z*.

Примірник класу *Affine* можна створити за допомогою класу-фабрики *AffineBuilder*, за допомогою конструктора *public Affine()* або за допомогою статичного методу *affine()* класу *Transform*, що повертає Affine-об'єкт.

Афінні перетворення відображають *n*-мірний об’єкт в *n*-мірний, зберігають паралельність ліній і площин, а також пропорції паралельних об’єктів.

За допомогою афінних перетворень можна створювати трансформації обертання, переміщення, масштабування і зсуву.

Клас *Rotate* забезпечує обертання вузла графа сцени за допомогою властивостей:

angle – кут обертання;

pivotX – горизонтальна координата опорної точки обертання;

pivotY – вертикальна координата опорної точки обертання;

pivotZ – *Z*-координата опорної точки обертання.

Примірник класу *Rotate* можна створити за допомогою класу-фабрики *RotateBuilder*, за допомогою набору конструкторів:

public Rotate(), public Rotate(double angle)

public Rotate(double angle, Point3D axis)

public Rotate(double angle, double pivotX, double pivotY)

public Rotate(double angle, double pivotX, double pivotY, double pivotZ)

public Rotate(double angle, double pivotX, double pivotY,

double pivotZ, Point3D axis)

або за допомогою статичного методу *rotate()* класу *Transform*, який повертає *Rotate*-об’єкт.

Клас *Scale* забезпечує масштабування вузла графа сцени за допомогою властивостей:

x – множник масштабування по осі *x*;

y – множник масштабування по осі *y*;

z – множник масштабування по осі *z*;

pivotX – горизонтальна координата опорної точки масштабування;

pivotY – вертикальна координата опорної точки масштабування;

pivotZ – *Z*-координата опорної точки масштабування.

Примірник класу *Scale* можна створити за допомогою класу-фабрики *ScaleBuilder*, за допомогою набору конструкторів:

public Scale()

public Scale(double x, double y)

public Scale(double x, double y, double pivotX, double pivotY)

public Scale(double x, double y, double z)

public Scale(double x, double y, double z, double pivotX, double pivotY, double pivotZ)

або за допомогою статичного методу *scale()* класу *Scale*-об’єкт.

Клас *Shear* забезпечує зміщення вузла графа сцени за допомогою властивостей:

x – множник по осі *x* від -1 до 1;

y – множник по осі *y* від -1 до 1;

pivotX – горизонтальна координата опорної точки зсуву;

pivotY – вертикальна координата опорної точки зсуву.

Примірник класу *Shear* можна створити за допомогою класу-фабрики *ShearBuilder*, за допомогою набору конструкторів:

public Shear()

public Shear(double x, double y)

public Shear(double x, double y, double pivotX, double pivotY)

або за допомогою статичного методу *shear()* класу *Transform*, що повертає Shear-об'єкт.

Клас *Translate* забезпечує переміщення вузла графа сцени за допомогою властивостей:

x – зміщення по осі *x*;

y – зміщення по осі *y*;

z – зміщення по осі *z*.

Примірник класу *Translate* можна створити за допомогою класу-фабрики *TranslateBuilder*, за допомогою набору конструкторів

public Translate()

public Translate(double x, double y)

public Translate(double x, double y, double z)

або за допомогою статичного методу *translate()* класу *Transform*, що повертає *Translate*-об'єкт.

Завдання

1. Відкрийте середовище *NetBeans* з підтримкою платформи JavaFX 2.0. Для створення JavaFX-додатку з GUI-інтерфейсом, що містить трансформацію та анімацію, в меню *Файл* оберіть *Створити проект* | *Java with Ant* | *JavaFX* | *JavaFX Application*, натисніть кнопку *Далі*, введіть ім'я проекту *JavaFXApplicationTransition*, залишивши прапорець *Create Main Class*, та натисніть кнопку *Готово*.

У вікні редактора середовища NetBeans з'явиться код згенерованого класу *JavaFXApplicationTransition*, який розширює клас Application.

