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 - 7

Виконала:

ст. гр. КІ-305

Дзера А. Р.

Прийняв:

доц. каф. ЕОМ

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 містить мінімум 2 різні класи екземпляри які 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.

Варіант завдання – трюм корабля

**Код програми**

**Файл ShipHold.java**

package KI305.Dzera.Lab6;  
  
import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \* Клас ShipHold представляє трюм судна для зберігання вантажу. Він може містити об'єкти  
 \* класу, який розширює клас CargoItem.  
 \*  
 \** ***@param*** <*T*> *тип вантажу, який можна зберігати в трюмі, розширює клас CargoItem  
 \*/*public class ShipHold<T extends CargoItem> {  
 private List<T> items; // список для зберігання вантажу  
  
 */\*\*  
 \* Конструктор класу ShipHold. Ініціалізує пустий список для зберігання вантажу.  
 \*/* public ShipHold() {  
 items = new ArrayList<>();  
 }  
  
 */\*\*  
 \* Додає вантаж до трюму.  
 \*  
 \** ***@param*** *item вантаж, який додається до трюму  
 \*/* public void placeItem(T item) {  
 items.add(item);  
 }  
  
 */\*\*  
 \* Видаляє та повертає останній доданий вантаж з трюму.  
 \*  
 \** ***@return*** *вантаж, який був видалений з трюму, або null, якщо трюм пустий  
 \*/* public T removeItem() {  
 if (!items.isEmpty()) {  
 return items.remove(items.size() - 1);  
 } else {  
 return null;  
 }  
 }  
  
 */\*\*  
 \* Знаходить та повертає вантаж з найбільшим об'ємом в трюмі.  
 \*  
 \** ***@return*** *вантаж з найбільшим об'ємом в трюмі, або null, якщо трюм пустий  
 \*/* public T findMaxItem() {  
 if (!items.isEmpty()) {  
 T max = items.get(0);  
 for (T item : items) {  
 if (item.getVolume() > max.getVolume()) {  
 max = item;  
 }  
 }  
 return max;  
 } else {  
 return null;  
 }  
 }  
  
 */\*\*  
 \* Підраховує кількість вантажу з певним вмістом в трюмі.  
 \*  
 \** ***@param*** *content вміст, який шукається у вантажі  
 \** ***@return*** *кількість вантажу з вказаним вмістом в трюмі  
 \*/* public int countItemsByContent(String content) {  
 int count = 0;  
 for (T item : items) {  
 if (item instanceof Barrel) {  
 Barrel barrel = (Barrel) item;  
 if (barrel.getContent().equals(content)) {  
 count++;  
 }  
 }  
 }  
 return count;  
 }  
}

**Файл Barrel.java**

package KI305.Dzera.Lab6;  
  
*/\*\*  
 \* Клас Barrel представляє бочку вантажу для зберігання на судні. Реалізує інтерфейс CargoItem,  
 \* надаючи змогу отримати вміст бочки та її об'єм.  
 \*/*public class Barrel implements CargoItem {  
 private String content; // вміст бочки  
 private int volume; // об'єм бочки  
  
 */\*\*  
 \* Конструктор класу Barrel.  
 \*  
 \** ***@param*** *content вміст бочки  
 \** ***@param*** *volume об'єм бочки в кубічних метрах  
 \*/* public Barrel(String content, int volume) {  
 this.content = content;  
 this.volume = volume;  
 }  
  
 */\*\*  
 \* Отримує вміст бочки.  
 \*  
 \** ***@return*** *вміст бочки  
 \*/* public String getContent() {  
 return content;  
 }  
  
 */\*\*  
 \* Реалізація методу інтерфейсу CargoItem для отримання об'єму бочки.  
 \*  
 \** ***@return*** *об'єм бочки в кубічних метрах  
 \*/* @Override  
 public int getVolume() {  
 return volume;  
 }  
  
 */\*\*  
 \* Реалізація методу інтерфейсу CargoItem для виведення інформації про бочку.  
 \*/* @Override  
 public void displayInfo() {  
 System.*out*.println("Barrel with " + content + ", volume: " + volume + " cubic meters.");  
 }  
}

**Файл CargoBox.java**

package KI305.Dzera.Lab6;  
  
*/\*\*  
 \* Клас CargoBox представляє товарний ящик вантажу для зберігання на судні. Реалізує інтерфейс CargoItem,  
 \* надаючи змогу отримати назву ящика та його об'єм.  
 \*/*public class CargoBox implements CargoItem {  
 private String name; // назва товарного ящика  
 private int volume; // об'єм товарного ящика  
  
