Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна

ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ (МОВА JAVA)

Методичні рекомендації
до виконання практичних та індивідуальних (розрахунково-графічних) робіт для здобувачів вищої освіти факультету математики і інформатики першого (бакалаврського) рівня вищої освіти

Електронний ресурс

Харків – 2025

УДК 004.42+004.43
О-12

Рецензенти:

О. Г. Толстолузька – доктор технічних наук, старший науковий співробітник, професор кафедри теоретичної та прикладної системотехніки факультету комп'ютерних наук Харківського національного університету імені В. Н. Каразіна;

М. О. Момот – кандидат технічних наук, доцент, доцент закладу вищої освіти кафедри комп'ютерних наук та інформаційних технологій Національного аерокосмічного університету ім. М. Є. Жуковського «Харківський авіаційний інститут», м. Харків.

Затверджено до розміщення в мережі Інтернет рішенням Науково-методичної ради Харківського національного університету імені В. Н. Каразіна (протокол № 8 від 28 березня 2025 року)

Об'єктно-орієнтоване програмування (мова JAVA) : методичні рекомендації до
О-12 виконання практичних та індивідуальних (розрахунково-графічних) робіт для здобувачів вищої освіти факультету математики і інформатики першого (бакалаврського) рівня вищої освіти [Електронний ресурс] / А. Г. Морозова, М. В. Владимирова, І. Т. Зарецька, Л. П. Бєлова. – Харків : ХНУ імені В. Н. Каразіна, 2025. – (PDF 49 с.)

Методичні рекомендації містять теоретичний матеріал, завдання для практичних та індивідуальних (розрахунково-графічних) робіт, зразки виконання розрахунково-графічних робіт, питання для самоконтролю, завдання для самостійної роботи, рекомендовану літературу.

Навчально-методичне видання призначається для здобувачів вищої освіти факультету математики і інформатики першого (бакалаврського) рівня вищої освіти, які вивчають дисципліну «Об'єктно-орієнтоване програмування (мова JAVA)».

УДК 004.42+004.43

ЗМІСТ

Практична робота № 1
РОЗРОБКА JAVA КЛАСУ.

Теоретичний матеріал

Опис класу:

```
access_modifier class <class_name> {
        data member;
        method;
        constructor;

}
```

Модифікатори доступу (access_modifier)

- **public** – загальнодоступний;
- **protected** – захищений та пакетний;
- **private** – закритий;
- **default (package visible)** – пакетний рівень доступу.

Ім’я класу (class_name)

Починається з великої літери.

Якщо ім’я складається з декілька слів, то вони пишуться разом без пробілів, при цьому кожне слово пишеться з великої літери, наприклад ColorPoint, Point.

Якщо клас оголошений як public, то ім’я класу повинно співпадати з ім’ям файлу java, в якому він описаний.

В одному java файлі може бути лише один public клас та необмежена кількість інших (не public) класів.

Поле класу (data member)

- Змінна, зв'язана з класом або об'єктом.
- Всі дані об'єкта зберігаються в його полях.
- Доступ до полів здійснюється по імені.
- Починається з маленької літери.

Метод класу (method)

- Функції, визначені всередині класу.
- Використовуються для виконання різних дій або операцій над об'єктами класу.
- Можуть приймати аргументи та повертати значення.
- Починається з маленької літери.

Конструктор класу (constructor)

- Має те саме ім'я, що і клас.
- Нічого не повертає.
- Може мати будь-який модифікатор доступу.
- Не може бути abstract, static, final, native, synchronized.

Приклад класу

```
//Class Declaration
public class Dog {

        // data members
        String name;
        String breed;
        int age;
        String color;

        // Constructor Declaration of Class
        public Dog(String name, String breed, int age, String color){
            this.name = name;
            this.breed = breed;
            this.age = age;
            this.color = color;
        }

        // method 1
        public String getName() {
          return name;
        }
        // method 2
        public String getBreed() {
          return breed;
        }
        // method 3
        public int getAge() {
          return age;
        }
        // method 4
        public String getColor() {
          return color;
        }
}
```

Клас Object

Усі класи JAVA неявно успадковуються від класу Object. Це суперклас для всіх класів.

Основні методи:

- *public String toString();*
- *public int hashCode();*
- *public boolean equals(Object obj).*

toString() – метод повертає строкове представлення об'єкта. За замовчуванням він повертає рядок, що містить ім'я класу і хеш-код об'єкта. Метод toString() може бути перевизначений у користувацькому класі для надання більш корисної інформації про стан об'єкта.

hashCode() – метод повертає хеш-код об'єкта у вигляді цілого числа.

equals() – метод використовується для порівняння об'єктів на рівність. Він перевіряє, чи є два об'єкти однаковими або різними. За замовчуванням метод equals() порівнює посилання на об'єкти, але його можна перевизначити в користувацькому класі для порівняння значень полів об'єкта.

Завдання

1. Розробити в середовище розробки JAVA клас Point. Реалізувати метод перевірки належності точки колу з центром в заданій точці та заданого радіусу.
2. Перевизначити методи toString та equals.
3. Розробити в середовище розробки JAVA клас для тестування з методом main.
4. В методі main реалізувати наступну логіку:
    a. Створити масив об'єктів класу Point, ініціалізувати довільними координатами так, щоб координати повторювалися.
    b. Вивести ті з об'єктів класу Point, які потрапили всередину кола з центром у точці (1, 2) та радіусом 5.
    c. З клавіатури ввести довільну точку та підрахувати кількість точок у масиві рівних заданій.

Перелік документів для включення до звіту ПР № 1

- постановка задачі;
- java код розробленого класу Point з коментарями;
- java код класу для тестування з методом main;
- скріншот результату роботи java-програми.

Питання для самоконтролю

1. З яких елементів складається заголовок класу?
2. З яких елементів складається заголовок методу?
3. Що таке конструктор у Java?
4. Які класи не успадковуються від Object?
5. Що таке модифікатори доступу?
6. У чому різниця між оператором == та методом equals в Java?
7. Назвіть основні методи класу Object?

Практична робота № 2
РОБОТА З МАСИВАМИ.

Теоретичний матеріал

Масив – це структура даних, у якій зберігаються елементи одного типу. Доступ до конкретного елементу здійснюється через його номер або індекс. Індекс починається з 0.

Рис. 1. Java масив

Оголошення масиву

- Оголосити посилання на масив:
```
dataType[] arrayName;
```
- Виділити пам'ять під необхідну кількість елементів N:
```
arrayName = new dataType[N];
```
- В мові Java масиви відносять до reference типів. Для отримання довжини масиву використовують поле *length*.

Приклад:

```
int[] myArray = new int[10];
System.out.println(myArray.length);
```

Висновок програми:
```
10
```

Кількість елементів не можна змінювати, але можна присвоїти посилання на інший масив.

Java контролює вихід за межі масиву генеруючи виключення ArrayIndexOutOfBoundsException.

Ініціалізація масиву

Після процедури виділення пам’яті під елементи масиву, ми отримуємо не порожній масив, а масив, заповнений значеннями за замовчуванням. Наприклад, у випадку *int* це будуть 0.

Якщо масив з даними типу посилання, то за замовчуванням у кожному елементі записані *null*.

