Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра ЕОМ



**Звіт**

до лабораторної роботи №6

З дисципліни: «Кросплатформні засоби програмування»

На тему: «ПАРАМЕТРИЗОВАНЕ ПРОГРАМУВАННЯ»

**Варіант 3**

Виконав: ст. гр. КІ-305

Гнідець В.М.

Прийняв:

Іванов Ю.С.

Львів 2023

**Мета роботи:** 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 9 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.

**Варіант 3:** Побутовий пакет

**Код програми**

**Class HouseholdPackageDriver**

package KI305\_Gnidec\_Lab6;  
  
*/\*\*  
 \* Клас, який демонструє використання класів HouseholdItem та  
 HouseholdPackage.  
 \*/*public class HouseholdPackageDriver {  
 public static void main(String[] args) {  
 HouseholdPackage<HouseholdItem> package1 = new HouseholdPackage<>();  
 HouseholdPackage<HouseholdItem> package2 = new HouseholdPackage<>();  
  
 HouseholdItem item1 = new HouseholdItem("Диван", 25);  
 HouseholdItem item2 = new HouseholdItem("Телевізор", 50);  
 HouseholdItem item3 = new HouseholdItem("Посудомийна машина", 90);  
 HouseholdItem item4 = new HouseholdItem("Диван", 10);  
 HouseholdItem item5 = new HouseholdItem("Диван", 15);  
  
  
  
 package1.addItem(item1);  
 package1.addItem(item2);  
 package1.addItem(item3);  
  
 System.*out*.println();  
  
 package2.addItem(item4);  
 package2.addItem(item5);  
 System.*out*.print("Кількість диванів в пакеті: ");  
  
 System.*out*.println(HouseholdItem.*getAmount*());  
  
  
 HouseholdItem maxItem1 = package1.findMax();  
  
 if (maxItem1 != null) {  
 System.*out*.print("Найдорожчий предмет в пакеті 1: ");  
 maxItem1.print();  
 } else {  
 System.*out*.println("Пакет 1 порожній.");  
 }  
  
 HouseholdItem maxItem2 = package2.findMax();  
  
 if (maxItem2 != null) {  
 System.*out*.print("Найдорожчий предмет в пакеті 2: ");  
 maxItem2.print();  
 } else {  
 System.*out*.println("Пакет 1 порожній.");  
 }  
  
 System.*out*.println();  
 System.*out*.println("Видаляємо предмет з пакету 1.");  
  
 package1.removeItem(2);  
 System.*out*.println();  
  
  
 System.*out*.println("Видаляємо предмет з пакету 2.");  
  
 package2.removeItem(0);  
 System.*out*.println();  
  
  
 System.*out*.println( item2.compareTo(item1));  
  
 }  
}

**Class HouseholdPackage**

package KI305\_Gnidec\_Lab6;  
import java.util.ArrayList;  
*/\*\*  
 \* Клас, що представляє пакет для зберігання побутових предметів.  
 \* Пакет може містити об'єкти, які є підкласами класу  
 HouseholdItem.  
 \*  
 \* @param* <*T*> *тип об'єктів, які можуть бути додані до пакету.  
 \*/*public class HouseholdPackage<T extends Data> {  
 */\*\*  
 \* Список елементів у пакеті.  
 \*/* private ArrayList<T> items;  
  
 */\*\*  
 \* Конструктор за замовчуванням, створює порожній пакет.  
 \*/* public HouseholdPackage() {  
 items = new ArrayList<>();  
 }  
  
 */\*\*  
 \* Пошук найбільшого об'єкта у пакеті за їхньою ціною.  
 \*  
 \* @return найдорожчий об'єкт у пакеті або null, якщо пакет  
 \* порожній.  
 \*/* public T findMax() {  
 if (!items.isEmpty()) {  
 T max = items.get(0);  
 for (int i = 1; i < items.size(); i++) {  
 if (items.get(i).compareTo(max) > 0) {  
 max = items.get(i);  
 }  
 }  
 return max;  
 }  
 return null;  
 }  
  
