МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний університет будівництва і архітектури

КРОС-ПЛАТФОРМНЕ ПРОГРАМУВАННЯ

Методичні вказівки
до виконання лабораторних робіт
для студентів 3-го курсу
спеціальності 122 «Комп’ютерні науки»

Київ 2020

УДК 681.3
К83

Укладачі: В.М. Хроленко, канд. техн. наук, доцент;
В.Г. Голенков, старш. викладач;
В.В. Гоц, канд. техн. наук, доцент

Рецензент М.І. Цюцюра, канд. техн. наук, доцент

Відповідальний за випуск С.В. Цюцюра, д-р техн. наук, професор

Затверджено на засіданні кафедри інформаційних технологій, протокол № 2 від 9 жовтня 2019 року.

В авторській редакції.

Крос-платформне програмування: методичні вказівки до виконання
К83 лабораторних робіт / уклад.: В.М. Хроленко, В.Г. Голенков, В.В. Гоц. – Київ: КНУБА, 2020. – 31 с.

Містять теоретичні відомості та програму до виконання лабораторних робіт з дисципліни «Крос-платформне програмування», які сприяють закріпленню та поглибленню студентами теоретичних знань з дисципліни та дають змогу студентам набути практичних навичок зі складання програм мовою Java.

Призначено для студентів 3-го курсу спеціальності 122 «Комп’ютерні науки».

ЗАГАЛЬНІ ПОЛОЖЕННЯ

Мета дисципліни: освоєння інструментальних програмних засобів, які мають широке застосування серед фахівців з розробки та впровадження інформаційних технологій.

Завдання дисципліни: набути навичок з програмування мовою Java.

Лабораторні роботи з дисципліни «Крос-платформне програмування» **сприяють**:

закріпленню і поглибленню студентами теоретичних знань з дисципліни;

дозволяють студентам набути практичних навичок зі складання програм мовою Java, тестування та налагодження програм.

Виконання лабораторних робіт дасть змогу студентам набути практичних навичок з програмування мовою Java та краще підготуватись до вивчення наступних спецдисциплін, передбачених навчальним планом, а також зробити вагомий крок до дипломного проектування.

ЛАБОРАТОРНА РОБОТА № 1
Побудова класу: опис даних класу, конструктори класу, методи класу

Мета роботи: дослідити механізм побудови класу.

1. Короткі теоретичні відомості з об’єктно-орієнтованого підходу побудови програм

Характерні риси об’єктно-орієнтованого підходу

Об’єктно-орієнтований підхід означає, що програмування здійснюється з використанням таких принципів: інкапсуляція даних, спадкування класів, поліморфізм об’єктів.

Інкапсуляція означає обмеження доступу до даних. Кожний об’єкт володіє певним набором даних. Доступ до цих даних інших об’єктів забезпечується за допомогою методів (функцій). Це забезпечує значне зменшення кількості помилок, пов’язаних із неправильним використанням даних. Використання даних іншого об’єкта дозволяється тільки через посередництво написаних програмістом методів.

Спадкування означає, що типи об’єктів можуть створюватись на основі інших уже створених типів об’єктів. Клас-нащадок повністю наслідує структуру даних батьківського класу та має інші дані. Таким чином, класи, які мають спільні риси структури, можна будувати на основі одного спільного типу об’єктів.

Цікаво, що змінним, які мають тип батьківського класу, можна присвоювати значення класів-нащадків (але не навпаки). Це особливо зручно, якщо потрібно організувати однакові дії для усіх об’єктів-нащадків, що

відносяться до одного батьківського класу. Наприклад, клас об’єктів «студент 5-ого курсу» може бути побудований на основі класу «студент» з додаванням методів «виконання дипломного проекту», «захист дипломного проекту», «отримання диплому». Клас «таксі» – на основі класу «авто» з додаванням методів «виконання замовлення», «розрахунок оплати замовлення».

Поліморфізм означає вибір методу залежно від об’єкта, на який фактично він посилається. Про поліморфізм об’єктів можна говорити тільки, якщо об’єкти знаходяться у відносинах спадкування. Наприклад, якщо в програмі визначений метод з однаковим ім’ям для батьківського об’єкта й об’єкта-нащадка, то під час виконання програми викликатись буде метод, на який фактично вказує в цей момент об’єкт = для кожного класу-нащадка свій метод, а якщо він не визначений, то метод класу-батька.

Означення класу

Отже, *клас* — це набір даних і методів, що призначені для їх обробки. Складовими частинами класу (у будь-якій мові ООП) є поля, в яких зберігається інформація про дані, методи, які призначені для обробки даних, і конструктори, що призначені для створення екземпляру класу, тобто *об’єкта*.

Отже, клас є структура даних разом з методами для їх обробки. Об’єкт є екземпляром класу з притаманними тільки йому даними.

Опис класу будь якою ОО мовою складається з: 1) назви класу; 2) опис полів (даних) класу; 3) опису конструктора класу; 4) опису методів класу.

Наприклад,

```
public class queue {   //  початок опису класу з ім’ям  queue
  //     поля класу
...
//     конструктори класу
  ...
//   методи класу
     ...
}  // кінець опису класу
```

Опис поля класу містить модифікатор доступу («private», «public», «protected»), визначення типу даних і назву поля. За принципом інкапсуляції даних поля класу об’являються з модифікатором доступу «private». За потреби для доступу до даних з інших класів розробляються відповідні методи.

Наприклад,

```
   private int state;
   private double salary;
   private String name
```

Опис конструктора класу містить модифікатор доступу («public»), назву, що збігається з назвою класу, і список параметрів у дужках. Список параметрів може бути порожнім, тоді в дужках нічого не розміщується, але дужки пишуться. Дужки означають для компілятора, що описується метод. Конструктор з порожнім списком параметрів називається «конструктором за замовчуванням» і відіграє важливу роль у реалізації механізму наслідування (про це в наступних лабораторних роботах). Основне призначення конструктора – ініціалізація полів даних об'єкта класу. Навіть, якщо не потрібно передавати спеціальні значення при створенні екземпляру класу, бажано прописувати значення, які присвоюються значенням полів, не покладаючись на їх ініціалізацію за замовчуванням.