Ініціалізація масиву – це заповнення його конкретними даними (не за замовчуванням).

```
//оголошення масиву із 4 елементів типу String, кожен елемент має null
   String[] seasons  = new String[4];
   //ініціалізація кожного елементу масиву
   seasons[0] = "Winter";
   seasons[1] = "Spring";
   seasons[2] = "Summer";
   seasons[3] = "Autumn";
АБО
String[] seasons  = new String[] {"Winter", "Spring", "Summer", "Autumn"};
АБО
String[] seasons  = {"Winter", "Spring", "Summer", "Autumn"};
```

Виведення масиву Java на екран

Вивести елементи масиву на екран (тобто в консоль) можна тільки по елементах, наприклад, за допомогою циклу *for*.

```
String[] seasons = new String[] {"Winter", "Spring", "Summer", "Autumn"};
for (int i = 0; i < seasons.length; i++) {
      System.out.println(seasons[i]);
}
```

Висновок програми:

```
Winter Spring Summer Autumn
```

Завдання

1. Розробити в середовище розробки JAVA клас NVector для роботи з n-вимірним вектором (задається як масив координат типу double).
2. Реалізувати конструктор копіювання для створення копії вектору.
3. Реалізувати розумну поведінку, зокрема методи обчислення довжини (норми) вектора, отримання та установка елементів вектора, виведення на екран, тощо.
4. Перевизначити методи toString та equals.
5. Розробити в середовище розробки JAVA клас для тестування з методом main.
6. В методі main реалізувати наступну логіку:
    a. Створити об'єкт класу NVector, ініціалізувати довільними координатами.
    b. Вивести вміст (координати) вектора на екран.
    c. Підрахувати довжину вектора та вивести на екран.

d. Змінити довільні координати вектора та вивести змінений вектор на екран.
e. Створити ще один вектор-копію першого вектора.
f. Порівняти два вектори та вивести результат на екран.

Перелік документів для включення до звіту ПР № 2

- постановка задачі;
- java код розробленого класу NVector з коментарями;
- java код класу для тестування з методом main;
- скріншот результату роботи java-програми.

Питання для самоконтролю

1. Що називається масивом?
2. Як створити масив?
3. Чи можемо змінити розмір масиву під час виконання?
4. Чи можете ви оголосити масив, не вказуючи його розмір?
5. Яке значення елементів масиву за замовчуванням?
6. Як роздрукувати елемент масиву?
7. Як порівняти два масиви?
8. Коли виникає виняток ArrayIndexOutOfBoundsException?

Практична робота № 3
СТАТИЧНІ ПОЛІ ТА МЕТОДИ КЛАСУ.

Теоретичний матеріал

Статичні змінні

Коли клас завантажується в пам'ять, для нього відразу створюється статичний об'єкт класу. Цей об'єкт зберігає статичні змінні класу (статичні поля класу).

Статичний об'єкт класу існує, навіть якщо не було створено жодного звичайного об'єкта класу.

Щоб створити статичну змінну класу, потрібно перед її ім'ям написати ключове слово ***static***. Оголошення статичної змінної має такий загальний вигляд:

```
static тип ім'я = значення;
```

Наприклад:

```
private static int count = 0;
```

На статичні поля поширюється дія специфікаторів доступу, тому статичні поля, описані як private, можна змінити за допомогою статичних методів.

Статичні поля доступні через ім'я класу.

```
Ім'яКласу.статична_змінна
```

Наприклад:

Змінна	Клас	Звернення до змінної з іншого класу
`public static int size = 100;`	`Solution`	`Solution.size`
`private static int count = 0;`	`Counter`	Змінну `private` поза її класом не видно

Різниця між статичними й нестатичними змінними

Звичайні змінні класу прив'язані до об'єктів свого класу (екземплярів класу), а статичні змінні — до статичного об'єкта класу.

Якщо екземплярів класу декілька, у кожному з них є своя копія нестатичних (звичайних) змінних класу. Статичні змінні класу завжди

існують тільки в одному екземплярі, який знаходиться всередині статичного об'єкта класу.

Статичні методи

Окрім статичних змінних, у класах можуть також бути статичні методи.

Звичайні методи прив'язані до об'єктів (екземплярів) класу й можуть звертатися до звичайних змінних класу (а також до статичних змінних і методів). Натомість статичні методи прив'язані до статичного об'єкта класу й можуть звертатися ***тільки*** до статичних змінних і/або інших статичних методів класу.

Щоб викликати звичайний метод класу, спочатку потрібно створити об'єкт цього класу, а тільки потім можна викликати метод для створеного об'єкта. Викликати звичайний метод не в об'єкті, а в класі не можна.

Завдання

1. Розробити в середовище розробки JAVA клас «Банківський рахунок» (BankAccount).
2. Забезпечити можливість додавання та зняття грошей з рахунку, перевірки поточного балансу, а також нарахування відсотків.
3. Використати статичні змінні для зберігання відсоткової ставки та підрахунку загального балансу за всіма рахунками.
4. Створити статичні методи для отримання загального балансу за всіма рахунками та отримання/зміни відсоткової ставки.
5. Розробити в середовище розробки JAVA клас для тестування з методом main.
6. В методі main реалізувати наступну логіку:
    a. Створити масив об'єктів класу BankAccount, ініціалізувати довільними значеннями.
    b. Вивести поточний загальний балансу банку (по всім рахункам).
    c. Зняти з довільного рахунку довільну суму грошей.
    d. Покласти на довільний рахунок довільну суму грошей.
    e. Вивести баланс по кожному рахунку.
    f. Вивести поточний загальний балансу банку (по всім рахункам).

g. Нарахувати відсотки на всі рахунки.
h. Вивести поточний загальний балансу банку після нарахування відсотків.

Перелік документів для включення до звіту ПР № 3

- постановка задачі;
- java код розробленого класу BankAccount з коментарями;
- java код класу для тестування з методом main;
- скріншот результату роботи java-програми.

Питання для самоконтролю

1. Що таке статична змінна Java?
2. Скільки екземплярів статичної змінної існує для класу?
3. Чи можна явно неініціалізувати статичні полі класу?
4. Що таке статичний метод Java?
5. Чи можливо всередині статичного методу мати доступ до нестатичних полів класу?
6. Чи можливо всередині статичного методу викликати інші нестатичні методи класу?
7. Статичний метод може посилатися на ключові слова this або super?
8. Чи потрібно створювати об'єкт класу для виклику статичного методу?

Практична робота № 4
СПАДКУВАННЯ ТА ПОЛІМОРФІЗМ.

Теоретичний матеріал

Спадкування – це механізм у програмуванні, який дозволяє описати новий клас на основі вже існуючого. Клас-спадкоємець при цьому отримує доступ до полів та методів батьківського класу.

Можна наслідувати (спадкувати) лише один клас.

Термінологія:

- ***підклас*** (дочірній) – клас, який успадковує інший клас;
- ***суперклас*** (батьківський) – клас, який успадковується.

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

Приклад наслідування класів наведено нижче.

```
class Figure {
       protected Point p;
       public Figure(float x, float y){
              p = new Point(x,y);
       }

    public void show(){
              System.out.print(p);
       }

}

class Rectangle extends Figure {
    private double w;
    private double h;

    Rectangle(double w, double h, float x, float y){
        super(x,y);
        this.w = w;
        this.h = h;
    }

    @Override
    public void show() {
      super.show();
         System.out.print( w +"*"+ h);
    }
}
```

Зверніть увагу, що у конструкторі класу-спадкоємця має бути спершу викликаний конструктор базового класу (super(параметри)), а потім ініціалізація полів класу-спадкоємця.

Якщо немає явного виклику конструктора базового класу, тоді буде викликаний конструктор за замовчуванням (він має існувати у базовому класі).

В середині похідного класу доступне ключове слова super – посилання на об’єкт базового класу.

Якщо в класі-спадкоємці перевизначено (Override) метод, і всередині потрібно викликати версію базового класу, використовують super.methodName(параметри).

Зміна доступу під час перевизначення (Override) методу

При перевизначені методів не можна робити більш обмежений доступ, ніж у методу базового класу.

Розширити доступ можна. Список допустимих модифікаторів доступу при перевизначені методів наведено в таблиці 4.1.

Таблиця 4.1. Зміна модифікаторів доступу при спадкуванні

Базовий клас	**Клас-спадкоємець**			
	public	protected	private	default
public	+	-	-	-
protected	+	+	-	-
private	-	-	-	-
default	+	+	-	+

Перетворення типів

up cast – приведення дочірнього типу до батьківського за ієрархією спадкування, перетворення виконується автоматично (неявно):

```
Figure f = new Rectangle(...);
```

down cast – приведення батьківського типу до дочірнього, перетворення виконується явно:

```
Rectangle r = (Rectangle)f;
```

ClassCastException

При *down cast* перетворенні, якщо об'єкт не є об'єктом класу, до якого перетворюється, тоді виникає виняток *ClassCastException.*

```
Figure f= new Circle(...);
Rectangle r = (Rectangle)f; --> ClassCastException
```

Оператор instanceof Java

Для перевірки того, чи створено об'єкт на основі якогось класу, Java існує спеціальний оператор — *instanceof*. Він повертає значення *true*, якщо перевірка показала істинність, або *false,* якщо результат був помилковим.