 */\*\*  
 \* Конструктор класу CargoBox.  
 \*  
 \** ***@param*** *name назва товарного ящика  
 \** ***@param*** *volume об'єм товарного ящика в кубічних метрах  
 \*/* public CargoBox(String name, int volume) {  
 this.name = name;  
 this.volume = volume;  
 }  
  
 */\*\*  
 \* Реалізація методу інтерфейсу CargoItem для отримання об'єму товарного ящика.  
 \*  
 \** ***@return*** *об'єм товарного ящика в кубічних метрах  
 \*/* @Override  
 public int getVolume() {  
 return volume;  
 }  
  
 */\*\*  
 \* Реалізація методу інтерфейсу CargoItem для виведення інформації про товарний ящик.  
 \*/* @Override  
 public void displayInfo() {  
 System.*out*.println("Cargo Box: " + name + ", Volume: " + volume + " cubic meters");  
 }  
}

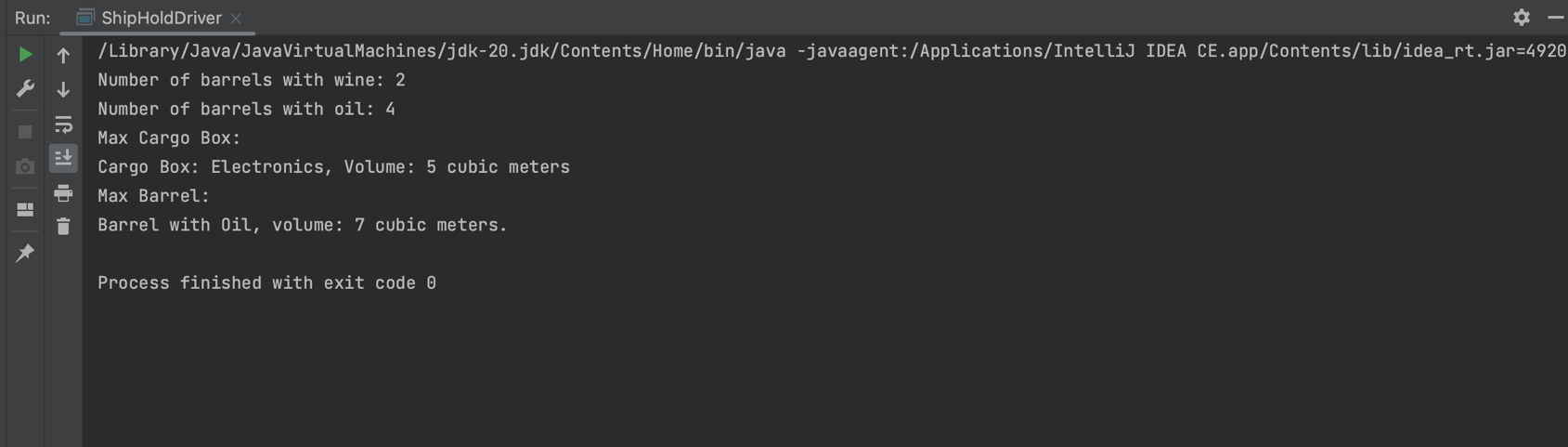
**Файл ShipHoldDriver.java**

package KI305.Dzera.Lab6;  
  
*/\*\*  
 \* Клас ShipHoldDriver є вхідною точкою програми та демонструє використання класів ShipHold, CargoBox і Barrel.  
 \* Створюється тримач для товарних ящиків та бочок, заповнюється вантажем, рахується кількість бочок з певним  
 \* вмістом та виводяться інформація про товари з найбільшим об'ємом в тримачі.  
 \*/*public class ShipHoldDriver {  
 public static void main(String[] args){  
 // Створення та заповнення тримача для товарних ящиків  
 ShipHold<CargoBox> cargoBoxHold = new ShipHold<>();  
 cargoBoxHold.placeItem(new CargoBox("Electronics", 5));  
 cargoBoxHold.placeItem(new CargoBox("Clothes", 2));  
  
 // Створення та заповнення тримача для бочок  
 ShipHold<Barrel> barrelHold = new ShipHold<>();  
 barrelHold.placeItem(new Barrel("Oil", 2));  
 barrelHold.placeItem(new Barrel("Wine", 4));  
 barrelHold.placeItem(new Barrel("Oil", 3));  
 barrelHold.placeItem(new Barrel("Wine", 5));  
 barrelHold.placeItem(new Barrel("Oil", 1));  
 barrelHold.placeItem(new Barrel("Oil", 7));  
  