Наприклад,

```
public Queue()  {     // конструктор «за замовчуванням»
       state = 0;      // ініціалізація поля значенням 0
       quantity = 0;
}

public Queue(int startState)  {   //
       state = startState;  //  передача значення аргументу у поле об'єкта
       quantity = 0; // ініціалізація поля значенням 0
}
```

Опис методу містить модифікатор доступу, визначення типу значення, що повертається як результат виконання методу (можливо, що в результаті виконання методу не повертається ніяке значення, тоді говорять, що повертається порожній тип), назву методу, перелік аргументів методу (може бути порожнім).

Наприклад,

```
public void seize()  {
   if(state==0)
state = state+1;
   else
numUnserv++;
}
```

Методи доступу за загальноприйнятим стандартом повинні мати назву «getA()», «setA» (…), де A – назва поля.

Наприклад,

```
public int getState()   {
       return state;
}
```

Типи даних й оператори мови Java докладно описані у багатьох підручниках і на сайтах, наприклад:

http://math.sgu.ru/sites/chairs/prinf/materials/java/lesson1.htm.

Корисно самостійно опрацювати приклади, наведені на таких сайтах:

http://www.frolov-lib.ru/programming/javasamples/vol1/vol1_1/index.html

http://www.frolov-lib.ru/programming/javasamples/vol1/vol1_5/index.html

Опис класу з використанням мови програмування Java

Наведемо приклад опису класу об'єктів, що відтворюють функціонування черги. Об'єкт «черга» характеризується кількістю елементів, що знаходяться в очікуванні і кількістю усіх елементів, які знаходились в очікуванні протягом часу спостереження. Залежно від зовнішніх обставин відбувається або зайняття місця в черзі або вивільнення місця в черзі.

Наприклад,

```
package smallsmo;  //  назва пакету класів

public class Queue {

  //     поля класу
    private int state;
    private int quantity;

//     конструктори класу
    public Queue()  {   //конструктор «за замовчуванням»
          state = 0;
          quantity = 0;
    }

    public Queue(int startState)  {
          state = startState;
          quantity = 0;
    }

//   методи класу
    public int getQuantity()  {   // метод доступу до значення поля
          return quantity;
    }

    public int getState()   {    // метод доступу до значення поля
          return state;
    }
```

```
   public void seize()  {
     state++;
   }

   public void realize() {
 quantity++;   // підрахунок кількості оброблених
 if (state>0)
                    state = state-1; //  або state--;
          else
System.out.println («Звільнити чергу не можливо-черга порожня»);
   }

} //кінець
```

Об'єкт. Створення та знищення об'єктів

Основне призначення класу – створення об'єктів із заданою структурою даних і заданими методами для їх обробки. Клас – це тип даних, а об'єкт - екземпляр класу. Створення об'єктів здійснюється з використанням конструктора класу. Знищення об'єктів здійснюється автоматично «збирачем сміття» віртуальної машини Java.

Створення екземпляру класу здійснюється за допомогою ключового слова «new», що означає виклик конструктора класу і конструктора класу з параметрами чи без.

```
Queue q = new Queue(); // створення екземпляру класу
```

Головна програма

Головний клас повинен містити метод «main», тому що саме з нього розпочинається виконання програми. Не можна змінювати назву цього методу чи список аргументів(!!). Потрібно завжди писати:

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

При використанні інтегроване.

Наведемо приклад головної програми, в якій здійснюється тестування класу «Queue»:

```
package smallsmo;  //  назва пакету класів
public class Main { // назва класу може бути іншою
public static void main(String[] args) {  //  початок main-методу

   Queue q = new Queue(); // створення екземпляру (об'єкта) класу
```

```
System.out.println("Поточний стан черги"+q. getState()
+"Кількість оброблених"+q. getQuantity());
   q. seize();
System.out.println("Поточний стан черги"+q. getState()
+"Кількість оброблених"+q. getQuantity());
   q. seize();
   q. seize();
   q. realize ();
System.out.println("Поточний стан черги"+q. getState()
+"Кількість оброблених"+q. getQuantity());
  } // кінець main-методу
} //кінець опису класу
```

Статичні методи

Є методи, для яких існування екземпляру класу не потрібне, тому вони можуть викликатись, навіть якщо жоден екземпляр класу не створений. Виконуючи обчислення в такому методі, не використовуються значення полів класу (окрім статичних). Такі методи описують з модифікатором «static».

Важливим прикладом такого методу є main-метод. У попередньому прикладі екземпляр класу «Main» не створюється, а запускається статичний метод цього класу «main».

Важливим прикладом класу, який містить тільки статичні методи є клас «Math», який призначений для виконання математичних розрахунків. Методами цього класу є наприклад, методи «Math.sqrt()»; «Math.sin()»; «Math.atan()»; «Math.pow()».

Наведемо приклад головної програми, в якій здійснюється розрахунок об’єма кулі.

```
public class Main { //назва класу може бути іншою
public static void main(String[] args) {  //  початок main-методу
 double R = 3.5;
 System.out.println("V = "+3./4.*c(R,3.)*Math.PI);
  } // кінець main-методу
} //кінець опису класу
```

Статичні поля

Поле, яке описане з модифікатором «static», існує в одному екземплярі для усіх об’єктів класу. Будь-який екземпляр класу має доступ до статичного поля, але це спільний доступ до одного і того ж самого поля. Якщо поле не статичне, то усі об’єкти містять копію цього поля і доступ об’єкт має тільки до локальної копії поля. Наприклад, потрібно створити ідентифікатор об’єктів:

```
public class avto{
…
private static int nextId=1; //наступний номер об’єкта
private int id;

public avto(){
…
id = nextId;  // ідентифікаційний номер об’єкта
nextId++;
}
…
        }
```

2. Завдання до роботи

Створити клас згідно з варіантом завдання. У головній програмі створити екземпляр класу і виконати розрахункові дії з використанням методів класу.

