Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра ЕОМ

Лабораторна робота №6

з дисципліни: «Кросплатформенні засоби програмування»

на тему: «Параметризоване програмування»

Варіант № 25

Виконав:

ст. гр. КІ-305

Федусь Н.В.

Прийняв:

Іванов Ю. С.

Львів – 2023

Мета: Оволодіти навиками параметризованого програмування мовою Java.

Завдання:

1. Створити параметризований клас, що реалізує предметну область задану варіантом. 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. Скласти звіт про виконану роботу з приведенням тексту програми, результату її виконання та фрагменту згенерованої документації.

4. Дати відповідь на контрольні запитання.

Код Main.java:

```
package KI305.Fedus.Lab6;


import KI305.Fedus.Lab6.devices.DeviceInterface;
import KI305.Fedus.Lab6.devices.Refrigerator;
import KI305.Fedus.Lab6.devices.Smartphone;

import java.time.LocalDateTime;

public class Main {
public static void main(String[] args) {
GoodsStorage goodsStorage = new GoodsStorage();

//Make some objects
Smartphone phone1 = new Smartphone();
Smartphone phone2 = new Smartphone(DeviceInterface.Color.BLACK);

Refrigerator refrig1 = new Refrigerator();
Refrigerator refrig2 = new Refrigerator(DeviceInterface.Color.YELLOW);
Refrigerator refrig3 = new Refrigerator(DeviceInterface.Color.WHITE, 60);
Refrigerator refrig4 = new Refrigerator(DeviceInterface.Color.BLACK, 80, 1.7, 0.5, 0.6);
```

```
//Make goods for objects
AbstractGood<Smartphone> agphone1 = new AbstractGood(phone1);
AbstractGood<Smartphone> agphone2 = new AbstractGood(phone2, 500);

AbstractGood<Smartphone> agrefrig1 = new AbstractGood(refrig1);
AbstractGood<Smartphone> agrefrig2 = new AbstractGood(refrig2, 600);
AbstractGood<Smartphone> agrefrig3 = new AbstractGood(refrig3, 500, LocalDateTime.of(2021,
12, 3, 5, 0));
AbstractGood<Smartphone> agrefrig4 = new AbstractGood(refrig4, 650, LocalDateTime.of(2020,
4, 19, 13, 0), "Bosch");

goodsStorage.addAGood(agphone1);
goodsStorage.addAGood(agphone2);

goodsStorage.addAGood(agrefrig1);
goodsStorage.addAGood(agrefrig2);
goodsStorage.addAGood(agrefrig3);
goodsStorage.addAGood(agrefrig4);

goodsStorage.printAll();
}
}
```

Код GoodsStorage.java:

```
package KI305.Fedus.Lab6;


import java.util.ArrayList;
import java.util.List;

public class GoodsStorage<Good extends AbstractGood> {
//List of goods
private List<Good> listOfGoods;
private int MAX_CAPACITY;

//Constructors
public GoodsStorage() {
MAX_CAPACITY = 40;
listOfGoods = new ArrayList<>(MAX_CAPACITY);
}
```

```
public GoodsStorage(int capacity) {
MAX_CAPACITY = capacity;
listOfGoods = new ArrayList<>(MAX_CAPACITY);
}

//add a good
public void addAGood(Good good){
if(listOfGoods.size() == MAX_CAPACITY)
System.out.println("The storage is full");
else
listOfGoods.add(good);
}

//take a good
public Good takeAGood(int num){
if(num >= 0 && num < MAX_CAPACITY)
{
Good template = listOfGoods.get(num);
listOfGoods.remove(num);
return template;
}
System.out.println("There is no a good with such id, try: 0 to " + (MAX_CAPACITY - 1));
return null;
}

//print all goods
public void printAll(){
for (int i = 0; i < listOfGoods.size(); ++i)
System.out.printf("Good #%s\n%s", i, listOfGoods.get(i));
}
}
```