 */\*\*  
 \* Додає об'єкт до пакету і виводить повідомлення про успішне  
 \* додавання.  
 \*  
 \* @param item об'єкт, який додається до пакету.  
 \*/* public void addItem(T item) {  
 items.add(item);  
 System.*out*.print("Предмет додано: ");  
 item.print();  
 }  
  
 */\*\*  
 \* Видаляє об'єкт із пакету за вказаним індексом і виводить  
 \* повідомлення про успішне видалення.  
 \*  
 \* @param index індекс об'єкта, який потрібно видалити.  
 \*/* public void removeItem(int index) {  
 if (index >= 0 && index < items.size()) {  
 T removedItem = items.remove(index);  
 System.*out*.print("Предмет видалено: ");  
 removedItem.print();  
 } else {  
 System.*out*.println("Неіснуючий індекс. Предмет не видалено");  
 }  
 }  
}

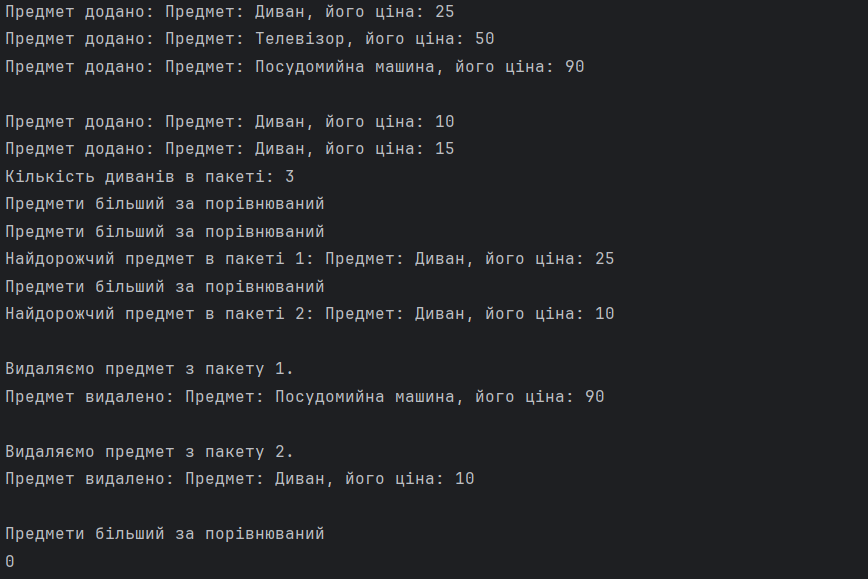
**Class HouseholdItem**

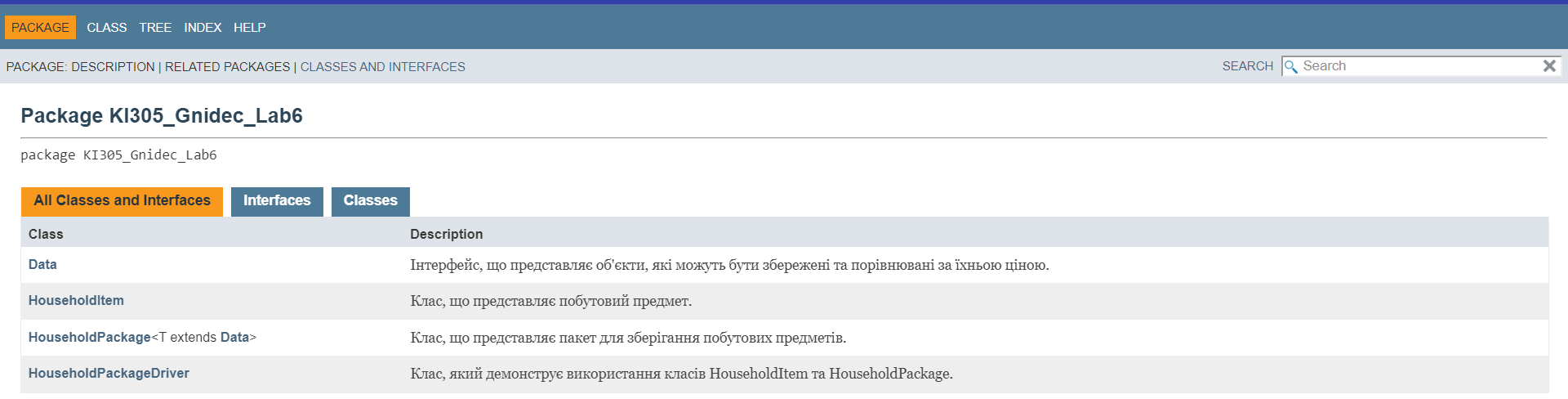
package KI305\_Gnidec\_Lab6;  
  