Дослідити можливість використання кількох конструкторів класу у різних мовах ООП; можливість розробки класу без конструктора класу. Дослідити механізм *інкапсуляції* даних.

Дослідити поняття статичного поля класу.

Дослідити поняття статичного методу класу. Опрацювати математичні розрахунки з використанням класу «Math». З використанням документації Java ознайомитись з методами класу та програмним кодом класу.

3. Варіанти

1.1) Клас «населений пункт».

1.2) Клас «авто».

1.3) Клас «світлофор».

1.4) Клас «студент».

1.5) Клас «аудиторія».

1.6) Клас «співробітник».

1.7) Клас «вершина графу».

1.8) Клас «вкладник банку».

2.1) Розрахунок відстані між точками у тривимірному просторі.

2.2) Розрахунок об’єму конуса для заданого діаметру основи конуса та заданого кута нахилу твірної до площини основи.

2.3) Розрахунок дальності польоту тіла, кинутого під заданим кутом до горизонту (початкова швидкість задана).

2.4) Розрахунок висоти, на яку підніметься тіло, підкинуте догори (задана початкова швидкість).

2.5) Розрахунок суми депозиту за умов капіталізації річних відсотків (задані початковий вклад, річний відсоток, кількість років).

2.6) Розрахунок кута нахилу твірної конуса до площини основи за умови заданого об’єму конуса й заданого діаметру основи конуса.

2.7) Розрахунок кута до горизонту для досягнення заданої дальності польоту тіла, кинутого під кутом (початкова швидкість задана).

2.8) Розрахунок кількості років для накопичення заданої суми депозити за умов капіталізації річних відсотків (задані початковий вклад, річний відсоток).

Хід виконання роботи

1. Вивчити характерні риси ООП.
2. Вивчити типи даних мови Java.
3. Вивчити оператори мови Java.
4. Побудувати клас мовою Java відповідно до варіанта завдання.
5. Побудувати головну програму та виконати тестування побудованого класу.
6. Дослідити механізм інкапсуляції даних.
7. Ознайомитись з класом «Math» мови Java та виконати математичні розрахунки у головній програмі.

ЛАБОРАТОРНА РОБОТА № 2

Створення класів

Мета роботи: потрібно створити клас для банківського рахунку, як показано в методичних вказівках, і додати до нього певні методи, а також створити клас, що наслідує. За бажанням можна дописати ще кілька методів для реалізації якихось операцій.

1. Створення класів для банківського рахунку

Класи і об’єкти. Наслідування

Створимо наступний клас для рахунку – Account.

```
class Account {
 int accountNo;
 String accountName;

 double balance;
}
```

А також клас із методом «main», в якому створимо об’єкт класу рахунку і перевіримо його роботу, наприклад, так:

```
class CreateAccount {
public static void main(String[] args) {
Account fredsAccount = new Account();
 fredsAccount.accountNo = 123;
 fredsAccount.accountName = "Fred";
 fredsAccount.balance = 50;
 System.out.println("A/c no: " + fredsAccount.accountNo +
 " A/c name: " + fredsAccount.accountName + " Balance: "
 + fredsAccount.balance);
 }
 }
```

Наразі у нашого класу рахунку немає жодного методу. Додаймо до нього методи, наприклад, для створення депозиту, очищення балансу і друку стану нашого балансу, наприклад, у такий спосіб:

```
public void deposit(double amount) {
balance = balance + amount;
}
public void clearBalance() {
balance = 0;
}
public double withdraw(double amount) {
if (balance − amount < 0) {
System.out.println("Insufficient Funds");
}
else {
balance = balance − amount;
}
return balance;
}
```

Протестуйте подані методи і для вашого об’єкта.

2. Завдання до роботи

Створіть метод конструктора класу, в якому задайте полям наступні значення:

```
accountNo = 0;
accountName = null;
balance = 0.0;
```

Перевантажте метод для депозиту так, щоб він друкував значення балансу лише, якщо балансу перевищує певний заданий рівень.

3. Наслідування

Тепер створімо клас, який мав усі ті ж характеристики, що і клас рахунку, за винятком того, що в нього обов'язково має бути мінімальне значення. Назвемо його «SavingAccount». Він буде підкласом «Account».

```
class SavingsAccount extends Account {
 double minBalance;

public SavingsAccount(int no, String name, double balance) {
 super(no, name, balance);
 minBalance = 100;
 }

public double withdraw(double amount) {
 if (balance − amount < minBalance) {
 System.out.println("Insufficient Funds");
 } else {
 balance = balance − amount;
 }
return balance;
 }
 }
```

4. Завдання

1. Перевірте роботу створеного класу в методі «main»
2. Напишіть метод для визначення депозиту, який би виводив значення із батьківського.

ЛАБОРАТОРНА РОБОТА № 3

Перевизначення методів (overriding)

Мета роботи: потрібно перевизначити метод за умови, якщо метод не відзначений як «final».

Перевага перевизначення

Якщо підклас успадковує метод суперкласу то є можливість перевизначити метод за умови, що метод не відзначений як «final».

Перевага перевизначення: здатність визначати поведінку, що є специфічна для типу підкласу, перевизначити функціональність наявного методу.