Код AbstractGood.java:

```
package KI305.Fedus.Lab6;


import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class AbstractGood<Good> {
//Data fields
private String fullGoodName;
```

```
private double price = 0;
private LocalDateTime deliveredDate;
private LocalDateTime soledDate;
private static final DateTimeFormatter DATE_FORMAT = DateTimeFormatter.ofPattern("dd-MM-
yyyy HH:mm:ss");
private Good good;

//Constructors
public AbstractGood(Good good){
this.good = good;
deliveredDate = LocalDateTime.now();
fullGoodName = good.getClass().getSimpleName();
}

public AbstractGood(Good good, double price){
this.good = good;
this.price = price;
deliveredDate = LocalDateTime.now();
fullGoodName = good.getClass().getSimpleName();
}

public AbstractGood(Good good, double price, LocalDateTime deliveredDate){
this.good = good;
this.price = price;
this.deliveredDate = deliveredDate;
fullGoodName = good.getClass().getSimpleName();
}

public AbstractGood(Good good, double price, LocalDateTime deliveredDate, String
fullGoodName){
this.good = good;
this.price = price;
this.deliveredDate = deliveredDate;
this.fullGoodName = fullGoodName;
}

//getter methods
public double getPrice() {
return price;
}

public LocalDateTime getDeliveredDate() {
```

```
return deliveredDate;
}

public LocalDateTime getSoledDate() {
return soledDate;
}

public String getFullGoodName() {
return fullGoodName;
}

public Good getGood() {
return good;
}

//setter methods
public void setPrice(double price) {
this.price = price;
}

public void setDeliveredDate(LocalDateTime deliveredDate) {
this.deliveredDate = deliveredDate;
}

//sell
public void toSell(){
if(soledDate == null)
soledDate = LocalDateTime.now();
else
System.out.println("Is already soled.");
}

//get good info
public void getGoodInfo(){
System.out.println(toString());
}

@Override
public String toString() {
return String.format(">\t\t\t\t<\n> %s: %s$\n> Delivered date: %s\n> Soled date: %s\n%s\n>\t\t\t\t<\
n",
fullGoodName, (price == 0)?"The price isn't set":price, deliveredDate.format(DATE_FORMAT),
```

```
(soledDate == null)?"Isn't soled":soledDate.format(DATE_FORMAT),
good.toString());
}
}
```

Код devices. DeviceInterface.java:

```
package KI305.Fedus.Lab6.devices;


public interface DeviceInterface {
enum Power {
POWER_OFF, POWER_ON
}

enum Color{
WHITE, BLACK, RED, PINK, YELLOW, GREEN, BLUE
}

void powerOff();
void powerOn();

Color getColor();
}
```

Код devices.Refrigerator.java:

```
package KI305.Fedus.Lab6.devices;


public class Refrigerator implements DeviceInterface{
//Data fields
private Power switchPower = Power.POWER_OFF;
private Color color = Color.WHITE;
private double weight = 70.0;
private double height = 2.0;
private double width = 0.7;
private double length = 0.8;

//Constructors
public Refrigerator() {
}

public Refrigerator(Color color) {
```

```
this.color = color;
}

public Refrigerator(Color color, double weight) {
this.color = color;
this.weight = weight;
}

public Refrigerator(Color color, double weight, double height, double width, double length) {
this.color = color;
this.weight = weight;
this.height = height;
this.width = width;
this.length = length;
}

//Power methods
@Override
public void powerOff(){
if(switchPower == Power.POWER_OFF)
System.out.println("Smartphone is already powered off.");
else {
System.out.println("Smartphone is powered off.");
switchPower = Power.POWER_OFF;
}
}

@Override
public void powerOn(){
if(switchPower == Power.POWER_ON)
System.out.println("Smartphone is already powered on.");
else {
System.out.println("Smartphone is powered on.");
switchPower = Power.POWER_ON;
}
}

//getter methods
@Override
public Color getColor() {
return color;
}
```

```
public double getWeight() {
return weight;
}

public double getHeight() {
return height;
}

public double getWidth() {
return width;
}

public double getLength() {
return length;
}

//toString
@Override
public String toString() {
return String.format("> Color: %s\n> Weight: %s\n> Height: %s\n> Width: %s\n> Length: %s",
color.toString().toLowerCase(), weight, height, width, length);
}
}
```