*/\*\*  
 \* Клас, що представляє побутовий предмет.  
 \* Кожен побутовий предмет має назву (itemName) та ціну (price).  
 \*/*public class HouseholdItem implements Data {  
 */\*\*  
 \* Назва побутового предмета.  
 \*/* private String itemName;  
 */\*\*  
 \* Ціна побутового предмета.  
 \*/* private int price;  
  
 */\*\*  
 \* Конструктор, що ініціалізує назву та ціну побутового  
 \* предмета.  
 \*  
 \* @param name назва побутового предмета.  
 \* @param price ціна побутового предмета.  
 \*/* static int *count* = 0;  
 public HouseholdItem(String name, int price) {  
 this.itemName = name;  
 this.price = price;  
  
 if(name == "Диван"){  
 *count*++;  
 }  
 }  
  
 */\*\*  
 \* Повертає назву побутового предмета.  
 \*  
 \* @return назва побутового предмета.  
 \*/* public String getItemName() {  
 return itemName;  
 }  
 public static int getAmount(){  
 return *count*;  
 }  
 */\*\*  
 \* Повертає ціну побутового предмета.  
 \*  
 \* @return ціну побутового предмета.  
 \*/* public int getPrice() {  
 return price;  
 }  
 */\*\*  
 \* Порівнює побутові предмети за їхньою ціною для  
 \* сортування.  
 \*  
 \* @param otherItem інший побутовий предмет для порівняння.  
 \* @return від'ємне число, нуль або додатне число, якщо цей  
 \* об'єкт менший, рівний або більший за інший предмет.  
 \*/* @Override  
 public int compareTo(Data otherItem) {  
  
 int u = Integer.*compare*(this.price, otherItem.getPrice());  
 if(u==0){  
 System.*out*.println("Предмети однакової ціни");  
 }  
 else if(u==1){  
 System.*out*.println("Предмети більший за порівнюваний");  
 }  
 else if (u==-1){  
 System.*out*.println("Предмети менший за порівнюваний");  
 }  
 else{  
 System.*out*.println("Якийсь з предметів недоступний");  
  
 }  
 return 0;  
 }  
  
 */\*\*  
 \* Виводить інформацію про побутовий предмет (назва та ціна)  
 \* у консоль.  
 \*/* public void print() {  
 System.*out*.println("Предмет: " + itemName + ", його ціна: " + price);  
 }  
}