Приклад:

```
class Animal{

  public void move(){
    System.out.println("Animals can move");
  }
}

class Dog extends Animal{

  public void move(){
    System.out.println("Dogs can walk and run");
  }
}

public class TestDog{

  public static void main(String args[]){
    Animal a = new Animal(); // об’єкт і посилання на Animal
    Animal b = new Dog(); // посилання на Animal, але об’єкт Dog

    a.move();// запускає методи в класі Animal

    b.move();//запускає методи в класі Dog
  }
}
```

У наведеному вище прикладі ви можете побачити, що, хоча **b** типу «Animal», метод переміщення працює в класі «Dog». Причина цього полягає в тому: під час компіляції перевірка проводиться над типом посилання. А вже під час виконання JVM з'ясовує тип об'єкта, і буде працювати метод, який належить до того конкретного об'єкта.

Правила для перевизначення методів

Список аргументів має бути точно таким самим, як і в методі, що перевизначається.

Тип повертання має бути таким самим або підтипом оголошеним в початковому методі суперкласу.

Доступ не може бути більш строгим. Наприклад, якщо метод суперкласу є «public», то перевизначений метод не може бути «private» чи «protected».

Метод «final» не може бути перевизначеним

Метод, оголошений як «static», не може бути перевизначеним, але може бути пере оголошеним.

Якщо метод не може наслідуватися, то не може і перевизначатися.

Конструктор не може бути перевизначеним.

Підклас в середині того ж пакету може перевизначити будь-які методи суперкласу, що не оголошені як «private» чи «final».

Підклас у іншому пакеті може перевизначити тільки не «final» методи, оголошені як «public» чи «protected».

Використання ключового слова «super»

Якщо потрібно викликати метод суперкласу – використовуємо ключове слово «super»:

```
class Animal{
  public void move(){
    System.out.println("Animals can move");
  }
}

class Dog extends Animal{

  public void move(){
    super.move(); // Викликає метод суперкласу
    System.out.println("Dogs can walk and run");
  }
}

public class TestDog{

  public static void main(String args[]){

    Animal b = new Dog(); // посилання Animal, але об’єкт  Dog
    b.move(); //Запускає метод в класі Dog

  }
}
```

1. Завдання до роботи

Створити клас банку. У ньому метод для визначення процентної ставки. Від класу банку утворити декілька підкласів конкретних банків (наприклад «ПриватБанк», «Аваль» тощо.), з методами процентної ставки вже для них. Також додати методи для основних банківських послуг.

ЛАБОРАТОРНА РОБОТА № 4

Абстрактні класи. Інтерфейс. Поліморфізм

Мета роботи: потрібно засвоїти дві пов'язані теми – абстрактні класи й інтерфейси.

Щодо 1-го завдання про написання методу площі і периметру – мають бути два методи з правильно визначеними модифікаторами, і ці методи повертатимуть 0, оскільки для точки ми не можемо порахувати ані периметр, ані площу.

У другому потрібно доповнити приклад.

1. Абстрактний клас в Java

Абстрактні класи Java використовуються, щоб оголосити загальні характеристики підкласів. Абстрактний клас не може бути інстанційований. Він може бути використаний тільки як суперклас для інших класів, які розширюють абстрактний клас. Абстрактні класи оголошуються з ключовим словом «Abstract». Абстрактні класи використовуються для забезпечення шаблону або дизайну для конкретних підкласів вниз по дереву успадкування.

Абстрактний клас може включати в себе методи, які не містять реалізацію. Вони називаються абстрактні методи. Декларація абстрактного методу повинна потім закінчуватися крапкою з комою, а не блоком. Якщо клас має які-небудь абстрактні методи, оголошені або успадковані, весь клас повинен бути оголошений як абстрактний. Абстрактні методи використовуються для забезпечення шаблону для класів, які успадковують абстрактні методи.

Ми можемо реалізувати абстрактний клас для форми, щоб потім малювати лінії, кола, трикутники тощо.

Приклад:

```
abstract class Shape {

    public String color;
    public Shape() {
    }
```

```
	public void setColor(String c) {
		color = c;
	}
	public String getColor() {
		return color;
	}
	abstract public double area();
}
```

Клас точки, що розширює клас форми

Зверніть увагу, що для того, щоб створити об'єкт точки, його клас не може бути абстрактним. Це означає, що всі абстрактні методи класу «Shape» повинні бути реалізовані в класі «Point».

Підклас повинен визначити реалізацію для кожного абстрактного методу абстрактного суперкласу, або сам підклас також буде абстрактни:

```
.
public class Point extends Shape {

	static int x, y;
	public Point() {
		x = 0;
		y = 0;
	}

	public static void print() {
		System.out.println("point: " + x + "," + y);
	}

}
```

2. Завдання до роботи

Дописати до класу точки методи для знаходження периметру і площі. У класі з методом «main()» викликати метод «print()» і подивитися на результат.

3. Інтерфейс

Великий недолік використання абстрактних класів полягає у неможливості множинного наслідування. Тобто, коли клас розширює абстрактний клас, він не може розширювати будь-який інший клас. Ця проблема вирішена завдяки створенню потужної конструкції, що має назву інтерфейсу.

Інтерфейс може бути використаний для визначення родового шаблону, а потім можна використати один або кілька абстрактних класів, щоб визначити часткові реалізації інтерфейсу. Інтерфейси просто вказують декларацію методу (неявно «public» і «abstract») і можуть містити тільки поля (які неявно «public», «static», «final»). Визначення інтерфейсу починається з ключового слова «interface». Інтерфейс, як і абстрактний клас, не може бути інстанційований.

Один інтерфейс може розширювати кілька інтерфейсів (множинне наслідування). Java не підтримує множинне спадкування, але це дозволяє розширити один клас і реалізувати безліч інтерфейсів.

Якщо клас, який реалізує інтерфейс не визначає всі методи інтерфейсу, то він повинен бути оголошений як абстрактний і визначення методів повинно бути надано підкласу, який розширює абстрактний клас.