Послідовність ініціалізації під час створення об'єкта класу:

1. Базовий клас.
2. Поля та блоки ініціалізації у зазначеному порядку.
3. Інструкції конструктора похідного класу.

Завдання

1. Розробити в середовище розробки JAVA клас, похідний від класу Point, розробленого в практичній № 1.
2. Перевизначити методи базового класу Point.
3. Реалізувати власні методи похідного класу.
4. Розробити в середовище розробки JAVA клас для тестування з методом main.
5. В методі main реалізувати наступну логіку:
    a. Створити масив об'єктів базового класу.
    b. Ініціалізувати масив об’єктами базового класу та похідного.
    c. Протестувати поліморфне зв’язування.
    d. Протестувати down cast.

Перелік документів для включення до звіту ПР № 4

- постановка задачі;
- java код розробленого похідного класу з коментарями;
- java код класу для тестування з методом main;
- скріншот результату роботи java-програми.

Питання для самоконтролю

1. Які різновиди спадкування є в Java?
2. Де використовується ключове слово super?
3. Що таке перевизначення (override) методу?
4. Для успадкування класу public class Child extends Parent напишіть порядок ініціалізації об’єкта.

5. Які з перетворень типів виконуються автоматично – up cast або down cast?
6. Коли виникає виняток *ClassCastException*?
7. Як можна запобігти виникненню винятка *ClassCastException*?
8. Скільки базових класів може буди у одного класу?
9. Чи є в мові Java множинне спадкування?

Практична робота № 5
АБСТРАКТНІ КЛАСИ.

Теоретичний матеріал

Абстрактний клас в Java – використовується для оголошення спільної поведінки та надання можливості роботи з набором пов'язаних класів через загальний інтерфейс.

Абстрактний клас може містити як абстрактні методи, так і звичайні методи з реалізацією. Клас повинен бути обов'язково позначений як абстрактний, якщо він має хоча б один абстрактний метод.

Створити екземпляр абстрактного класу неможливо, при спробі компілятор повідомить про помилку.

Абстрактний метод в Java – оголошується всередині абстрактного класу і має тільки визначення, без реалізації (тіла методу). Абстрактний метод не може бути оголошений як **final, static** або **private**.

Приклад абстрактного класу:

```
public abstract class Figure {
protected Point p;
public Figure(float x, float y){
      p = new Point(x,y);
}

    public void show(){
       System.out.print(p);
}

    public abstract double area();
    public abstract double perimeter();


}
```

- в оголошенні абстрактного класу вказується ключове слово abstract;
- клас містить два абстрактних метода *area* та *perimeter*, в оголошенні яких використовується ключове слово *abstract*. Абстрактні методи *area* та perimeter не містять реалізацію (тіло);
- абстрактний клас може містити конструктор. В наведеному вище прикладі в конструкторі відбувається ініціалізація поля *p*;
- метод *show* є звичайним методом та містить реалізацію, котра за замовчуванням може бути використана в похідних класах.

Клас, котрий є похідним від абстрактного класу, має реалізувати всі його абстрактні методи, в іншому випадку він також повинен бути помічений як абстрактний.

Приклад спадкоємця абстрактного класу:

```
public class Trapezoid extends Figure{
```

```
        private double a;
        private double b;
        private double h;

        public Trapezoid(double a,  double b,  double h) {
                super(new Point ());
                this.a=a;
                this.b=b;
                this.h=h;
        }
        @Override
        public double area() {
                return h*((a+b)/2.0);
        }
        @Override
        public double perimeter() {
                return a + b + 2 * h;
        }
}
```

Конструктор похідного класу через *super* звертаються до конструктора базового абстрактного класу, куди передається значення для поля *p*.

Клас *Trapezoid* перевизначає (Override) абстрактні методи *area* та *perimeter* класу *Figure*, та завдає свою реалізацію для них.

Об'єкт абстрактного класу неможливо створити, але змінна типу *Figure* може посилатися на об'єкти похідних класів.

```
Figure tr = new Trapezoid(7,12,4);
```

Під час виклику *tr.show()*, для об'єкту буде задіяна реалізація за замовчуванням, з базового абстрактного класу *Figure*.

Під час виклику *tr.area()*, буде задіяна реалізація описана в класі *Trapezoid*.

Приклад спадкоємця абстрактного класу, який також є абстрактним:

```
public abstract class Tetragon extends Figure{
        protected Point p2;
        protected Point p3;
        protected Point p4;

        public Tetragon(Point p1, Point p2, Point p3, Point p4) {
                super(p1);
                this.p2=new Point (p2);
                this.p3=new Point (p3);
                this.p4=new Point (p4);
        }
}
```

Завдання

1. Розробити в середовище розробки JAVA абстрактний клас Figure з двома абстрактними методами – підрахунок площі фігури та периметру.

2. Успадкувати від нього класи Rectangle (прямокутник), Circle (коло) та Triangle (трикутник).
3. Реалізувати в кожному класі-спадкоємці абстрактні методи базового класу.
4. У класі Rectangle створити додатковий метод обчислення довжини діагоналі.
5. Розробити в середовище розробки JAVA клас для тестування з методом main.
6. В методі main реалізувати наступну логіку:
    a. Створити масив різних фігур.
    b. Обчислити та вивести значення периметра та площі кожної фігури залежно від її типу.
    c. Для фігур типу Rectangle також обчислити та вивести довжину діагоналі.

Перелік документів для включення до звіту ПР № 5

- постановка задачі;
- java код розроблених класів з коментарями;
- java код класу для тестування з методом main;
- скріншот результату роботи java-програми.

Питання для самоконтролю

1. Що таке абстрактний клас?
2. Що таке абстрактний метод?
3. Чи може бути абстрактний клас без жодного абстрактного методу?
4. Чи може бути статичний метод абстрактним?
5. Чи обов’язково клас-спадкоємець зобов’язаний реалізувати всі абстрактні методи батьківського класу?
6. Чи можна створювати об’єкти абстрактного класу?
7. Чи можна оголошувати посилання на абстрактний клас?
8. Чи може бути в абстрактному класі звичайні(не абстрактні) методи?
9. Чи може бути в абстрактному класі полі?
10. Чи може бути в абстрактному конструктори? Якщо так, до де їх можна викликати?

Практична робота № 6
ІНТЕРФЕЙСИ ТА ЇХ РЕАЛІЗАЦІЯ.

Теоретичний матеріал

Інтерфейс – опис типу в абстрактної формі – опис заголовків без реалізації.

Інтерфейс – це повністю абстрактний тип, який не містить жодних полів, крім констант.

Для оголошення інтерфейсу в Java використовується ключове слово ***interface***, за яким слідує ім’я інтерфейсу.

Інтерфейси також можуть містити:

- константи, які автоматично вважаються *static* та *final*;
- методи, які автоматично *public* та *abstract*;
- default методи – які автоматично public та мають реалізацію за замовчуванням;
- внутрішні (inner) інтерфейси – які автоматично public та static;
- внутрішні (inner) класи – які автоматично public та static.

Приклад оголошення інтерфейсу:

```
public interface Movable {

	void moveToPoint(Point p);

}
```

Інтерфейси також можуть успадковувати інші інтерфейси за допомогою ключового слова *extends*. Це дає змогу створювати ієрархію інтерфейсів і визначати спільну функціональність для кількох інтерфейсів.

Назва інтерфейсу часто відображає здатність (*ability*) об'єктів певних класів виконувати певні функції (контракт).