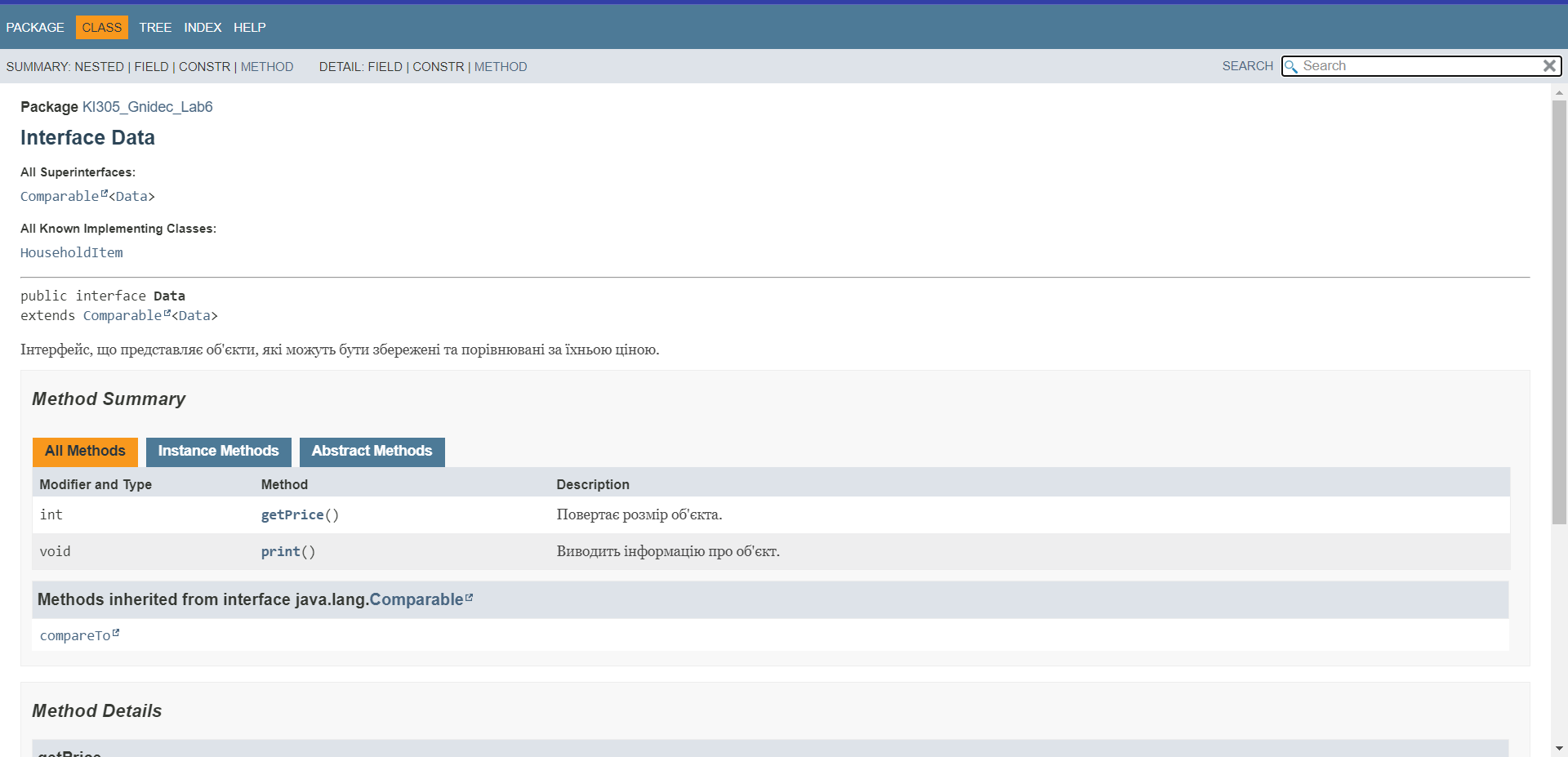
**Interface HouseholdItem**

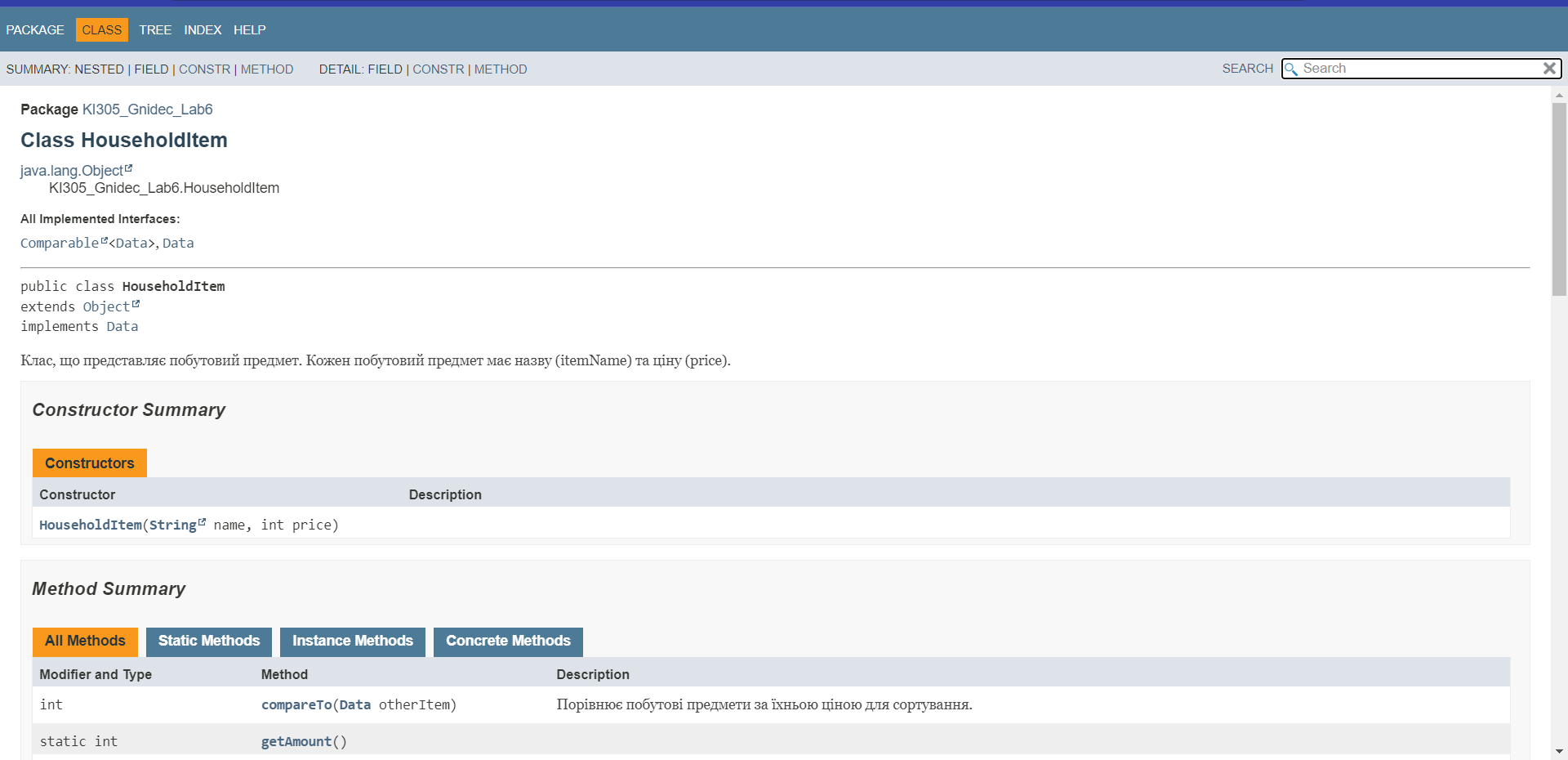
package KI305\_Gnidec\_Lab6;  
*/\*\*  
 \* Інтерфейс, що представляє об'єкти, які можуть бути збережені та  
 порівнювані за їхньою ціною.  
 \*/*public interface Data extends Comparable<Data> {  
 */\*\*  
 \* Повертає розмір об'єкта.  
 \*  
 \* @return розмір об'єкта.  
 \*/* int getPrice();  
  