Приклад:

```
interface Shape {

	public double area();
	public double volume();
}

public class Point implements Shape {

	static int x, y;
	public Point() {
		x = 0;
		y = 0;
	}
	public double area() {
		return 0;
	}
	public double volume() {
		return 0;
	}
	public static void print() {
		System.out.println("point: " + x + "," + y);
	}
	public static void main(String args[]) {
		Point p = new Point();
		p.print();
	}
}
```

Оскільки інтерфейси можуть розширювати декілька інтерфейсів, то можливий наступний синтаксис створення інтерфейсу із вказанням батьківського інтерфейсу через кому:

```
public interface MySubInterface extends
   SuperInterface1, SuperInterface2 { // Батьківські інтерфейси
   public void sayItAll();
}
```

4. Поліморфізм

Поліморфізм означає одне ім’я для декількох форм. Існує три форми поліморфізму в Java:

1. Перевантаження методів (Compile time polymorphism).
2. Перевизначення методів через наслідування (Run time polymorphism).
3. Перевизначення методів через інтерфейси (Run time polymorphism).

5. Завдання до роботи

Додати реалізацію інтерфейсу форми для геометричних фігур – прямокутника, паралелепіпеда, сфери, трикутника, трикутної призми:

```
interface Shape {

        public double area();
        public double volume();
}

class Cube implements Shape {

        int x = 10;
        public double area( ) {

           return (6 * x * x);
        }
        public double volume() {
                return (x * x * x);
        }
}
class Circle implements Shape {
        int radius = 10;
        public double area() {
```

```
            return (Math.PI * radius * radius);
        }
        public double volume() {
            return 0;
        }
}
public class PolymorphismTest {

        public static void main(String args[]) {
            Shape[] s = { new Cube(), new Circle() };
            for (int i = 0; i &lt; s.length; i++) {
                System.out.println("The area and volume of " + s[i].getClass()
                        + " is " + s[i].area() + " , " + s[i].volume());
            }
        }
}
```

ЛАБОРАТОРНА РОБОТА № 5

Обробка винятків

Мета роботи: здійснити обробку винятків.

У кінці роботи завдання. Там є питання. На 5-те питання треба відповісти, що «...не виконається, адже виникнення арифметичної помилки підведе до її обробки першим блоком «catch» і після цього продовжиться виконання методу «main()», і код, що спричиняє вихід за межі масиву ніколи не буде виконаним». На 8-ме питання відповідь така: батьківський клас «RuntimeException» або «Exception» (видно з дерева ієрархії).

1. Що таке винятки?

Виняток (або виняткова подія) є проблемою, яка виникає під час виконання програми. Коли відбувається виняток, нормальний потік програми порушується і програма / застосунок завершується аварійно, що не рекомендується, тому ці винятки повинні бути оброблені.

Винятки можуть спричиняти помилки користувача, програміста або інші фізичні ресурси (наприклад, відмова обладнання).

На підставі цього ми маємо три категорії винятків (ви повинні розуміти їх, для того, щоб знати, як обробка виключень працює в Java):

Винятки, що перевіряються (checked exceptions) – винятки, які відбуваються в момент компіляції; також мають назву винятки часу компіляції (compile time exceptions). Вони не можна просто ігнорувати під час компіляції, програміст повинен подбати про них (ручки).

Винятки, що не перевіряються (unchecked exceptions) – винятки, які виникають під час виконання компіляції. Вони також називаються «Runtime Exceptions». Включають помилки програмування, такі як логічні помилки або неправильне використання в API. Ігноруються під час компіляції.

Помилки (error) – проблеми, які виникають поза контролем користувача або програміста. Помилки (рис. 1) зазвичай ігноруються в коді, адже зрідка коли можна що-небудь зробити з помилками. Наприклад, якщо відбувається переповнення стека, виникає помилка (рис. 2). Вони також ігноруються під час компіляції.

