**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**Інститут комп’ютерних технологій, автоматики та метрології**

**кафедра “Електронних обчислювальних машин”**

****

Звіт

з лабораторної роботи №6

дисципліни «Кросплатформні засоби програмування»

на тему: **«**Параметризоване програмування**»**

Варіант 15

**Виконала:**

студентка групи КІ-303

Сухарник А.А.

**Прийняв:**

Іванов Ю.С.

Львів – 2024

ЛАБОРАТОРНА РОБОТА №6

**ПАРАМЕТРИЗОВАНЕ ПРОГРАМУВАННЯ**

**Мета роботи**: оволодіти навиками параметризованого програмування мовою Java.

**Теоретичний матеріал**

Параметризоване програмування є аналогом шаблонів у С++. Воно полягає у написанні коду, що можна багаторазово застосовувати з об’єктами різних класів. Користувачів параметризованого програмування можна поділити на 3 рівні кваліфікації: 1. ті, що користуються готовими класами; 2. ті, що користуються готовими класами і вміють виправляти помилки, що виникають при їх використанні; 3. ті, що пишуть власні параметризовані класи. Для успішного застосування параметризованого програмування слід навчитися розуміти помилки, що генерує середовище при компіляції програми, що можуть стосуватися, наприклад, неоднозначності визначення спільного суперкласу для всіх переданих об’єктів. З іншої сторони необхідно передбачити захист від присвоєння об’єктів параметризованого класу, що містять об’єкти підкласу об’єктам параметризованого класу, що містять об’єкти суперкласу і дозволити зворотні присвоєння. Для вирішення цієї проблеми у мові Java введено так звані підстановочні типи. Це далеко не всі «підводні камені», що виникають при застосуванні параметризованого програмування.

Параметризовані методи визначаються в середині як звичайних класів так і параметризованих. На відміну від звичайних методів параметризовані методи мають параметризований тип, що дозволяє за їх допомогою опрацьовувати різнотипні набори даних. Реальні типи для методів, як і для класів, визначаються у місці виклику методу шляхом передачі реального типу у трикутних дужках.

Бувають ситуації, коли клас або метод потребують накладення обмежень на змінні типів. Наприклад, може бути ситуація, коли метод у процесі роботи викликає з-під об’єкта параметризованого типу метод, що визначається у деякому інтерфейсі. У такому випадку немає ніякої гарантії, що цей метод буде реалізований у кожному класі, що передається через змінну типу. Щоб вирішити цю проблему у мові Java можна задати обмеження на множину можливих типів, що можуть бути підставлені замість параметризованого типу. Для цього після змінної типу слід використати ключове слово extends і вказати один суперклас, або довільну кількість інтерфейсів (через знак &), від яких має походити реальний тип, що підставляється замість параметризованого типу. Якщо одночасно вказуються інтерфейси і суперклас, то суперклас має стояти першим у списку типів після ключового слова extends.

**Завдання (Варіант №15)**

1. Створити параметризований клас, що реалізує предметну область задану варіантом “**Вагон**”. Клас має містити мінімум 4 методи опрацювання даних включаючи розміщення та виймання елементів. Парні варіанти реалізують пошук мінімального елементу, непарні – максимального. Написати на мові Java та налагодити програму-драйвер для розробленого класу, яка мстить мінімум 2 різні класи екземпляри яких розмішуються у екземплярі розробленого класу-контейнеру. Програма має розміщуватися в пакеті Група.Прізвище.Lab6 та володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету.
2. Автоматично згенерувати документацію до розробленого пакету.
3. Завантажити код на GitHub згідно методичних вказівок по роботі з GitHub.
4. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації та завантажити його у ВНС.
5. Дати відповідь на контрольні запитання.