 */\*\*  
 \* Виводить інформацію про об'єкт.  
 \*/* void print();  
}

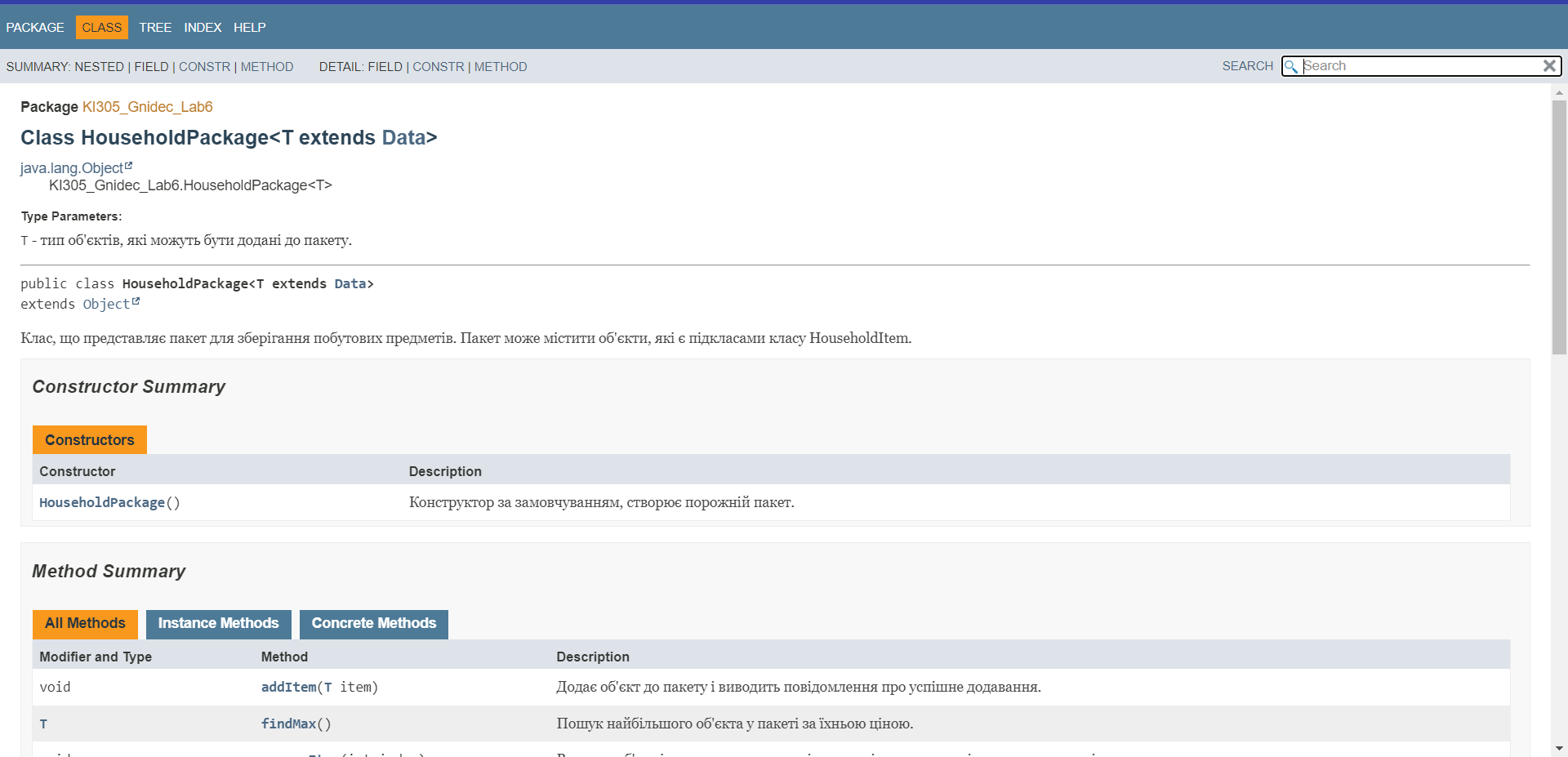
**Результат роботи програми:**

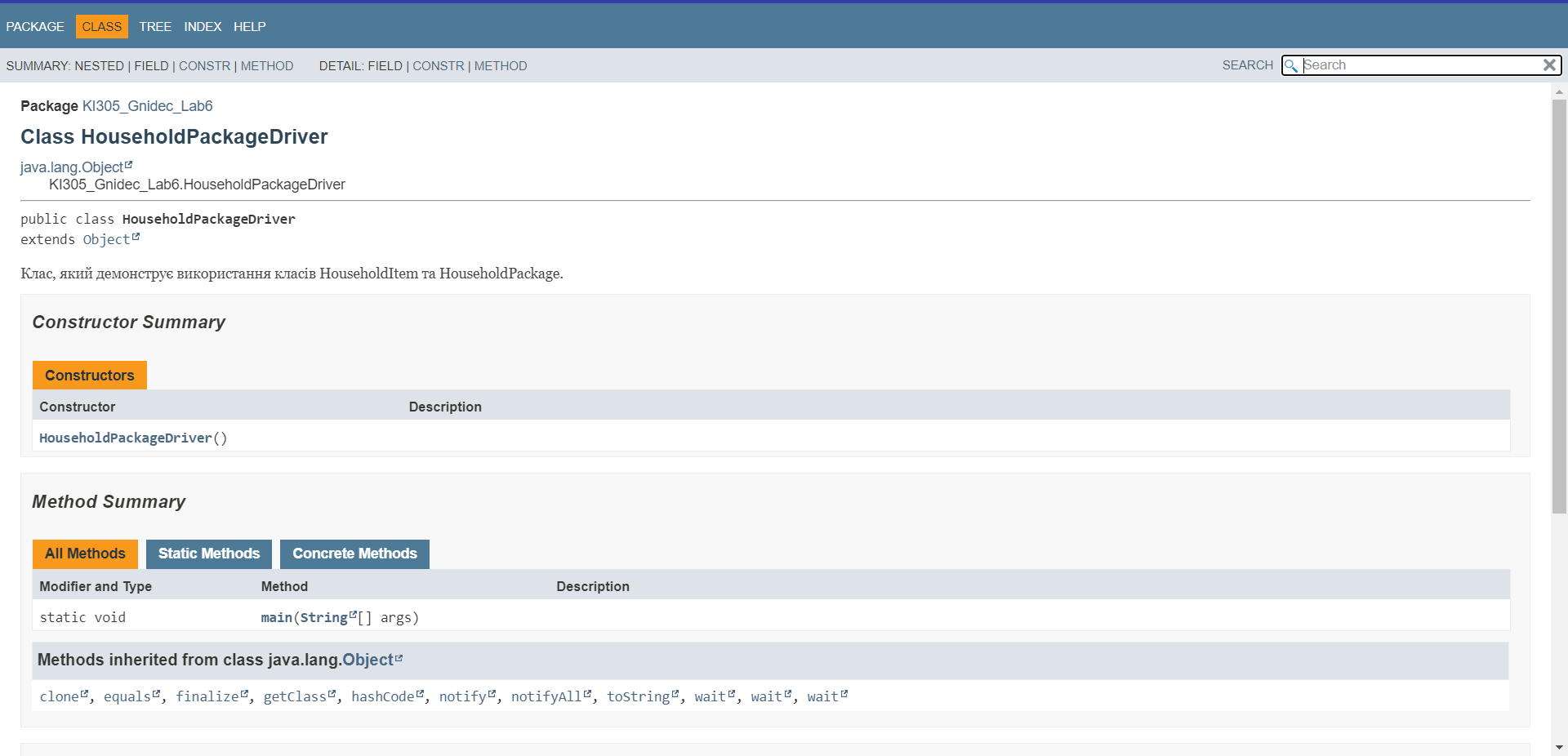
****

****

****





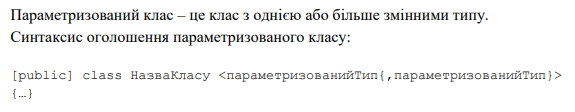


**Контрольні запитання**

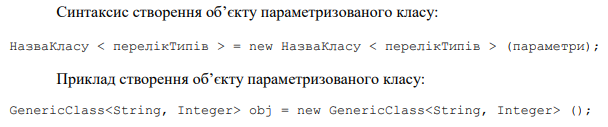
1. ***Дайте визначення терміну «параметризоване програмування».***

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.

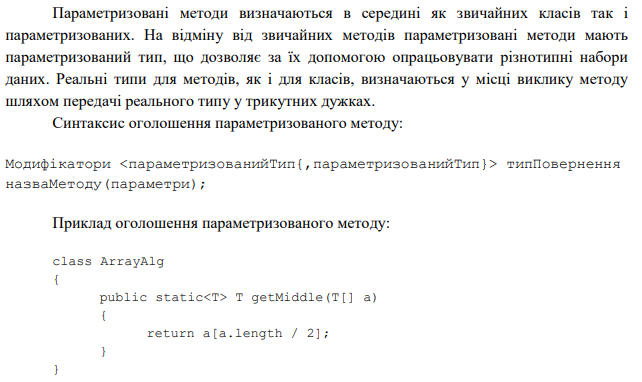
***2. Розкрийте синтаксис визначення простого параметризованого класу.***