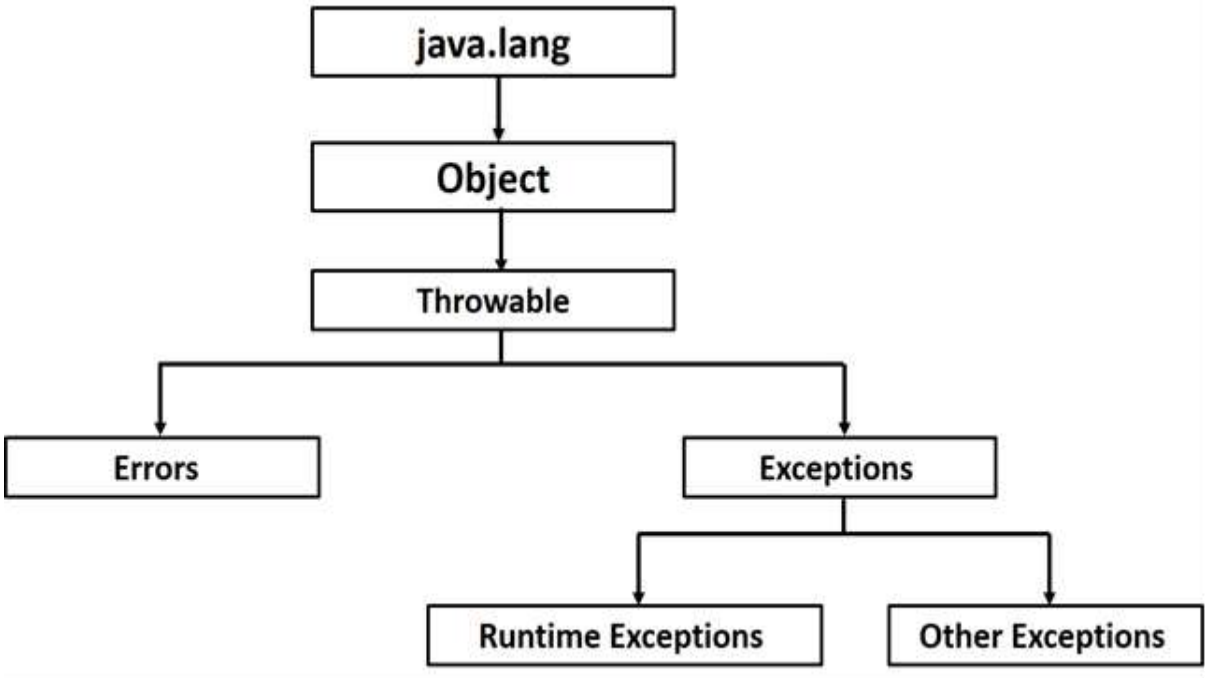

Рис.1. Ієрархія класів помилок

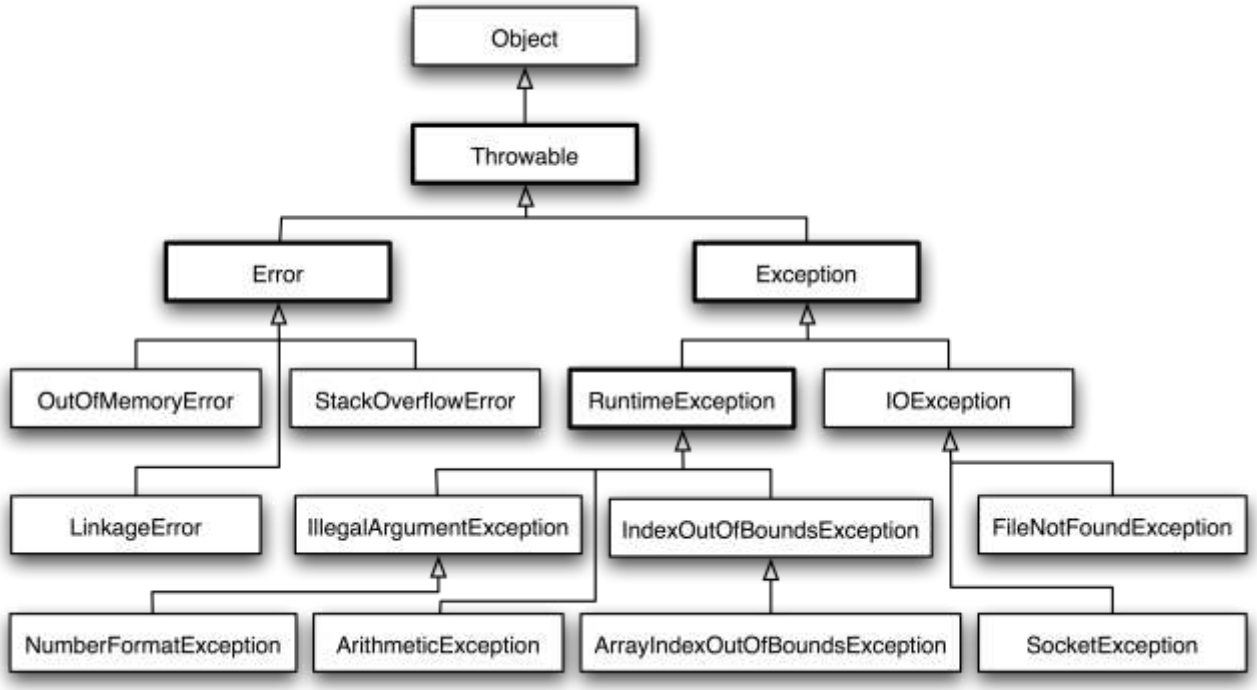

Рис. 2. Деякі підкласи класів помилок

2. Методи винятків

Нижче наведено список важливих методів, доступних в класі «Throwable»:

- public String getMessage(): повертає докладне повідомлення про виняток. Ініціалізовано в конструкторі «Throwable»;

- public Throwable getCause(): повертає причину винятку так, як представлено об’єктом «Throwable»;

- public String toString(): повертає ім'я класу, об'єднане з результатом методу «getMessage()»;

- public void printStackTrace(): повертає ім'я класу, об'єднане з результатом методу «getMessage()»;

- public StackTraceElement [] getStackTrace(): повертає масив, що містить усі елементи трасування стека. Елемент з індексом 0 представляє вершину стека викликів, і останній елемент в масиві являє собою метод у нижній частині стека викликів;

- public Throwable fillInStackTrace(): заповнює трасування стека цього «Throwable» об'єкта з поточного трасування стека, додаючи будь-яку інформацію до сигнатури «trace.hod».

3. Відловлювання винятків

Код, що може спричинити виняток, оброблюється блоком «Try» / «Catch» і виглядає так:

```
try
{
  //Protected code
}catch(ExceptionName e1)
{
  //Catch block
}
```

або містить декілька блоків:

```
try
{
  //Protected code
}catch(ExceptionType1 e1)
{
  //Catch block
}catch(ExceptionType2 e2)
{
  //Catch block
}catch(ExceptionType3 e3)
{
  //Catch block
}
```

Також може містити блок finally (блок обов’язково виконається):

```
try
{
  //Protected code
}catch(ExceptionType1 e1)
{
  //Catch block
}finally
{
  //The finally block always executes.
}
```

Приклад № 1

```
import java.io.*;
public class ExcepTest{

  public static void main(String args[]){
    try{
      int a[] = new int[2];
      System.out.println("Access element three :" + a[3]);
    }catch(ArrayIndexOutOfBoundsException e){
      System.out.println("Exception thrown  :" + e);
    }
    System.out.println("Out of the block");
  }
}
```

4. Ключові слова «throws»/»throw»

Якщо потрібно викликати помилку, використовується ключове слово «throw».

Якщо метод здатний викликати винятки, які сам не обробляє, він повинен оголосити про таку поведінку, щоб методи, які його викликають, могли захистити себе від цих винятків. Для завдання списку винятків, які можуть викликатись методом, використовується оператор «throws». Якщо метод в явному вигляді (тобто з допомогою оператора «throw») збуджує виключення відповідного класу, тип класу винятків повинен бути вказаний в операторі «throws» в оголошенні цього методу.