**Виконання завдання**

**Cargo.java**

Package 303.Sukharnyk.lab6;  
  
/\*\*  
 \* Клас, що представляє вантаж  
 \*/  
class Cargo implements Comparable<Cargo> {  
 private String description;  
 private double weight;  
 private int priority;  
  
 /\*\*  
 \* Конструктор класу Cargo  
 \* @param description опис вантажу  
 \* @param weight вага вантажу  
 \* @param priority пріоритет вантажу  
 \*/  
 public Cargo(String description, double weight, int priority) {  
 this.description = description;  
 this.weight = weight;  
 this.priority = priority;  
 }  
  
 public double getWeight() {  
 return weight;  
 }  
  
 public int getPriority() {  
 return priority;  
 }  
  
 @Override  
 public String toString() {  
 return "Cargo{description='" + description + "', weight=" + weight +  
 ", priority=" + priority + "}";  
 }  
  
 @Override  
 public int compareTo(Cargo other) {  
 return Integer.compare(this.priority, other.priority);  
 }  
}

**Passenger.java**

package 303.Sukharnyk.lab6;  
  
/\*\*  
 \* Клас, що представляє пасажира  
 \*/  
class Passenger implements Comparable<Passenger> {  
 private String name;  
 private int ticketNumber;  
 private int seatNumber;  
  
 /\*\*  
 \* Конструктор класу Passenger  
 \* @param name ім'я пасажира  
 \* @param ticketNumber номер квитка  
 \* @param seatNumber номер місця  
 \*/  
 public Passenger(String name, int ticketNumber, int seatNumber) {  
 this.name = name;  
 this.ticketNumber = ticketNumber;  
 this.seatNumber = seatNumber;  
 }  
  
 public int getTicketNumber() {  
 return ticketNumber;  
 }  
  
 public int getSeatNumber() {  
 return seatNumber;  
 }  
  
 @Override  
 public String toString() {  
 return "Passenger{name='" + name + "', ticketNumber=" + ticketNumber +  
 ", seatNumber=" + seatNumber + "}";  
 }  
  
 @Override  
 public int compareTo(Passenger other) {  
 return Integer.compare(this.seatNumber, other.seatNumber);  
 }  
}

**RailCar.java**

Package 303.Sukharnyk.lab6;  
  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.Comparator;  
import java.util.List;  
  
/\*\*  
 \* Параметризований клас, що реалізує вагон для перевезення  
 \* @param <T> тип об'єктів, що перевозяться у вагоні  
 \*/  
public class RailCar<T> {  
 private List<T> contents;  
 private final int capacity;  
 private final String type;  
 private final int carNumber;  
  
 /\*\*  
 \* Конструктор класу RailCar  
 \* @param capacity максимальна місткість вагону  
 \* @param type тип вагону  
 \* @param carNumber номер вагону  
 \*/  
 public RailCar(int capacity, String type, int carNumber) {  
 this.contents = new ArrayList<>();  
 this.capacity = capacity;  
 this.type = type;  
 this.carNumber = carNumber;  
 }  
  
 /\*\*  
 \* Додає об'єкт до вагону  
 \* @param item об'єкт для додавання  
 \* @return true якщо об'єкт додано успішно, false якщо вагон заповнений  
 \*/  
 public boolean loadItem(T item) {  
 if (contents.size() < capacity) {  
 return contents.add(item);  
 }  
 return false;  
 }  
  
 /\*\*  
 \* Видаляє об'єкт з вагону  
 \* @param item об'єкт для видалення  
 \* @return true якщо об'єкт видалено успішно  
 \*/  
 public boolean unloadItem(T item) {  
 return contents.remove(item);  
 }  
  
 /\*\*  
 \* Знаходить максимальний елемент у вагоні за заданим компаратором  
 \* @param comparator компаратор для порівняння елементів  
 \* @return максимальний елемент або null якщо вагон порожній  
 \*/  
 public T findMaxElement(Comparator<T> comparator) {  
 if (contents.isEmpty()) {  
 return null;  
 }  
 return Collections.max(contents, comparator);  
 }  
  
 /\*\*  
 \* Перевіряє наявність вільного місця у вагоні  
 \* @return true якщо є вільне місце  
 \*/  
 public boolean hasSpace() {  
 return contents.size() < capacity;  
 }  
  
 /\*\*  
 \* Повертає поточну кількість об'єктів у вагоні  
 \* @return кількість об'єктів  
 \*/  
 public int getCurrentLoad() {  
 return contents.size();  
 }  
  