Реалізація інтерфейсу

В Java реалізація (імплементація) інтерфейсів – це коли клас реалізує всі методи, описані в інтерфейсі. Під час імплементації інтерфейсу клас зобов’язаний надати свою власну конкретну реалізацію для кожного абстрактного методу, зазначеного в інтерфейсі. Це дає змогу класу повністю слідувати тим вимогам, які встановлені інтерфейсом, і забезпечити свою реалізації інтерфейсу. Клас зобов'язаний забезпечити реалізацію всіх не default методів, визначених в інтерфейсах, інакше в його оголошення слід включити модифікатор abstract.

Для імплементації інтерфейсу в класі використовується ключове слово ***implements***, за яким вказується ім’я інтерфейсу (одного або декількох). На відміну від наслідування класів, де можна розширити тільки один клас, множинна реалізація інтерфейсів дозволена.

Приклад реалізації інтерфейсу:

```
public class Figure implements Movable{
	protected Point origin;
	public Figure(float x, float y){
		origin = new Point(x,y);
	}

	@Override
	public void moveToPoint(Point p) {
		this.origin.setX(p.getX());
		this.origin.setY(p.getY());

	}
}
```

Ім'я інтерфейсу можна використовувати як ім'я типу. Можна створювати інтерфейсні посилання та ініціалізувати їх будь-яким типом, який реалізує інтерфейс.

```
Movable m = new Figure(new Point(1,2));
m.moveToPoint(new Point(5,8));
```

Модель делегування дозволяє використовувати готову реалізацію інтерфейсу там, де це потрібно. Якщо з'явиться більш ефективніша реалізація інтерфейсу – легко перейти до неї не змінюючи код.

Види інтерфейсів

1. **Functional Interface**

   Має тільки один абстрактний метод.

   Може буди реалізований за допомогою Лямбда виразів.

   Приклади Functional інтерфейсів:

   - Runnable – метод run();
   - ActionListener– метод actionPerformed();
   - ItemListener – метод itemStateChanged().

2. **Marker interface**

Інтерфейс, який не містить жодних методів, полів, абстрактних методів і будь-яких констант.

Marker інтерфейси позначають певну властивість або загальну ознаку належності класів певній групі.

Приклад Marker інтерфейсів

- Clonable
- Serializable
- Remote
- EventListener

Завдання

1. Розробити в середовище розробки JAVA інтерфейс Iterator з двома методами *boolean hasNext()* та *T next()* по масиву будь якого типу T.
2. Можна тип вказати явно, наприклад int, а можна створити шаблонний інтерфейс.
3. Для класу n-вимірного вектора (NVector) із практичної роботи № 2 реалізувати інтерфейс Iterator різними способами, а саме:
    a. з допомогою public внутрішніх класів;
    b. з допомогою private внутрішніх класів;
    c. з допомогою anonymous внутрішніх класів;
    d. з допомогою локальних внутрішніх класів.
4. Розробити в середовище розробки JAVA клас для тестування з методом main.
5. В методі main реалізувати наступну логіку:
    a. Створити об'єктів класу NVector, ініціалізувати довільними значеннями.
    b. Вивести поточний вміст вектора за допомогою Iterator, реалізованого public внутрішнім класом.
    c. Вивести поточний вміст вектора за допомогою Iterator, реалізованого private внутрішнім класом.
    d. Вивести поточний вміст вектора за допомогою Iterator, реалізованого anonymous внутрішнім класом.
    e. Вивести поточний вміст вектора за допомогою Iterator, реалізованого локальним внутрішнім класом.
6. Порівняти реалізації інтерфейсу та вибрати найбільш підходящий спосіб. Вибір обґрунтувати.

Перелік документів для включення до звіту ПР № 6

- постановка задачі;

- java код розроблених класів з коментарями;
- java код класу для тестування з методом main;
- скріншот результату роботи java-програми.

Питання для самоконтролю

1. Які дані може містити інтерфейс?
2. Які мають специфікатори (модифікатори) за замовчуванням всі полі, оголошені в інтерфейсі?
3. Які мають специфікатори (модифікатори) за замовчуванням всі методи, оголошені в інтерфейсі?
4. У чому різниця між абстрактним класом та інтерфейсом в Java?
5. Скільки інтерфейсів може реалізовувати (implements) один клас?
6. Що таке анонімні класи? Коли використовується?
7. Що таке Nested class? Коли використовується?
8. Які модифікатори доступу можуть бути у Nested класі?
9. Що таке default методи в інтерфейсі?
10. Чи обов’язково реалізовувати default методи в класі, що реалізує інтерфейс?

Практична робота № 7
РЕФЛЕКСІЯ.

Теоретичний матеріал

Рефлексія – це механізм, що дозволяє досліджувати будову типу (наприклад, класу) за його двійковим уявленням.

В основі рефлексії лежить тип Class<T>.

Рефлексія дозволяє отримати наступну інформацію:

- визначити клас об'єкта;
- отримати інформацію про модифікаторів класу, поля, методи, константи, конструктори та суперкласи;
- з'ясувати, які методи належать реалізованому інтерфейсу;
- створити екземпляр класу, причому ім'я класу може бути невідоме до виконання програми;
- отримати та встановити значення поля об'єкта за його іменем;
- викликати метод об'єкта.

Отримання об'єкта Class

- Під час виконання у об'єкта методом getClass()

```
A a = new A();
Class<A> cl = a.getClass();
```

- Під час компіляції, використовуючи суфікс class

```
Class<String> cl = String.class;
Class intCl=int.class;
```

- Через класи-оболонки примітивних типів

```
Class cl= Integer.TYPE;
```

- За допомогою статичного методу forName

```
Class<String> cl =Class.forName("java.lang.String");
```

Методи об'єкту Class

1. **Загальна інформація про класс**
    - *String getName()* – повне ім'я класу (java.lang.Object, int);
    - *String getSimpleName()* – просте ім'я класу (Object, int);
    - *Class< ? super T >getSuperClass()* – суперклас;
    - *Class<?>[] getInterfaces()* – реалізовані інтерфейси;

- *int getModifiers()* – модифікатори класу.

2. **Інформація про полі класу**

- *Field[] getFields()* – всі відкриті полі класу;
- *Field getField(name)* – відкрите поле класу за іменем;
- *Field[] getDeclaredFields()* – всі полі класу (не тільки відкриті);
- *Field getDeclaredField(name)* – поле класу за іменем;
- *NoSuchFieldException* – якщо поле за іменем не знайдено.

3. **Інформація про методи класу**

- *Method[] getMethods()* – всі відкриті методи класу;
- *Method getMethod(name, Class… parameters)* – відкритий метод класу за іменем та списком параметрів;
- *getDeclaredMethods()* – всі методи (не тільки відкриті);
- *getDeclaredMethod(name , Class… parameters)* – метод класу за іменем та списком параметрів;
- *NoSuchMethodException* – якщо метод не знайдено.

4. **Інформація про конструктори класу**

- *Constructor<?>[] getConstructors()* – всі відкриті конструктори класу;
- *Constructor<T> getConstructor(Class… parameters)* – відкритий конструктор за списком параметрів;
- *getDeclaredConstructors()* – всі конструктори класу (не тільки відкриті);
- *getDeclaredConstructor(Class… parameters)* – конструктор за списком параметрів;
- *NoSuchMethodException* – якщо конструктор не знайдено.

5. **Створення об’єктів класу**

- *T newInstance(Object ... args)* – створити новий об'єкт за допомогою відповідного конструктора (спочатку треба отримати конструктор, а потім у нього викликати метод *newInstance*);

- *class.newInstance()* – створити новий об'єкт, використовуючи конструктор за замовчуванням.

6. Виклик методів класу

- *public Object invoke(Object obj, Object...args) throws IllegalAccessException, IllegalArgumentException, InvocationTargetException*

- не статичний метод – вибір реалізації визначається на підставі фактичного типу obj;

- статичний метод - obj може бути null.