Приклад №2

```
class ThrowsDemo {
static void procedure() throws IllegalAccessException
{
        System.out.println(" inside procedure");
        throw new IllegalAccessException("demo");
}
public static void main(String aigs[]) {
try {
        procedure();
}
catch (IllegalAccessException e) {
        System.out.println("caught" + e);
}
}
}
```

5. Винятки, визначені користувачем

Ви можете створювати свої власні винятки в Java. Під час написання власних класів винятків необхідно пам'ятати такі моменти:

1. Усі винятки повинні бути нащадком «Throwable».

2. Якщо ви хочете написати перевірювальний виняток, необхідно розширити клас «Exception».

3. Якщо ви хочете написати виняток часу виконання, необхідно розширити клас «RuntimeException».

Ми можемо визначити наш власний клас винятків так:

```
class MyException extends Exception{
}
```

6. Завдання до роботи

1. Написати програмний код, який спричиняє арифметичну помилку (Arithmetic Exception) (наприклад, ділення на нуль).

2. Обробити код, помістивши його в блок «try».

3. Дописати код блоку «try» так, щоб в наступному рядку він спричиняв помилку виходу за межі масиву. Обробити помилку блоком «catch».

4. Додати обробник «catch» і для цієї помилки.

5. Дати відповідь на питання: чи виконається другий «catch»?

6. До «Throwable» об'єктів застосувати метод «getMessage()».

7. Додати блок «finally».

8. Дати відповідь на питання: якби викликані нами дві помилки можна було б обробити однаково одним блоком «catch», який тип помилки можливо було б вказати?

9. Створити клас власної помилки для програми з лабораторної роботи. №4, наприклад, не допустити від'ємних значень параметрів геометричних фігур.

ЛАБОРАТОРНА РОБОТА № 6

Розробка інтерфейсу користувача з використанням бібліотеки JavaFX

Мета роботи: розробити інтерфейс користувача з використанням бібліотеки JavaFX.

Окремо завдання до роботи не виділяються, вони включені до теоретичних відомостей.

1. Використання парадигми MVC під час розробки проекту

MVC (Model View Controller) – архітектурний шаблон програмного забезпечення, суть якого полягає у поділі системи на три частини: модель, вигляд (відображення), контролер (керування). Такий підхід дозволяє організувати програмний код так, що зміни, внесені до інтерфейсу користувача мінімально впливатимуть на роботу з даними, яка реалізована в моделі, а зміни в моделі даних можуть здійснюватися без змін в інтерфейсі користувача.

Діаграма взаємодії основних частин подана на рис. 3.

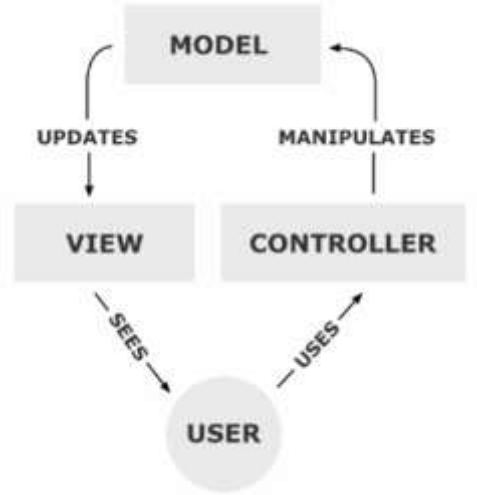

Рис. 3. Діаграма взаємодії основних частин

У першій частині роботи розробимо калькулятор з використанням MVC підходу. Користувач має можливість задавати два значення, над цими значеннями відбудуться операції додавання, віднімання, множення та ділення.

Почнемо з розробки трьох класів – «Model», «View», «Controller», а також класу для демонстрації роботи програми з методом «main()».

Розробка класу «Model»

У моделі повинні міститися дані, над якими проводитимуться розрахунки, а також самі розрахунки, які нам потрібні. Модель «не знає» про існування вигляду.

Для класу додайте змінні.

Напишіть методи для вказаних вище арифметичних операції.

Розробка класу «View»

Вигляд не знає про існування Контролера. Його єдина робота полягає в показі користувачу того, що він має бачити. У ньому не відбувається жодних розрахунків, він може передавати інформацію, введену користувачем туди, де вона потрібна.

Оскільки на цьому етапі ми ще не вміємо використовувати графічні бібліотеки, то відображувати значення будем в консоль.

Розробіть метод для виведення обрахованих значень у консоль користувачеві.

Розробка класу «Controller»

Контролер забезпечує взаємодію «Model» і «View», викликаючи необхідні методи з них.

Створіть об’єкти класів «Model» і «View», з модифікатором «private».

Напишіть методи взаємодії.

Демонстраційний клас

У методі «main()» створіть об’єкти усіх трьох класів MVC. Передайте необхідні значення змінним і виведіть результати обчислень у консоль.

2. JavaFX

JavaFX – набір графічних і мультимедіа пакетів, що дають змогу створювати дизайн, розробку, налагодження клієнтських застосунків для різних платформ.

Основні характеристики:

1. FXML and Scene Builder – FXML – декларативна мова розмітки, що базується на XML і дозволяє розроблювати графічний інтерфейс.
2. Scene Builder – програма для розробки інтерфейсу «вручну».
3. Built-in UI controls and CSS – вбудовані контролери інтерфейсу користувача і підтримка CSS.
4. Підтримка 3D графіки.
5. Підтримка формату «rich text».
6. Підтримка «multi-touch» і багато інших.
7.

Основні концепції JavaFX

Усі необхідні елементи знаходять в пакетах, що починаються з префіксу «javafx».

Найнеобхіднішими є наступні 4 елемента: «javafx.application», «javafx.scene», «javafx.stage», javafx.layout

Основною метафорою, реалізованою в JavaFX є «Stage». Це контейнер, на якому розміщується «Scene». Для зручності можна уявити «Stage» як сцену в театрі, а «Scene» – як різні декорації і саму виставу, що проходить.