******

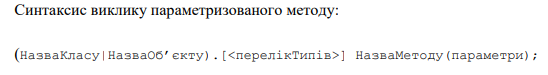
***3. Розкрийте синтаксис створення об’єкту параметризованого класу.***

******

***4. Розкрийте синтаксис визначення параметризованого методу.***

******

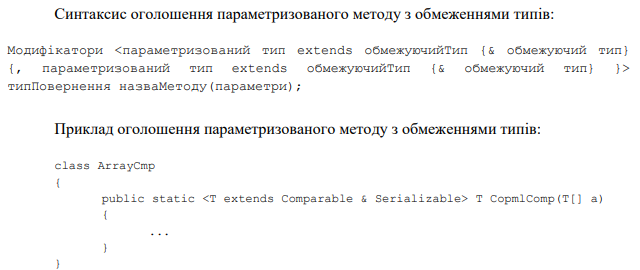
***5. Розкрийте синтаксис виклику параметризованого методу.***

******

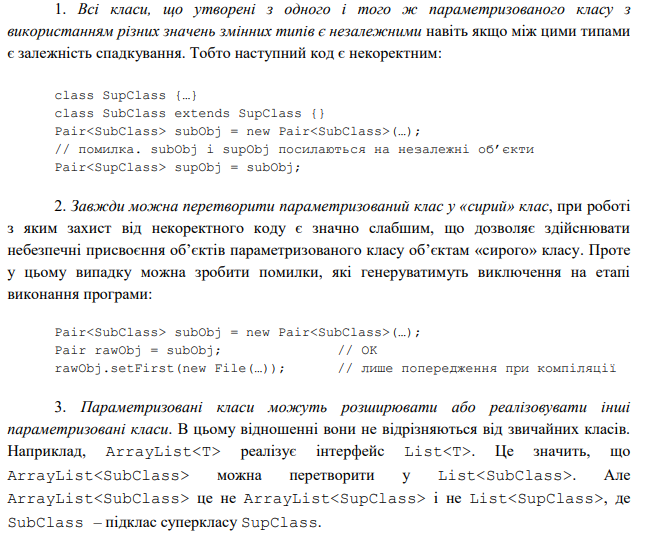
1. ***Яку роль відіграє встановлення обмежень для змінних типів?***

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.

1. ***Як встановити обмеження для змінних типів?***

******

***8. Розкрийте правила спадкування параметризованих типів.***

******

***9. Яке призначення підстановочних типів?***

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. Це в свою чергу дозволяє застосовувати обмеження для параметрів, що підставляються замість параметризованих типів. Завдяки цьому підвищується надійність параметризованого коду, полегшується робота з ним та розділяється використання безпечних методів доступу і небезпечних модифікуючих методів. Підстановочні типи застосовуються у вигляді параметру типу, що передається у трикутних дужках при утворені реального типу з параметризованого типу, наприклад, у методі main.

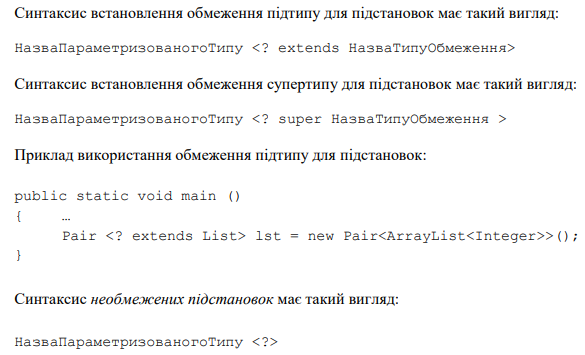
Підстановочні типи дозволяють реалізувати:

1. обмеження підтипу;

2. обмеження супертипу;

3. необмежені підстановки

***10. Застосування підстановочних типів.***

******

**Висновок:** оволодів навиками параметризованого програмування мовою Java.