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

```
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;
public class JavaFXApplicationTransition extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say ‘Hello World’");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

2. У методі *start()* вилучіть весь код.

3. Створіть кореневий вузол графа сцени та на його основі екземпляр сцени, який встановлюється для об’єкта Stage. Установіть заголовок основного вікна JavaFX-додатку “Тестування трансформації та анімації” та зробіть видимим об’єкт Stage:

```
Group root = new Group();
Scene scene = new Scene(root, 500, 500, Color.GRAY);
primaryStage.setScene(scene);
primaryStage.setTitle("Тестування трансформації та анімації");
primaryStage.show();
```

4. Продемонструйте *Transition*- і *Timeline*-анімацію букв тексту. *Transition*-анімація має складатись із паралельних анімацій переміщення, обертання і масштабування, а *Timeline*-анімація має

змінювати розташування джерела світла для ефекту підсвічування тексту:

```
// Створення ефекту підсвічування тексту
final Light.Point lightPoint = new Light.Point();
lightPoint.setColor(Color.WHITE);
lightPoint.setX(0.0);
lightPoint.setY(0.0);
lightPoint.setZ(100.0);
final Lighting effect = new Lighting();
effect.setLight(lightPoint);
effect.setDiffuseConstant(1.5);
effect.setSpecularConstant(1.5);
effect.setSurfaceScale(8);
// Створення букв тексту
final Text tJ = new Text();
tJ.setEffect(effect);
tJ.setX(80);
tJ.setY(250);
tJ.setText( "J");
tJ.setFill(Color.RED);
tJ.setFont(Font.font(null, FontWeight.BOLD, 80));
final Text ta1 = new Text();
ta1.setEffect(effect);
ta1.setX(120);
ta1.setY(250);
ta1.setText( "a");
ta1.setFill(Color.RED);
ta1.setFont(Font.font(null, FontWeight.BOLD, 80));
final Text tv = new Text();
tv.setEffect(effect);
tv.setX(170);
tv.setY(250);
tv.setText( "v");
tv.setFill(Color.RED);
tv.setFont(Font.font(null, FontWeight.BOLD, 80));
final Text ta2 = new Text();
ta2.setEffect(effect);
ta2.setX(220);
ta2.setY(250);
```

```
        ta2.setText( "a");
        ta2.setFill(Color.RED);
        ta2.setFont(Font.font(null, FontWeight.BOLD, 80));
        final Text tF = new Text();
        tF.setEffect(effect);
        tF.setX(270);
        tF.setY(250);
        tF.setText( "F");
        tF.setFill(Color.RED);
        tF.setFont(Font.font(null, FontWeight.BOLD, 80));
        final Text tX = new Text();
        tX.setEffect(effect);
        tX.setX(320);
        tX.setY(250);
        tX.setText( "X");
        tX.setFill(Color.RED);
        tX.setFont(Font.font(null, FontWeight.BOLD, 80));
        // Створення анімацій переміщення, обертання і масштабування
букв тексту
        final TranslateTransition ttJ = new
TranslateTransition(Duration.millis(1000), tJ);
        ttJ.setCycleCount(2);
        ttJ.setByX(-200.0);
        ttJ.setToY(-270.0);
        ttJ.setAutoReverse(true);
        final RotateTransition rtJ = new
RotateTransition(Duration.millis(500), tJ);
        rtJ.setByAngle(360);
        rtJ.setAxis(new Point3D(10.0, 10.0, 10.0));
        rtJ.setCycleCount(4);
        final ScaleTransition stJ = new
ScaleTransition(Duration.millis(1000), tJ);
        stJ.setByX(-1.5);
        stJ.setByY(-1.5);
        stJ.setCycleCount(2);
        stJ.setAutoReverse(true);
        final TranslateTransition tta1 = new
TranslateTransition(Duration.millis(1000), ta1);
        tta1.setCycleCount(2);
```