«Scene» містить в собі усі інші необхідні елементи взаємодії (рис. 4).

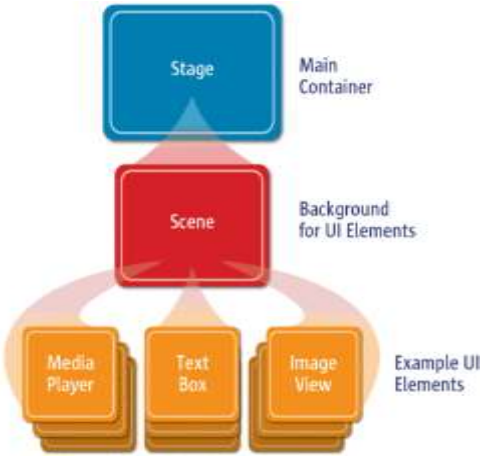

Рис. 4. Необхідні елементи взаємодії «Scene»

Елементи, що розміщуються в «Scene», утворюють дерево (рис. 5).

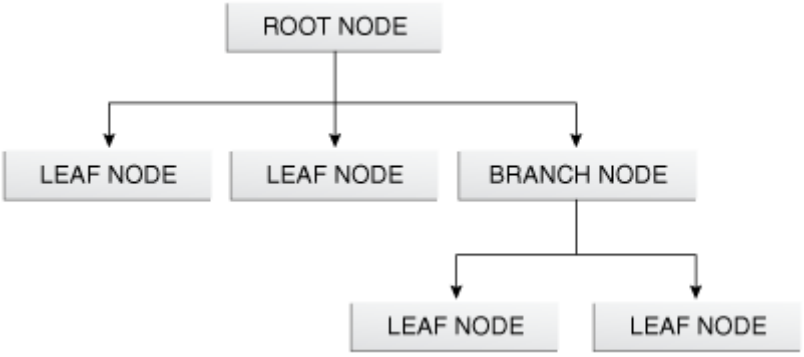

Рис. 5. Дерево елементів, що розміщуються в «Scene»

У класі «Application» містяться три основних методи, які ви можете перевизначити для створення власної програми. Це методи:

- «void init()»;
- «abstract void start (Stage primaryStage)»;
- «void stop()».

Метод «start()» викликає метод «init()». «Stop()» викликається при закритті вікна. Запуск відбувається завдяки методу «public static void launch(String ... args)».

Загалом проекти будуть мати наступну структуру:

```
// A JavaFX application skeleton.
import
import
import
```

```
import
javafx.application.*;
javafx.scene.*;
javafx.stage.*;
javafx.scene.layout.*;
public class JavaFXSkel extends Application {
public static void main(String[] args) {
        System.out.println("Launching JavaFX application.");
                // Start the JavaFX application by calling launch().
        launch(args);
}

// Override the init() method.
public void init() {System.out.println("Inside the init() method.");
}

// Override the start() method.
public void start(Stage myStage) {
        System.out.println("Inside the start() method.");
// Give the stage a title.


myStage.setTitle("JavaFX Skeleton.");

// Create a root node. In this case, a flow layout pane
// is used, but several alternatives exist.
FlowPane rootNode = new FlowPane();
// Create a scene.

Scene myScene = new Scene(rootNode, 300, 200);
// Set the scene on the stage.
myStage.setScene(myScene);
// Show the stage and its scene.
myStage.show();
}
// Override the stop() method.
public void stop() {
System.out.println("Inside the stop() method.");
}
}
}
```

3. Два шляхи розробки проектів з JavaFX

Розробка програми в середовищі JavaSceneBuilder2.0

Існує два шляхи написання програм з використанням JavaFX:

1. Виключно кодом мови.
2. З використанням FXML файлів для розробки інтерфейсу.

У поданій роботі розглянемо детальніше 2-й варіант.

Для початку зручно встановити програму для розробки інтерфейсу JavaSceneBuilder2.0.

Завантажте і встановіть його звідси:

http://www.oracle.com/technetwork/java/javafxscenebuilder-1x-archive-2199384.html

Як видно, на панелі ліворуч розміщено необхідні компоненти, праворуч – їхні властивості. Компоненти з лівої частини потрібно перетягувати на центральну частину середовища і задавати їм необхідні візуальні параметри, використовуючи праву панель.

СПИСОК ЛІТЕРАТУРИ

1. Java. Пособие для проведения практических занятий. Режим доступу: http://math.sgu.ru/sites/chairs/prinf/materials/java/lesson1.htm.
2. Библиотека примеров приложений Java. Базовые типы данных. Режим доступу:http://www.frolov-lib.ru/programming/javasamples/vol1/vol1_1/index.html.
3. Библиотека примеров приложений Java. Операторы структурного программирования. Режим доступу: http://www.frolov-lib.ru/programming/javasamples/vol1/vol1_5/index.html.

Для нотаток

Навчально-методичне видання

КРОС-ПЛАТФОРМНЕ ПРОГРАМУВАННЯ

Методичні вказівки
до виконання лабораторних робіт
для студентів 3-го курсу
спеціальності 122 «Комп'ютерні науки»

Укладачі: **ХРОЛЕНКО** Володимир Миколайович
ГОЛЕНКОВ Володимир Геннадійович
ГОЦ Владислав Володимирович

Випусковий редактор *В. С. Сасько*
Комп'ютерне верстання *А. П. Селівестрової*

Підписано до друку 15.10.2020 р. Формат 60 × 84 1/ 16
Ум. друк. арк. 1,86. Обл.-вид. арк 2,0
Електронний документ. Вид . № 27/III-20.

Видавець і виготовлювач:
Київський національний університет будівництва і архітектури

Повітрофлотський проспект, 31, Київ, Україна, 03680

Свідоцтво про внесення до Державного реєстру суб'єктів
видавничої справи ДК № 808 від 13.02.2002 р.