Модифікатори. Клас Modifiers

Таблиця 7.1. Модифікатори та відповідні константи та методи

Модифікатор	**Константа**	**Метод**
abstract	ABSTRACT	isAbstract
final	FINAL	isFinal
interface	INTERFACE	isInterface
native	NATIVE	isNative
private	PRIVATE	isPrivate
protected	PROTECTED	isProtected
public	PUBLIC	isPublic
static	STATIC	isStatic
strictfp	STRICT	isStrict
synchronized	SYNCHRONIZED	isSynchronized
transient	TRANSIENT	isTransient
volatile	VOLATILE	isVolatile

Масиви як типи

Ім'я типу масиву – [ім'я_типу_елемента

Методи:

boolean isArray() – чи є тип Class масивом.

Class<?> getComponentType() – тип елемента масиву.

Імена класів типів у масиві кодуються спеціальним чином, наведеним в таблиці 7.2.

Таблиця 7.2. Тип елементу масиву та відповідний йому код

Тип	**Код**
class або interface	Lclass;
boolean	Z
byte	B
char	C
double	D
float	F
int	I
long	J
short	S

Приклади:

Якщо массив Object [], то закодований тип елементів масиву буде [Ljava.lang.Object.

Якщо масив типу float[][], то закодований тип елементів буде [[F.

Анотація та Рефлексія

Java Reflection розпізнає *Annotation*, час життя якої під час виконання, тобто в яка має в описі

@Retention(RetentionPolicy.RUNTIME)

З об'єкта типу *Class*, *Method*, *Field* або *Constructor* можна отримати конкретні анотації, пов'язані з цим об'єктом, викликавши метод *getAnnotation()*.

```
Method m = c.getMethod("myMeth");
MyAnno anno = m.getAnnotation(MyAnno.class);
System.out.println(anno.str() + " " + anno.val());
```

Завдання

1. Розробити в середовище розробки JAVA клас для тестування методів рефлексії на типі String з методом main.
2. В методі main реалізувати наступну логіку:
    a. Отримати об'єкт типу Class трьома різними способами.
    b. Вивести усі модифікатори класу String.
    c. Вивести інформацію про всі public конструктори класу String.

d. Вивести інформацію про всі полі класу String.
e. Вивести інформацію про всі public методи класу String.
f. Викликати будь-який із не статичних методів з параметрами та роздрукувати результат.
g. Викликати будь-який із статичних методів з параметрами та роздрукувати результат.

Перелік документів для включення до звіту ПР № 7

- постановка задачі;
- java код класу для тестування з методом main;
- скріншот результату роботи java-програми.

Питання для самоконтролю

1. Що таке рефлексія в Java?
2. Які пакети містить Reflection API?
3. Де використовується рефлексія?
4. Яку інформацію про клас можна отримати за допомогою рефлексії?
5. Яку інформацію про метод можна отримати за допомогою рефлексії?
6. Яку інформацію про поле можна отримати за допомогою рефлексії?
7. За якої умови можна отримати інформацію про Анотацію засобами рефлексії?

Практична робота № 8
КОЛЕКЦІЇ.

Теоретичний матеріал

Collections Framework – це набір стандартних контейнерів (колекцій) та правил їх використання, знаходиться в пакеті java.util та містить в собі наступні елементи:

- Інтерфейси
- Реалізації
- Алгоритми

Колекція — невпорядкований набір елементів. В основі *Collections Framework* лежить інтерфейс *Collection*.

Інтерфейс Collection в Java

Інтерфейс Collection декларує основні методи, які доступні всім колекціям, та є найвищим інтерфейсом в ієрархії колекцій. Інтерфейс Collection реалізований інтерфейсами Set, List та Queue, та включає методи для роботи з колекціями: операції додавання, видалення, обходу, перевірку розміру та інше.

Структура Collections Framework

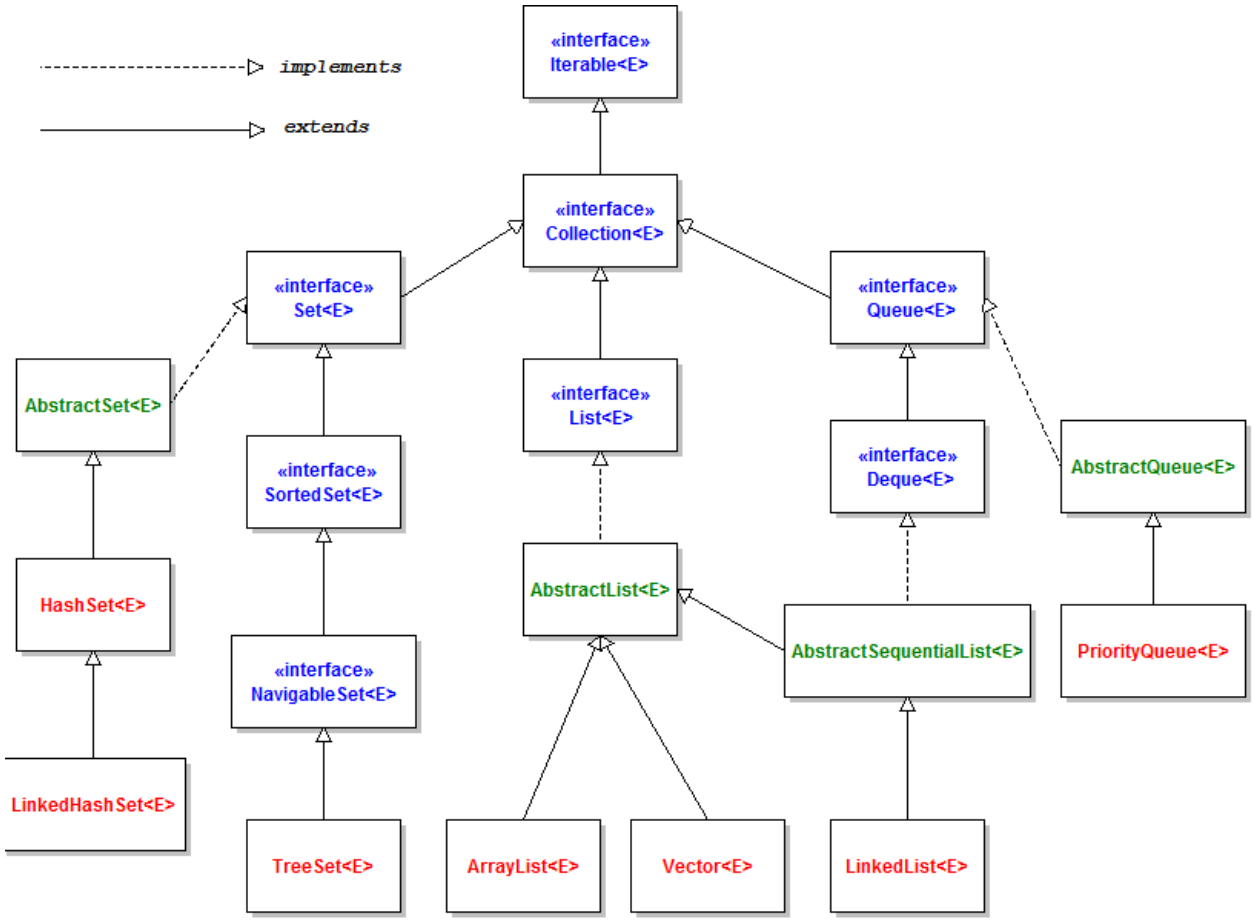


Рис. 8.1. Структура Collections Framework

Інтерфейс Collection напряму не реалізує жоден клас. Реалізуються більш специфічні інтерфейси, такі як *List, Set* та *Queue.*

List являє собою впорядкований список, де елементи можуть повторюватися. Приклади класів-реалізацій інтерфейсу *List* включають *ArrayList, LinkedList* та *Vector*.

Set – це колекція унікальних елементів, де кожен елемент може бути доданий тільки один раз. Приклади класів-реалізацій інтерфейсу *Set* включають *HashSet, TreeSet* і *LinkedHashSet*.

Queue – колекція, яка працює за принципом "першим прийшов – першим вийшов" (FIFO – First-In-First-Out). Приклади класів-реалізацій інтерфейсу *Queue* включають *LinkedList* та *PriorityQueue*.