 // Підрахунок кількості бочок з певним вмістом  
 int wineBarrelsCount = barrelHold.countItemsByContent("Wine");  
 int oilBarrelsCount = barrelHold.countItemsByContent("Oil");  
  
 // Виведення кількості бочок з певним вмістом  
 System.*out*.println("Number of barrels with wine: " + wineBarrelsCount);  
 System.*out*.println("Number of barrels with oil: " + oilBarrelsCount);  
  
 // Пошук та виведення товару з найбільшим об'ємом для товарних ящиків  
 CargoBox maxCargoBox = cargoBoxHold.findMaxItem();  
 if(maxCargoBox != null){  
 System.*out*.println("Max Cargo Box: ");  
 maxCargoBox.displayInfo();  
 } else {  
 System.*out*.println("No cargo boxes in the hold.");  
 }  
  
 // Пошук та виведення товару з найбільшим об'ємом для бочок  
 Barrel maxBarrel = barrelHold.findMaxItem();  
 if(maxBarrel != null){  
 System.*out*.println("Max Barrel: ");  
 maxBarrel.displayInfo();  
 } else {  
 System.*out*.println("No barrels in the hold.");  
 }  
 }  
}

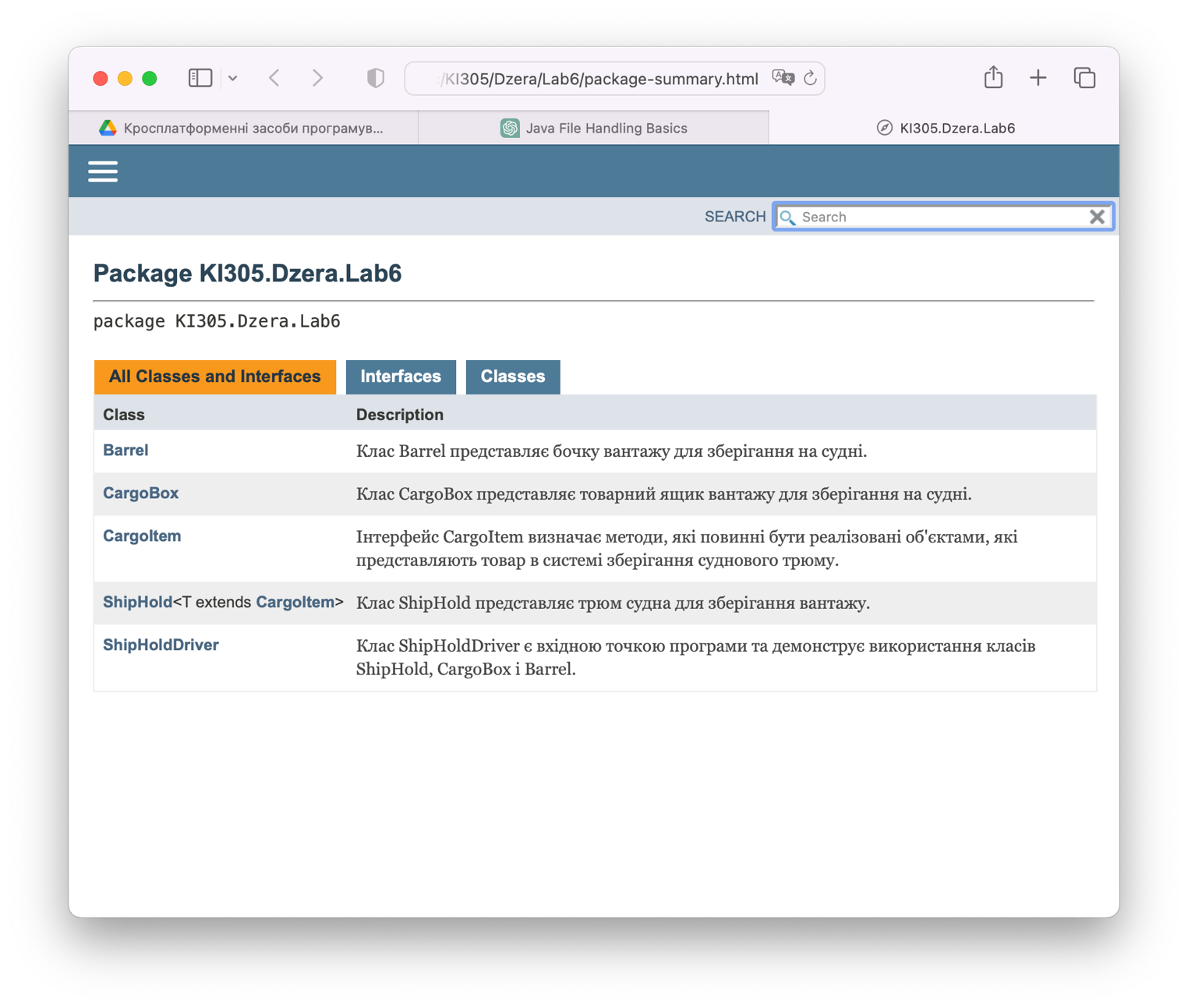
**Файл CargoItem.java**

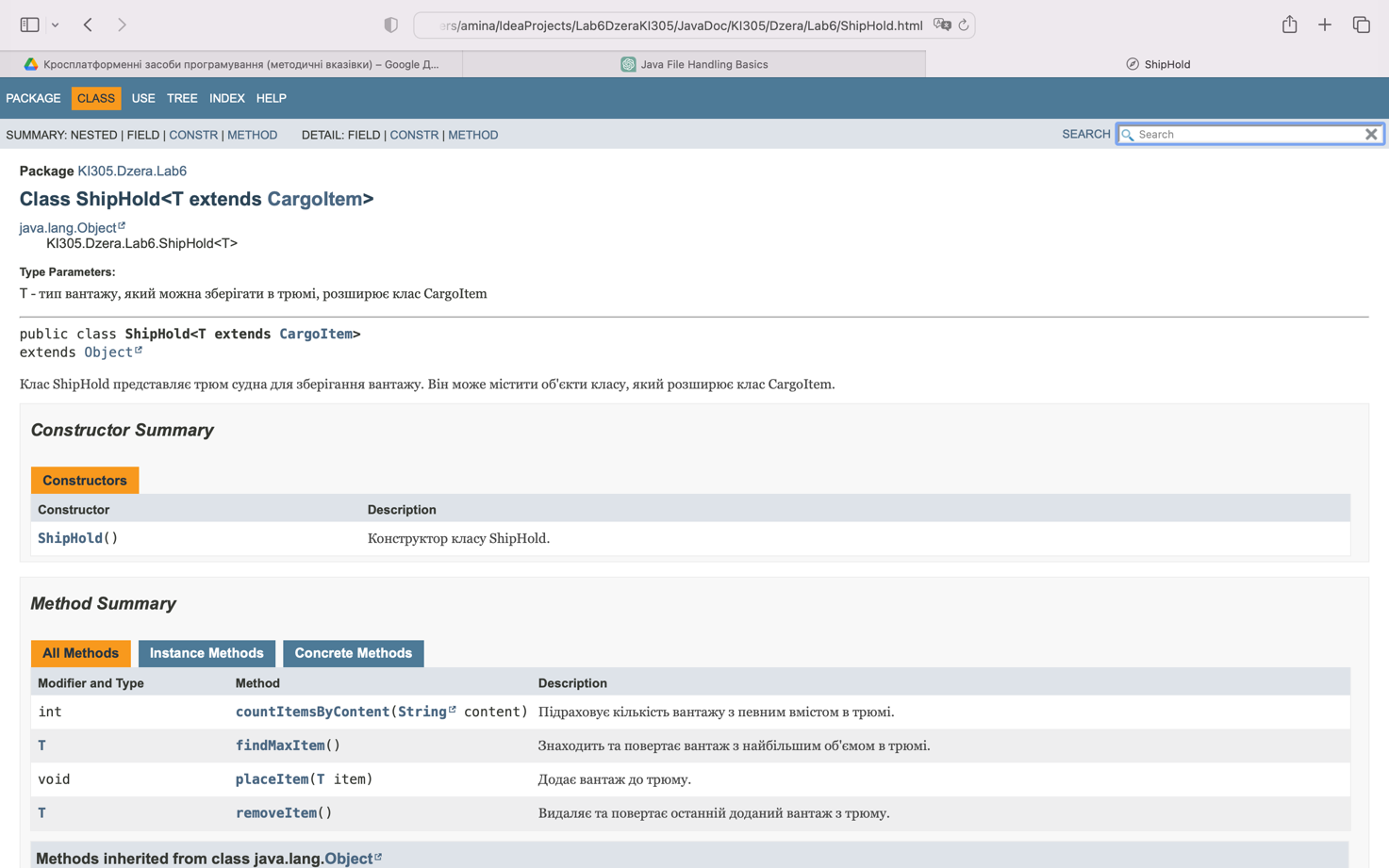
package KI305.Dzera.Lab6;  
  