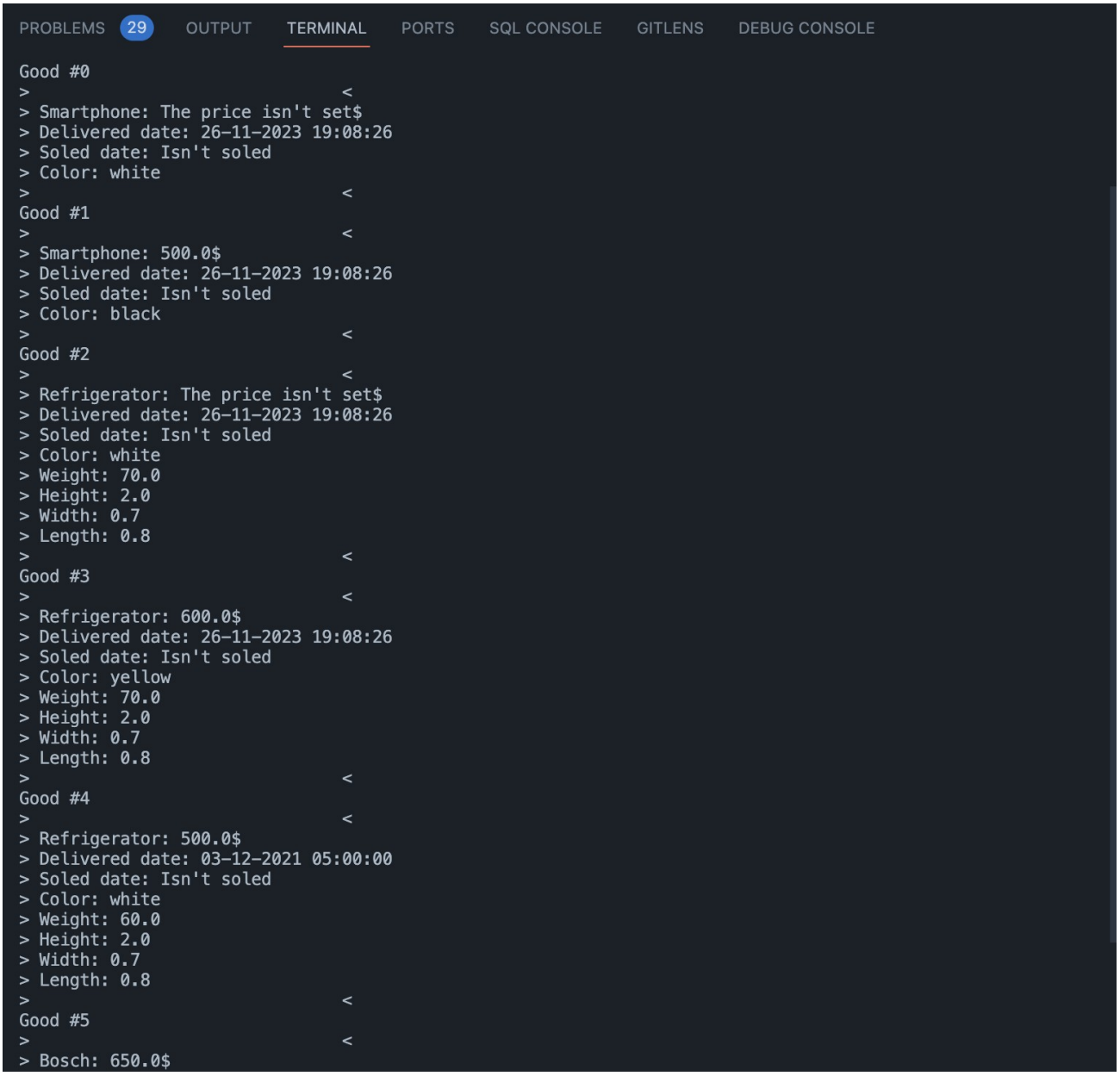
Код devices. Smartphone.java:

```
package KI305.Fedus.Lab6.devices;


public class Smartphone implements DeviceInterface{
//Data fields
private Power switchPower = Power.POWER_OFF;
private Color color = Color.WHITE;
private double charge = 100.0;

//Constructors
public Smartphone() {
}

public Smartphone(Color color) {
this.color = color;
}
```

```
//Power methods
@Override
public void powerOff(){
if(switchPower == Power.POWER_OFF)
System.out.println("Smartphone is already powered off.");
else {
System.out.println("Smartphone is powered off.");
switchPower = Power.POWER_OFF;
}
}

@Override
public void powerOn(){
if(switchPower == Power.POWER_ON)
System.out.println("Smartphone is already powered on.");
else {
System.out.println("Smartphone is powered on.");
switchPower = Power.POWER_ON;
}
}

//getter methods
@Override
public Color getColor() {
return color;
}

public double getCharge() {
return charge;
}

//to charge
public void toCharge(){
if(charge == 100.0)
System.out.println("Is already charged.");
else
charge += 0.1;
}

//work
public void doSomeWork(){
```

```
if(charge <= 20)
System.out.println("Be careful, your phone is running off.");
if(charge > 0) {
System.out.println("Phone is working.");
charge -= 0.2;
}
if(charge <= 0)
switch (switchPower) {
case POWER_ON -> powerOff();
case POWER_OFF -> System.out.println("Your phone is powered off, you can't use it");
}
}

//toString
@Override
public String toString() {
return String.format("> Color: %s", color.toString().toLowerCase());
}
}
```