Робота з колекціями: основні операції. Методи Collection

1. Визначення розміру

   int size() – кількість елементів

   boolean isEmpty() – перевірка на пустоту

2. Перевірки на належність елементів

   boolean contains(Object o) – одного элемента

   boolean containsAll(Collection<?> c) – всіх елементів колекції *c*

3. Додавання елемента

   boolean add(E e) – одного элемента

   boolean addAll(Collection<? Extends E> c) – всіх елементів колекції *c*

4. Видалення елементів

   boolean remove(Object e) – одного елемента

   boolean removeAll(Collection<?> c) – видалення елементів колекції *c*

   boolean retainAll(Collection<?> c) – видалення елементів, які не містяться в іншій колекції *c*

   void clear() – видалення всіх елементів

5. Винятки

   UnsupportedOperationException

6. Ітерація по колекції.

   За допомогою for-each (інтерфейс *Iterable*)

   За допомогою ітератора (інтерфейс *Iterator*).

Методи ітераторів

boolean hasNext() – визначення наявності наступного елемента

E next() – отримання наступного елемента

void remove() – видалення элемента

Винятки:

NoSuchElementException – при досягненні кінця колекції.

ConcurrentModificationException – при зміні колекції паралельними потоками.

Приклад обходу колекції за допомогою ітератора**:**

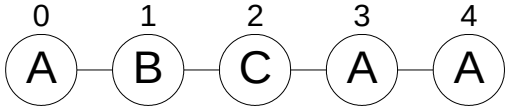
```
Iterator<E> i = list.iterator();
while(i.hasNext()) {
    E element = i.next();
    System.out.ptintln(element);
}
```

Приклад обходу колекції за допомогою for-each**:**

```
for(E element: list) {
    System.out.ptintln(element);
}
```

Списки. Інтерфейс List

Список – це колекція, де кожному елементу поставлено у відповідність індекс (починається з 0).



В основі лежить інтерфейс *List<E>*. Основні реалізації – ArrayList, LinkedList,Vector.

Методи List<E>

1. Доступ за індексом

   E get(int index) – отримати елемент по індексу

   E set(int index, E el) – заміщення елемента, повертає посилання на попереднє значення

   void add(int index, E el) – додавання нового елементу

   E remove(int index) – видалення елементу

2. Пошук елементів

   int indexOf(Object e) – пошук з початку індексу елемента

   int lastIndexOf(Object e) – пошук з кінця індексу елемента

3. Отримання частини списку

   List<E> subList(int from, int to)

Реалізація List. Клас ArrayList

ArrayList – реалізація інтерфейсу *List* на базі масиву. Характеризується таким поняттям як місткість (Capacity).

1. Плюси

   Швидкий доступ за індексом O(const)

   Швидка вставка та видалення елементів з кінця

2. Мінуси

   Повільна вставка та видалення елементів у середину та початок O(N)

   Реалізація не синхронізована

3. Конструктори ArrayList

   ArrayList() – порожній список

   ArrayList(Collection<E> c) – копія колекції

   ArrayList(int initialCapacity) – порожній список заданої місткості

4. Застосування ArrayList

   "Нескінченний" масив

   Стек

5. Приклад ArrayList

```
List<Integer> list = new ArrayList<>();
for (int i = 0; i < 10; i++) {
    list.add(i+1);
}

for (int i = 0; i < list.size(); i++) {
    System.out.println(list.get(i));

}
```

Реалізація List. Клас LinkedList

LinkedList – реалізація інтерфейсу *List* у вигляді двозв'язного списку.

1. Плюси

   Швидке додавання та видалення елементів O(const)

2. Мінуси

   Повільний доступ за індексом O(N)

3. Конструктори

   LinkedList() – порожній список

   LinkedList(Collection<? extends E> c) – копія колекції

4. Методи

   void addFirst(E o) – додати на початок списку

   void addLast(E o) – додати в кінець списку

   E removeFirst() – видалити перший елемент

   E removeLast() – видалити останній елемент

   E getFirst() – отримати перший елемент

   E getLast() – отримати останній елемент

5. Застосування LinkedList

   Стек

   Черга

   Дек

Множини. Інтерфейс Set

Множина – це колекція елементів, що не повторюються. Значення null може бути лише одно.

В основі лежить інтерфейс *Set<E>*. Основні реалізації – HashSet<E>, TreeSet<E>.

Операції над множинами

addAll(Collection c) – об'єднання множин

retainAll(Collection c) – перетин множин

containsAll(Collection c) – перевірка входження

removeAll(Collection c) – різниця множин

Реалізації HashSet та TreeSet

HashSet в Java це клас, який реалізує інтерфейс Set. Він використовує хеш-таблицю для зберігання елементів, яка дозволяє виконувати операції додавання, видалення та перевірки наявності елемента за сталу часову складність – O(1).

HashSet не зберігає елементи в впорядкованому порядку і не дозволяє дублікатів. Це означає, що кожен елемент в *HashSet* є унікальним.

TreeSet це реалізація *Set* інтерфейсу, яка зберігає унікальні елементи в відсортованому порядку. *TreeSet* сортує елементи відповідно до їх природного порядку або використовує компаратор для визначення порядку. *TreeSet* використовується, коли ви хочете зберігати унікальні елементи в відсортованому вигляді.

Порівняння елементів

Інтерфейс *Comparable<E>* з методом *int compareTo(E o)* задає природній порядок елементів (може бути тільки один).

Інтерфейс *Comparator<E>* з методом *int compare(E o1, E o2)* – порівняння елементів способом, відмінним від природного.

Для порівняння *o1* та *o2* використовують наступне правило:

Якщо *o1 < o2*, то метод повинен повертати від'ємне число.

Якщо *o1 = o2*, то метод повинен повертати 0.

Якщо *o1 > o2*, то метод повинен повертати додатне число.

<	=	>
-	0	+

Клас Collections

1. Алгоритми для роботи з колекціями

   - Прості операції

     fill(List<? super E > l, E v) – заповнення списку вказаним значенням

     reverse(List<?> l) – зміна порядку елементів в списку

     copy(List<? super E> l1, List l2<? Extends E>) – копіювання зі списку до списку

   - Перемішування

shuffle(List<?> l)

shuffle(List<?> l, Random r)

- Сортування

  sort(List<T> l) – сортування списку (природний порядок)

  sort(List<T> l, Comparator<? super T> c) – сортування списку (зазначений порядок)

- Двійковий пошук

  int binarySearch(List list, T key) – шукає key у списку (природний порядок)

  binarySearch(List list, T key, Comparator c) – шукає key у списку, порядок задан Comparator

- Пошук мінімуму та максимуму

  min(Collection c) – мінімальний елемент (природний порядок)

  min(Collection c, Comparator cmp) – мінімальний елемент (зазначений порядок)

  max(Collection c) – максимальний елемент (природний порядок)

  max(Collection c, Comparator cmp) – максимальний елемент (зазначений порядок)

2. Спеціальні колекції

   emptySet() – порожня множина

   emptyList() – пустий список

   singleton(T o) – множина з одного елементу

   singletonList(T o) – список з одного елементу

3. Оболонки колекцій

   unmodifiableSet(Set s) – незмінна множина

   unmodifiableList(List l) – незмінний список

Завдання

1. Протестувати колекції ArrayList та TreeSet на задачі зберігання об'єктів класу Point із практичної роботи № 1.

2. Розробити в середовище розробки JAVA клас для тестування з методом main.
3. В методі main для кожної колекції виконати наступне:
    a. Заповнити колекцію випадковими даними (забезпечити наявність однакових елементів).
    b. Вивести вміст колекції за допомогою foreach.
    c. Ввести з клавіатури значення точки, яка присутня в колекції в декількох екземплярах.
    d. Обійти колекцію за допомогою ітератора та при обході видалити всі елементи, які дорівнюють точці, введеній з клавіатури відповідно до попереднього пункту.
    e. Протестувати методи колекції основні методи колекції.
    f. Протестувати алгоритми сортування та пошуку класу Collections.