```
        tta1.setByX(-100.0);
        tta1.setToY(-270.0);
        tta1.setAutoReverse(true);
        final RotateTransition rta1 = new
RotateTransition(Duration.millis(500), ta1);
        rta1.setByAngle(360);
        rta1.setAxis(new Point3D(10.0, 10.0, 10.0));
        rta1.setCycleCount(4);
        final ScaleTransition sta1 = new
ScaleTransition(Duration.millis(1000), ta1);
        sta1.setByX(-1.5);
        sta1.setByY(-1.5);
        sta1.setCycleCount(2);
        sta1.setAutoReverse(true);
        final TranslateTransition ttv = new
TranslateTransition(Duration.millis(1000), tv);
        ttv.setCycleCount(2);
        ttv.setByX(0.0);
        ttv.setToY(-270.0);
        ttv.setAutoReverse(true);
        final RotateTransition rtv = new
RotateTransition(Duration.millis(500), tv); rtv.setByAngle(360);
        rtv.setAxis(new Point3D(10.0, 10.0, 10.0));
        rtv.setCycleCount(4);
        final ScaleTransition stv = new
ScaleTransition(Duration.millis(1000), tv); stv.setByX(-1.5);
        stv.setByY(-1.5);
        stv.setCycleCount(2);
        stv.setAutoReverse(true);
        final TranslateTransition tta2 = new
TranslateTransition(Duration.millis(1000), ta2);
        tta2.setCycleCount(2);
        tta2.setByX(100.0);
        tta2.setToY(-270.0);
        tta2.setAutoReverse(true);
        final RotateTransition rta2 = new
RotateTransition(Duration.millis(500), ta2); rta2.setByAngle(360);
        rta2.setAxis(new Point3D(10.0, 10.0, 10.0));
        rta2.setCycleCount(4);
```

```
        final ScaleTransition sta2 = new
ScaleTransition(Duration.millis(1000), ta2); sta2.setByX(-1.5);
        sta2.setByY(-1.5);
        sta2.setCycleCount(2);
        sta2.setAutoReverse(true);
        final TranslateTransition ttF = new
TranslateTransition(Duration.millis(1000), tF);
        ttF.setCycleCount(2);
        ttF.setByX(200.0);
        ttF.setToY(-270.0);
        ttF.setAutoReverse(true);
        final RotateTransition rtF = new
RotateTransition(Duration.millis(500), tF); rtF.setByAngle(360);
        rtF.setAxis(new Point3D(10.0, 10.0, 10.0));
        rtF.setCycleCount(4);
        final ScaleTransition stF = new
ScaleTransition(Duration.millis(1000), tF);
        stF.setByX(-1.5);
        stF.setByY(-1.5);
        stF.setCycleCount(2);
        stF.setAutoReverse(true);
        final TranslateTransition ttX = new
TranslateTransition(Duration.millis(1000), tX);
        ttX.setCycleCount(2);
        ttX.setByX(300.0);
        ttX.setToY(-270.0);
        ttX.setAutoReverse(true);
        final RotateTransition rtX = new
RotateTransition(Duration.millis(500), tX);
        rtX.setByAngle(360);
        rtX.setAxis(new Point3D(10.0, 10.0, 10.0));
        rtX.setCycleCount(4);
        final ScaleTransition stX = new
ScaleTransition(Duration.millis(1000), tX);
        stX.setByX(-1.5);
        stX.setByY(-1.5);
        stX.setCycleCount(2);
        stX.setAutoReverse(true);
        // Групування анімацій в паралельне виконання
```