Результат роботи програми:



Контрольні питання:

1. Дайте визначення терміну «параметризоване програмування».

Відповідь: Параметризоване програмування є аналогом шаблонів у C++. Воно полягає у написанні коду, що можна багаторазово застосовувати з об’єктами різних класів.

2. Розкрийте синтаксис визначення простого параметризованого класу.

Відповідь: Параметризований клас – це клас з однією або більше змінними типу.

[public] class НазваКласу<параметризованийТип {,параметризованийТип} > {…}

3. Розкрийте синтаксис створення об’єкту параметризованого класу.

Відповідь: НазваКласу < перелікТипів > = new НазваКласу < перелікТипів > (параметри);

4. Розкрийте синтаксис визначення параметризованого методу.

Відповідь: Параметризовані методи визначаються в середині як звичайних класів так і параметризованих.

Модифікатори< параметризованийТип{,параметризованийТип} > типПовернення назваМетоду(параметри);

5. Розкрийте синтаксис виклику параметризованого методу.

Відповідь: (НазваКласу|НазваОб’єкту).[<перелікТипів>] НазваМетоду(параметри);

6. Яку роль відіграє встановлення обмежень для змінних типів?

Відповідь: 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.

7. Як встановити обмеження для змінних типів?

Відповідь: 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.

8. Розкрийте правила спадкування параметризованих типів.

Відповідь:

- Всі класи, що утворені з одного і того ж параметризованого класу з використанням різних значень змінних типів є незалежними навіть якщо між цими типами є залежність спадкування.

- Завжди можна перетворити параметризований клас у «сирий» клас.

- Параметризовані класи можуть розширювати або реалізовувати інші параметризовані класи.

9. Яке призначення підстановочних типів?

Відповідь: Підстановочні типи були введені у мову Java для збільшення гнучкості жорсткої існуючої системи параметризованих типів. На відміну від неї підстановочні типи дозволяють враховувати залежності між типами, що виступають параметрами для параметризованих типів.

10. Застосування підстановочних типів.

Відповідь:

- обмеження підтипу;

- обмеження супертипу;

- необмежені підстановки.

Висновок:

Виконавши лабораторну роботу, я оволодів навиками параметризованого програмування мовою Java.