Перелік документів для включення до звіту ПР № 8

- постановка задачі;
- java код розроблених класів з коментарями;
- java код класу для тестування з методом main;
- скріншот результату роботи java-програми.

Питання для самоконтролю

1. Яка ієрархія колекцій у Java Collection Framework?
2. Яка внутрішня будова ArrayList?
3. Яка внутрішня будова LinkedList?
4. Яка внутрішня будова HashSet?
5. Яка внутрішня будова TreeSet?
6. В чому різниця між інтерфейсами Comparable та Comparator?
7. Які впорядковані колекції ви знаєте?
8. Як задати природній порядок для впорядкованої колекції?
9. Чим відрізняється ArrayList від LinkedList?
10. Для чого потрібні ітератори?
11. Як відсортувати колекцію елементів?

Практична робота № 9
ПОТОКИ ВВЕДЕННЯ-ВИВЕДЕННЯ. СЕРІАЛІЗАЦІЯ.

Теоретичний матеріал

Потік (Stream) – це потік даних, який не залежить від конкретних пристроїв введення/виведення.

В мові Java існує 2 види потоків введення/виведення – це байтові потоки та символьні.

Байтові потоки – послідовність байт (byte).

Символьні – послідовність двобайтових символів Unicode (char).

Класи потоків знаходяться в пакеті java.io. Основні типи винятків при роботі з потоками – IOException, FileNotFoundException.

Основні класи потоків введення/виведення наведено в таблиці 1.

Таблиця 1. Класи потоків введення/виведення

	Байти (byte)	**Символи (char)**
Читання даних	InputStream	Reader
Запис даних	OutputStream	Writer

Розглянемо кожен клас окремо.

InputStream

`public abstract int read() throws IOException` – зчитує один байт та повертає його у вигляді числа 0÷255. Якщо немає інформації повертає «-1».

`public int read(byte[] b, int off, int len) throws IOException` – зчитує b[off]÷b[off+len-1] байт якщо раніше не буде досягнуто кінця файлу. Повертає «-1», якщо не було зчитано жодного байту, тобто досягнуто кінця потоку.

`public int read(byte[] b) throws IOException` еквівалентний виклику методу `read(b,0,b.lenght).`

`public long skip(long n) throws IOException` – пропускає n байт або всі до кінця потоку, якщо їх менше ніж n. Повертає кількість фактично пропущених байтів.

`public int available() throws IOException` – повертає кількість байт, які можуть бути введені чи пропущені. За замовчуванням 0.

`public void close() throws IOException` – закриває потік. Закритий потік не може виконувати операції виведення і не може бути повторно відкритий.

OutputStream

`public abstract void write(int b) throws IOException` – виводить у вихідний потік значення у вигляді 1 байта (молодший байт враховується, інші ігноруються)

`public void write(byte[] b, int off, int len) throws IOException` – виводить масив байт b[off]÷b[off+len-1] у вихідний потік.

`public void write(byte[] b) throws IOException` еквівалентний виклику методу write(b,0,b.lenght)

`public void flush() throws IOException` – очищає вихідний потік та примусово записує будь-які буферизовані вихідні байти.

`public void close() throws IOException` – закриває вихідний потік та звільняє всі системні ресурси, пов'язані з цим потоком.

Reader

`public int read() throws IOException` – зчитує символ і повертає його у вигляді цілого числа з діапазону 0÷65535 (2 молодших байат int) або «-1».

`public int read(char[] cbuf) throws IOException` – зчитує масив символів та повертає кількість прочитаних символів або «-1».

`public abstract int read(char[] cbuf, int off, int len) throws IOException` – зчитує масив символів та зберігає його в cbuf[off]÷cbuf[off+len-1] та повертає кількість фактично прочитаних символів або «-1».

`public abstract void close() throws IOException` – закриває вхідний потік та звільняє всі системні ресурси, пов'язані з цим потоком.

Writer

`public void write(int c) throws IOException` – записує символ у потік у вигляді цілого числа з діапазону 0÷65535 (2 молодших байта int).

`public void write (char[] cbuf) throws IOException` – записує масив символів.

`public abstract void write(char[] cbuf, int off, int len) throws IOException` – записує масив символів з cbuf[off]÷cbuf[off+len-1.

`public abstract void flush() throws IOException` – очищує вихідний потік та примусово записує будь-які буферизовані вихідні символи.

`public abstract void close() throws IOException` – закриває вихідний потік та звільняє всі системні ресурси, пов'язані з цим потоком.

Серіалізація

Серіалізація – це процес збереження стану об'єкта у послідовність байт. ***Десеріалізація*** – це процес відновлення об'єкта з потоку байт.

Для збереження стану об'єкта у послідовність байт, клас об'єкта має реалізувати інтерфейс-маркер *java.io.Serializable.*

Автоматична серіалізація

Процес запису об'єкта (серіалізація):

- Записуються метадані про клас асоційований з об'єктом.
- Записується предок.
- Записуються значення всіх *non-static* полів, що не мають модифікатора *transient*.

Процес читання (десеріалізація):

- Виділяється пам'ять під об'єкт.
- Зчитується предок.
- Зчитуються значення всіх полів, які не мають модифікатора *transient* (вони набувають значення за замовчуванням).

Щоб клас можна було серіалізувати в одній JVM, а десеріалізувати в іншій необхідно, щоб обидві JVM були завантажені класи з однаковим описом і *serialVersionUID*.

```
private static final long serialVersionUID=…;
```

Якщо не вказати явно *serialVersionUID*, то механізм серіалізації зробить це сам і опис класу буде чутливим до будь-яких змін у класі, аж до реалізації компілятора.

Серіалізація вручну

На механізм серіалізації можно вплинути. Для цього або визначити у класі, що реалізує інтерфейс *java.io.Serializable* закриті методи:

```
private void writeObject(ObjectOutputStream out) throws IOException

private void readObject(ObjectInputStream in) throws IOException,
ClassNotFoundException
```

або реалізувати інтерфейс *Externalizable:*

```
public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException
public void writeExternal(ObjectOutput out) throws IOException
```

Стандартна серіалізація в цих методах доступна через виклик методів `in.deafaultReadObject()` та `out.defaultWriteObject()`.

Завдання

1. Розробити в середовище розробки JAVA програму, яка веде облік читачів та книг у бібліотеці. Для цього слід створити набір класів, що описують бібліотеку (Автор, Книга, Бібліотека, Читач, Прокат, ...).
2. Клас Книга може характеризуватись назвою (поле типу String), списком авторів (список-масив з елементів типу Автор), роком видання та номером видання (полі типу int).
3. Клас Читач, який представляє читача (крім імені та прізвища читач характеризується його «реєстраційним номером» та списком отриманих книг (список-масив об'єктів Книга)).
4. Клас Бібліотека, який представляє бібліотеку; характеризується назвою бібліотеки, списком книг, що зберігаються, і списком зареєстрованих читачів (списки-масиви, що зберігають об'єкти відповідних типів).
5. Крім того, створити клас LibraryDriver зі статичними методами, що виконують серіалізацію та десеріалізацію бібліотеки та методом main().
6. У кожному класі має бути конструктор за замовчуванням, гетери та сеттери, перевизначений метод toString(), а також методи, що реалізують базову функціональність.
7. За допомогою механізму серіалізації збережіть у файлі стан системи та відновіть систему з цього ж файлу.
8. Реалізуйте три версії програми, що реалізують різні стратегії серіалізації:
    a. всі класи, що входять до системи, є серіалізованим (реалізують інтерфейс java.io.Serializable);
    b. серіалізуються (реалізують інтерфейс java.io.Serializable) не всі класи системи: класи Автор, Читач, Книга є НЕ серіалізуються;

c. класи, які входять у систему, реалізують інтерфейс java.io.Externalizable.

