Міністерство освіти і науки України

Національний університет „Львівська політехніка”

Кафедра “Електронних обчислювальних машин”



**з дисципліни**

**Кросплатформні засоби програмування**

**Звіт з Лабораторної роботи № 5**

**ФАЙЛИ У JAVA**

**Виконав:**

КІ-305

Гунька А.А.

**Перевірив:**

Іванов Ю.С

**2023**

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

**Теоретичний вступ**

ТЕОРЕТИЧНІ ВІДОМОСТІ

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.

Визначення простого параметризованого класу

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

Тут T і U – це змінні параметризованих типів, що використовуються по всьому тілу

класу для специфікації типу повернення методів, типів полів і локальних змінних. При

створенні об’єкту параметризованого класу замість них підставляються реальні типи, що

визначаються в трикутних дужках у місці створення об’єкту параметризованого класу.

Синтаксис створення об’єкту параметризованого класу:

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

Приклад створення об’єкту параметризованого класу:

GenericClass<String, Integer> obj = new GenericClass<String, Integer> ();

**Варіант 6**

6. Шафа

**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.
2. Програма має розміщуватися в пакеті Група.Прізвище.Lab6 та володіти коментарями, які дозволять автоматично згенерувати документацію до розробленого пакету. 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 їївиконання та фрагменту згенерованої документації та завантажити його у ВНС.
5. Дати відповідь на контрольні запитання.

**Код**

import java.util.ArrayList;  
  
*/\*\*  
 \* Class Wardrobe implements wardrobe  
 \*  
 \* @author Adrian Hunka  
 \* @version 1.0  
 \*/*public class Wardrobe<T extends Thing> {  
  
 */\*\*  
 \* Constructor  
 \*/* public Wardrobe() {  
 list = new ArrayList<T>();  
 }  
  
 ArrayList<T> list;  
  
 */\*\*  
 \* Method puts thing into a box  
 \*/* public void add(T t) {  
 list.add(t);  
 }  
  
 */\*\*  
 \* Method prints info about thing  
 \*/* public void get(int index) {  
 list.get(index).print();  
 }  
  
 */\*\*  
 \* Method removes thing from box  
 \*/* public void remove(int index) {  
 list.remove(index);  
 }  
  
 */\*\*  
 \* Method prints weight of box  
 \*/* public void weight() {  
 int sumWeight = 0;  
 for (T t : list)  
 sumWeight += t.getWeight();  
 System.*out*.println("Total weight of things in this wardrobe is: " + sumWeight + " g");  
 }  
  
 */\*\*  
 \* Method returns thing with the little weight  
 \*/* public T findMin() {  
 T min = list.get(0);  
 for (T t : list)  
 if (t.compareTo(min) == -1)  
 min = t;  
 return min;  
 }  
}

*/\*\*  
 \* Interface Thing describes rules for thing type  
 \*  
 \* @author Adrian Hunka  
 \* @version 1.0  
 \*/*interface Thing {  
  
 */\*\*  
 \* Method prints info about thing  
 \*/* public void print();  
  
 */\*\*  
 \* Method returns thing's weight  
 \*/* public int getWeight();  
  
 */\*\*  
 \* Method compares current Thing with another Thing  
 \*  
 \*/* public int compareTo(Thing d);  
}

*/\*\*  
 \* Class Shirt implements shirt  
 \*  
 \* @author Adrian Hunka  
 \* @version 1.0  
 \*/*public class Shirt implements Thing {  
  
 */\*\*  
 \* Constructor  
 \*  
 \* @param weight  
 \*/* public Shirt(int weight) {  
 this.weight = weight;  
 }  
  
 */\*\*  
 \* Constructor  
 \*/* public Shirt() {  
 this.weight = 50;  
 }  
  
 private int weight;  
  
 */\*\*  
 \* Method prints info about thing  
 \*/* @Override  
 public void print() {  
 System.*out*.println("Type of thing: shirt");  
 System.*out*.println("Weight: " + weight + " g");  
 }  
  
 */\*\*  
 \* Method returns thing's weight  
 \*/* @Override  
 public int getWeight() {  
 return weight;  
 }  
  
 */\*\*  
 \* Method compares current Thing with another Thing  
 \*/* @Override  
 public int compareTo(Thing d) {  
 if (this.weight < d.getWeight())  
 return -1;  
 if (this.weight == d.getWeight())  
 return 0;  
 return 1;  
 }  
}

*/\*\*  
 \* Class Pants implements pants  
 \*  
 \* @author Adrian Hunka  
 \* @version 1.0  
 \*/*public class Pants implements Thing {  
  
 */\*\*  
 \* Constructor  
 \*  
 \* @param weight  
 \*/* public Pants(int weight) {  
 this.weight = weight;  
 }  
  
 */\*\*  
 \* Constructor  
 \*/* public Pants() {  
 this.weight = 200;  
 }  
  
 private int weight;  
  
 */\*\*  
 \* Method prints info about thing  
 \*/* @Override  
 public void print() {  
 System.*out*.println("Type of thing: pants");  
 System.*out*.println("Weight: " + weight + " g");  
 }  
  
 */\*\*  
 \* Method returns thing's weight  
 \*/* @Override  
 public int getWeight() {  
 return weight;  
 }  
  
 */\*\*  
 \* Method compares current Thing with another Thing  
 \*/* @Override  
 public int compareTo(Thing d) {  
 if (this.weight < d.getWeight())  
 return -1;  
 if (this.weight == d.getWeight())  
 return 0;  
 return 1;  
 }  
}

*/\*\*  
 \* Driver class implements main method for Wardrobe class possibilities demonstration  
 \*  
 \* @author Adrian Hunka  
 \* @version 1.0  
 \*/*public class Driver {  
  
 public static void main(String[] args) {  
 Wardrobe<? super Thing> wardrobe = new Wardrobe<Thing>();  
 wardrobe.add(new Pants(150));  
 wardrobe.add(new Shirt(80));  
 wardrobe.add(new Pants(180));  
 wardrobe.add(new Shirt());  
 wardrobe.add(new Pants());  
 wardrobe.add(new Shirt(50));  
 wardrobe.weight();  
 wardrobe.remove(0);  
 wardrobe.weight();  
 Thing thing = wardrobe.findMin();  
 System.*out*.println("The lightest thing is: ");  
 thing.print();  
 }  
}

**Результат:**

Книги у шафі:

Book{title='Java Programming', pageCount=300}

Book{title='Data Structures', pageCount=250}

Мінімальна кількість сторінок: 250

Посуд у шафі:

Dish{name='Plate', weight=0.5}

Dish{name='Cup', weight=0.3}

Максимальна вага посуду: 0.5

**Короткий висновок**

Під час виконання даної лабораторної роботи я оволодів навиками параметризованого програмування мовою Java.