 /\*\*  
 \* Повертає номер вагону  
 \* @return номер вагону  
 \*/  
 public int getCarNumber() {  
 return carNumber;  
 }  
  
 /\*\*  
 \* Очищає вагон  
 \*/  
 public void clear() {  
 contents.clear();  
 }  
  
 /\*\*  
 \* Повертає список всіх об'єктів у вагоні  
 \* @return список об'єктів  
 \*/  
 public List<T> getAllContents() {  
 return new ArrayList<>(contents);  
 }  
}

**RailCarDriver.java**

Package 303.Sukharnyk.lab6;  
  
/\*\*  
 \* Клас-драйвер для демонстрації роботи класу RailCar  
 \*/  
public class RailCarDriver {  
 public static void main(String[] args) {  
 // Створюємо пасажирський вагон  
 RailCar<Passenger> passengerCar = new RailCar<>(50, "Passenger", 1234);  
  
 // Додаємо пасажирів  
 passengerCar.loadItem(new Passenger("John Doe", 101, 15));  
 passengerCar.loadItem(new Passenger("Jane Smith", 102, 23));  
 passengerCar.loadItem(new Passenger("Bob Johnson", 103, 7));  
  
 // Знаходимо пасажира з найбільшим номером місця  
 Passenger maxSeatPassenger = passengerCar.findMaxElement(Passenger::compareTo);  
 System.out.println("Пасажир з найбільшим номером місця: " + maxSeatPassenger);  
  
 // Створюємо вантажний вагон  
 RailCar<Cargo> cargoCar = new RailCar<>(100, "Cargo", 5678);  
  
 // Додаємо вантажі  
 cargoCar.loadItem(new Cargo("Electronics", 500.0, 1));  
 cargoCar.loadItem(new Cargo("Food supplies", 1000.0, 3));  
 cargoCar.loadItem(new Cargo("Medical supplies", 750.0, 2));  
  
 // Знаходимо вантаж з найвищим пріоритетом  
 Cargo maxPriorityCargo = cargoCar.findMaxElement(Cargo::compareTo);  
 System.out.println("Вантаж з найвищим пріоритетом: " + maxPriorityCargo);  
  
 // Демонструємо інші методи  
 System.out.println("Кількість пасажирів у вагоні: " + passengerCar.getCurrentLoad());  
 System.out.println("Наявність вільних місць у пасажирському вагоні: " +  
 passengerCar.hasSpace());  
  
 // Вивантажуємо вантаж  
 cargoCar.unloadItem(maxPriorityCargo);  
 System.out.println("Кількість вантажів після вивантаження: " +  
 cargoCar.getCurrentLoad());  
 }  
}

**Відповіді на контрольні питання**

1. Дайте визначення терміну «параметризоване програмування». *Параметризоване програмування дозволяє працювати з узагальненими типами, зменшуючи дублювання коду та підвищуючи гнучкість.*
2. Розкрийте синтаксис визначення простого параметризованого класу. class ClassName<T> { /\* поля та методи \*/ }
3. Розкрийте синтаксис створення об’єкту параметризованого класу. *ClassName<Type> obj = new ClassName<>();*
4. Розкрийте синтаксис визначення параметризованого методу. *<T> ReturnType methodName(T param) { /\* тіло методу \*/ }*
5. Розкрийте синтаксис виклику параметризованого методу. *object.methodName(value);*
6. Яку роль відіграє встановлення обмежень для змінних типів? *Обмеження визначають, з якими типами може працювати параметризований клас або метод, забезпечуючи безпеку типів.*
7. Як встановити обмеження для змінних типів? *<T extends SuperType>*
8. Розкрийте правила спадкування параметризованих типів. *Параметризовані типи не можуть бути спадковані, але можна використовувати підстановочні типи.*
9. Яке призначення підстановочних типів? *Підстановочні типи дозволяють працювати з об'єктами будь-яких типів, надаючи гнучкість для параметризації.*
10. Застосування підстановочних типів. *Підстановочні типи використовують для обмеження методів, що працюють із різними типами, не обмежуючи конкретні.*

**Висновок:** Я оволоділа навиками параметризованого програмування мовою Java.