9. В методі main для кожної версії виконати наступне:
    a. створити «бібліотеку» з книгами та списком читачів;
    b. видати кілька книг;
    c. вивести інформацію про поточний стан бібліотеки;
    d. провести серіалізація бібліотеки у файл;
    e. провести десеріалізація бібліотеки з файлу;
    f. вивести стан десеріалізованої системи;
    g. порівняти стан системи до та після серіалізація/ десеріалізація.

Перелік документів для включення до звіту ПР № 9

- постановка задачі;
- java код розроблених класів з коментарями;
- java код класу для тестування з методом main;
- скріншот результату роботи java-програми.

Питання для самоконтролю

1. Які види потоків існують в Java?
2. Які основні класи потоків в Java?
3. Які класи існують для введення/виведення із файлів?
4. У чому різниця між інтерфейсом Serializable та Externalizable у Java?
5. Скільки методів у Serializable?
6. Що таке serialVersionUID? Що станеться, якщо ви його не визначите?
7. При серіалізації ви хочете, щоб деякі члени не серіалізувалися? Як цього досягти?
8. Що станеться, якщо один із членів класу не реалізує інтерфейс Serializable?
9. Якщо клас серіалізується, а його суперклас - ні, яким буде стан змінних екземпляра, успадкованих від суперкласу після десеріалізації?
10. Чи можете ви налаштувати процес серіалізації або перевизначити процес серіалізації за замовчуванням Java?
11. Чи можемо передати серіалізований об'єкт через мережу?
12. Які типи змінних не серіалізуються під час серіалізації Java?

Індивідуальна робота № 1
ПРОЕКТУВАННЯ СЕРЕДНЬОГО ШАРУ ІНФОРМАЦІЙНОЇ СИСТЕМИ.

Це завдання передбачає виявлення та аналіз функцій інформаційної системи у певній предметній галузі, побудову її концептуальної моделі у вигляді UML діаграми класів.

Завдання

1. Обрати будь-яку предметну область для моделювання інформаційної системи.
2. Визначити мету розробки інформаційної системи.
3. Визначити список користувачів інформаційної системи.
4. Визначити основну функціональність інформаційної системи.
5. Визначити дані, з якими буде працювати інформаційна система.
6. Розробити набір класів інформаційної системи.
7. Забезпечити наявність класів, пов'язаних ієрархією спадкування.
8. Розробити діаграму класів предметної області та формат файлів для зберігання даних.

Перелік документів для включення до звіту ІР № 1

- титульний лист;
- постановка задачі;
- функціональність інформаційної системи;
- UML діаграма розроблених класів;
- опис формату файлів для зберігання даних;
- java код розроблених класів з коментарями;
- java код класу для тестування з методом main;
- скріншот результату роботи java-програми.

Індивідуальна робота № 2
РОЗРОБКА СЕРЕДНЬОГО ШАРУ ІНФОРМАЦІЙНОЇ СИСТЕМИ.

Це завдання передбачає розробку програми з консольним інтерфейсом, що реалізує модель інформаційної системи, побудованої в індивідуальній роботі №1.

Завдання

1. Забезпечити реалізацію основної функціональності інформаційної системи, визначеної в індивідуальній роботі № 1.
2. Використати колекції для зберігання об’єктів.
3. Обґрунтувати вибір колекції.
4. Забезпечити збереження даних (стану) інформаційної системи у файлі.
5. Забезпечити відновлення стану інформаційної системи з файлу.
6. Розробити класи для тестування та демонстрації основної функціональності.

Перелік документів для включення до звіту ІР № 2

- титульний лист;
- постановка задачі;
- java код розроблених класів з коментарями;
- заповнені тестовими даними файли для збереження даних;
- java код класу для тестування з методом main;
- скріншот результату роботи java-програми.

Індивідуальна робота № 3
ПОТОКИ ОБЧИСЛЕНЬ.

Це завдання передбачає розробку програми, що демонструє взаємодію між потоками обчислень.

Завдання

1. Розробити програму для моделювання роботи готелю у реальному (прискореному) часі.
2. Розробити клас "Hotel", який характеризується місткістю (скільки всього постояльців може прийняти) і пам'ятає список людей, які проживають у ньому на даний момент.
3. Розробити клас "Booking", який характеризується прізвищем постояльця та кількістю мілісекунд проживання.
4. У класі "Hotel" реалізувати наступні операції:
    a. поселення "заявки (booking)" – метод або успішно поселяє і повертає true, або повертає false у разі нестачі вільних місць;
    b. виселення заданої "заявки" – метод явно викликається самою заявкою в потрібний момент часу – готель не повинен самостійно відраховувати мілісекунди за кожного постояльця, тому що потенційно може продовжити своє проживання;
    c. пошуку постояльця за прізвищем.
5. "Заявка(Booking)" повинна самостійно поселяється в готель, створюючи при цьому окремий потік (це може відбуватися як автоматично в конструкторі, так і у спеціальному методі).
6. У разі неможливості оселитися, заявка переходить у режим очікування появи вільних місць.
7. Після успішного поселення заявка відраховує (sleep) задану кількість мілісекунд – і виселяється, звільняючи місце у готелі.
8. Функція main намагається "поселити" до готелю кілька заявок. Але не одночасно, а через деякі проміжки часу – щоб змоделювати ситуацію неодночасного їхнього приходу.
9. Клас "Hotel" виводить на екран діагностичні повідомлення, що повідомляють про операції поселення (успішного та безуспішного) та

виселення – із зазначенням часу, інформації про заявку та кількість місць, що залишилися.

10. Потрібно наочно продемонструвати безуспішне поселення, очікування та подальше успішне поселення до готелю після виселення.

Перелік документів для включення до звіту ІР № 3

- титульний лист;
- постановка задачі;
- java код розроблених класів з коментарями;
- java код класу для тестування з методом main;
- скріншот результату роботи java-програми.

Рекомендована література

Основна література

1. Herbert Schildt. Java: A Beginner's Guide. McGraw-Hill Education; 8th edition. 2018
2. Cay S. Horstmann. Core Java Volume I – Fundamentals. Prentice Hall; 11th edition. 2018
3. James Gosling. Java Language Specification, Java SE 8 Edition. Addison-Wesley Professional; 1st edition. 2014
4. Bruce Eckel. Thinking in Java 4th Edition. Pearson. 2006
5. https://javarush.com/
6. https://docs.oracle.com/en/java/

Допоміжна література

1. Brian Goetz with Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, and Doug Lea. Java Concurrency in Practice. Addison-Wesley Professional. 2006
2. Scott Oaks. Java Performance : The Definitive Guide . O'Reilly and Associates; 1st edition. 2014
3. Eric Freeman, Elisabeth Robson. Head First Design Patterns (A Brain Friendly Guide). O'Reilly and Associates; 1st edition. 2014

ДОДАТКИ

Додаток №1. Титульний лист індивідуальної роботи

Міністерство освіти і науки України

Харківський національний університет імені В.Н.Каразіна

Кафедра теоретичної та прикладної інформатики

Звіт по дисципліні
Об'єктно-орієнтоване програмування (мова JAVA)
Індивідуальне завдання № __

Студента:_____

Групи:_____

Необхідний термін здачі завдання:_____

Фактичний термін здачі завдання:_____

Кількість балів:_____

Харків рік

Електронне навчальне видання комбінованого використання
Можна використовувати в локальному та мережному режимі

Морозова Анастасія Геннадіївна
Владимирова Марина Володимирівна
Зарецька Ірина Тимофіївна
Бєлова Лілія Петрівна

ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ (МОВА JAVA)

Методичні рекомендації
до виконання практичних та індивідуальних (розрахунково-графічних) робіт для здобувачів вищої освіти факультету математики і інформатики першого (бакалаврського) рівня вищої освіти

В авторській редакції

Підписано до розміщення 28.03.2025. Гарнітура Times New Roman.
Ум. друк. арк. 2,96. Обсяг 1,359 Кб. Зам. № 124/25.

Харківський національний університет імені В. Н. Каразіна,
61022, м. Харків, майдан Свободи, 4.
Свідоцтво суб'єкта видавничої справи ДК № 3367 від 13.01.2009
Видавництво ХНУ імені В. Н. Каразіна