```
final ParallelTransition ptJ = new ParallelTransition(tJ, ttJ, rtJ, stJ);
final ParallelTransition pta1 = new ParallelTransition(ta1, tta1, rta1, sta1);
final ParallelTransition ptv = new ParallelTransition(tv, ttv, rtv, stv);
final ParallelTransition pta2 = new ParallelTransition(ta2, tta2, rta2, sta2);
final ParallelTransition ptF = new ParallelTransition(tF, ttF, rtF, stF);
final ParallelTransition ptX = new ParallelTransition(tX, ttX, rtX, stX);
// Створення анімації переміщення джерела світла ефекту підсвічування тексту
final Timeline timeline = new Timeline();
timeline.setCycleCount(2);
timeline.setAutoReverse(true);
KeyValue kv = new KeyValue(lightPoint.xProperty(), 500.0, Interpolator.EASE_BOTH);
KeyFrame kf = new KeyFrame(Duration.millis(1000), kv);
timeline.getKeyFrames(). addAll(kf);
timeline.setDelay(Duration.millis(2000));
// Створення кнопки запуску анімації
Button btn = new Button();
btn.setLayoutX(10);
btn.setLayoutY(450);
btn.setText( "Анімація");
btn.setStyle("- fx-font: bold italic 14pt Georgia; -fx-text-fill: white; -fx-background-color: gray; -fx-border-width: 1px; -fx-border-color: white ");
btn.setOnAction((ActionEvent event) -> {
    if(! timeline.getStatus().equals(Animation.Status.RUNNING)) {
        ptJ.play();
        pta1.play();
        ptv.play();
        pta2.play();
        ptF.play();
        ptX.play();
        timeline.play(); }
});
```

4. Створені об’єкти додайте в кореневий вузол графа сцени:

```
root.getChildren().addAll(btn, tJ, ta1, tv, ta2, tF, tX);
```

5. Запустіть створений JavaFX-додаток, клацнувши правою кнопкою миші на піктограмі *Очистити та побудувати проект* (Shift+F11), а потім – *Виконати проект* (F6).

У результаті можна буде побачити *Transition*- і *Timeline*-анімацію букв тексту (рис. 1).

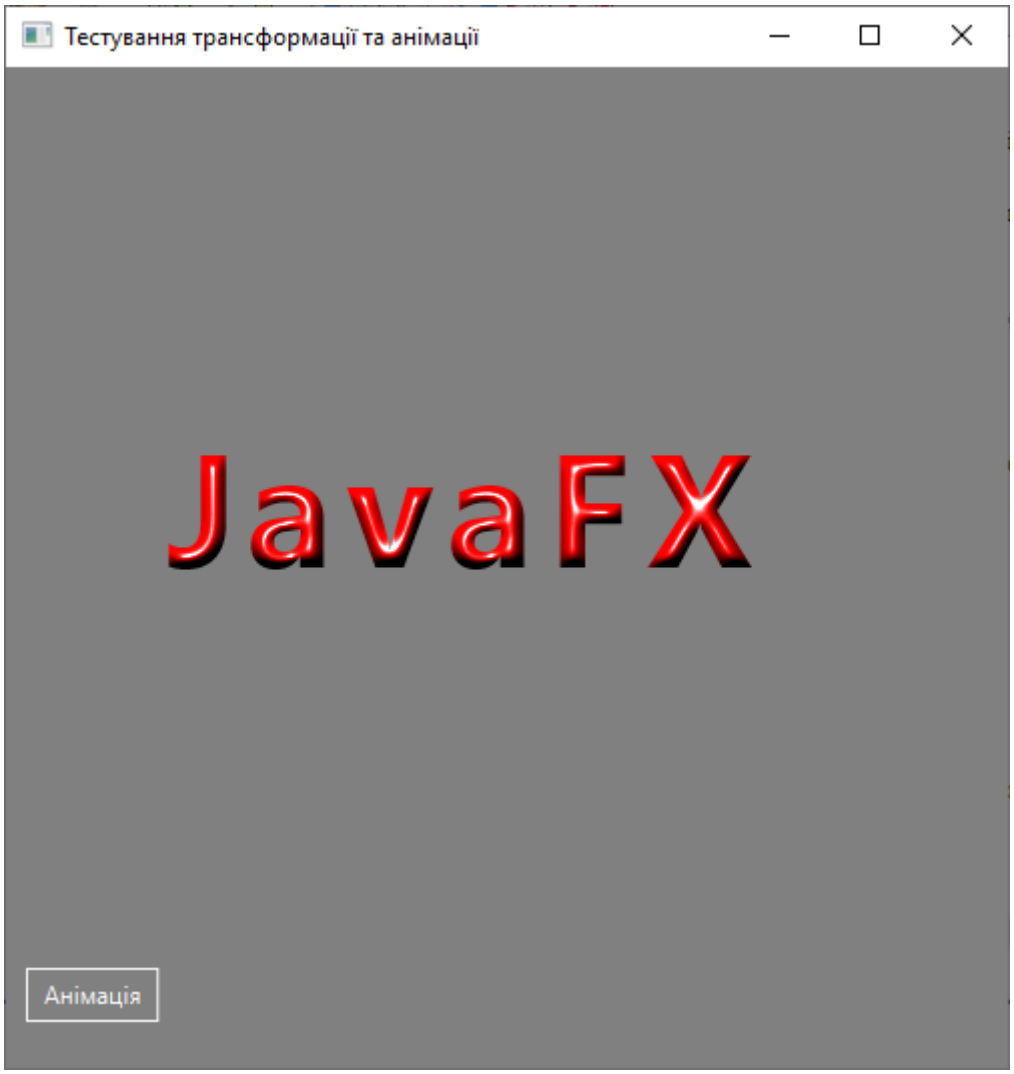


Рис. 1. JavaFX-додаток, що демонструє *Transition*- і *Timeline*-анімацію букв тексту

Контрольні запитання

1. Яке призначення *Transition*- і *Timeline*-анімації?
2. Як створити *Transition*- і *Timeline*-анімацію?
3. Які основні властивості мають *Transition*- і *Timeline*-анімації?

Список використаних джерел

1. Машнин Т. С. JavaFX 2.0: разработка RIA-приложений. СПб.: БХВ-Петербург, 2012. 320 с.
2. Муляр В. П. Основи розробки додатків з використанням технології JavaFX. Комп'ютерно-інтегровані технології: освіта, наука, виробництво. 2018. Вип. № 30-31. С. 104–110.
3. Муляр В. П. Розробка JavaFX-додатків із використанням Scene Builder. Комп'ютерно-інтегровані технології: освіта, наука, виробництво. 2020. Вип. № 39. С. 181–189.
4. Муляр В. П., Яцюк С. М. Елементи комп'ютерної графіки у візуалізації результатів моделювання фізичних явищ і процесів. Комп'ютерно-орієнтовані технології: освіта, наука, виробництво. 2016. № 23. С. 80–84.
5. Спірінцева О. В., Литвинов О. А., Герасимов В. В. Java-технології та мобільні пристрої. Алгоритми і структури даних: навч. посіб. Д.: Вид-во ДНУ ім. О. Гончара, 2016. 140 с.
6. Java Tutorial. URL: https://www.w3schools.com/java/default.asp
7. Java. Классы. Объектно-ориентированное программирование. URL: https://metanit.com/java/tutorial/3.1.php
8. Підручник з Java. URL: https://www.javatpoint.com/java-tutorial
9. GDB online Debugger / Compiler. URL: https://www.onlinegdb.com/
10. Java – Учебники по программированию. URL: https://betacode.net/
11. Apache NetBeans. URL: https://netbeans.apache.org/download/index.html
12. Java Course. URL: http://java-course.ru/begin/introduce/
13. Java SE Downloads. URL: https://www.oracle.com/java/technologies/javase-downloads.html
14. JavaFX. URL: https://gluonhq.com/products/javafx/
15. Scene Builder. URL: https://gluonhq.com/products/scene-builder/
16. Введение в Java FX. URL: https://metanit.com/java/javafx/1.1.php
17. Руководство JavaFX для начинающих – Hello JavaFX. URL: https://betacode.net/10623/javafx-tutorial-for-beginners

Для нотаток

Для нотаток

Навчальне видання

Муляр Вадим Петрович

ОБ’ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

Лабораторний практикум