*/\*\*  
 \* Інтерфейс CargoItem визначає методи, які повинні бути реалізовані об'єктами,  
 \* які представляють товар в системі зберігання суднового трюму.  
 \*/*public interface CargoItem {  
  
 */\*\*  
 \* Метод, який повертає об'єм товару.  
 \*  
 \** ***@return*** *об'єм товару в кубічних метрах  
 \*/* int getVolume();  
  
 */\*\*  
 \* Метод, який виводить інформацію про товар.  
 \* Реалізація цього методу повинна виводити назву товару та його об'єм.  
 \*/* void displayInfo();  
}

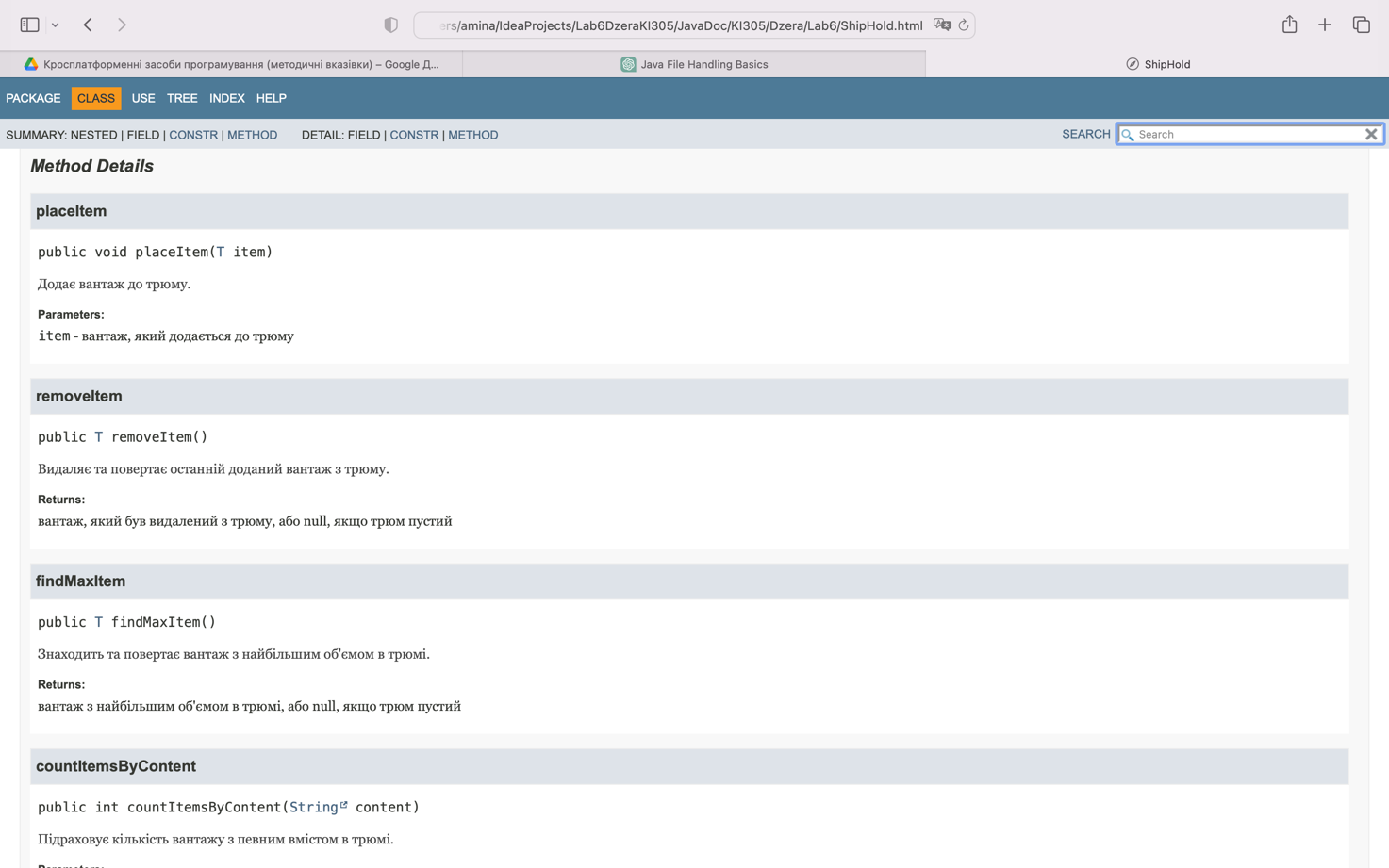
**Результати роботи програми**

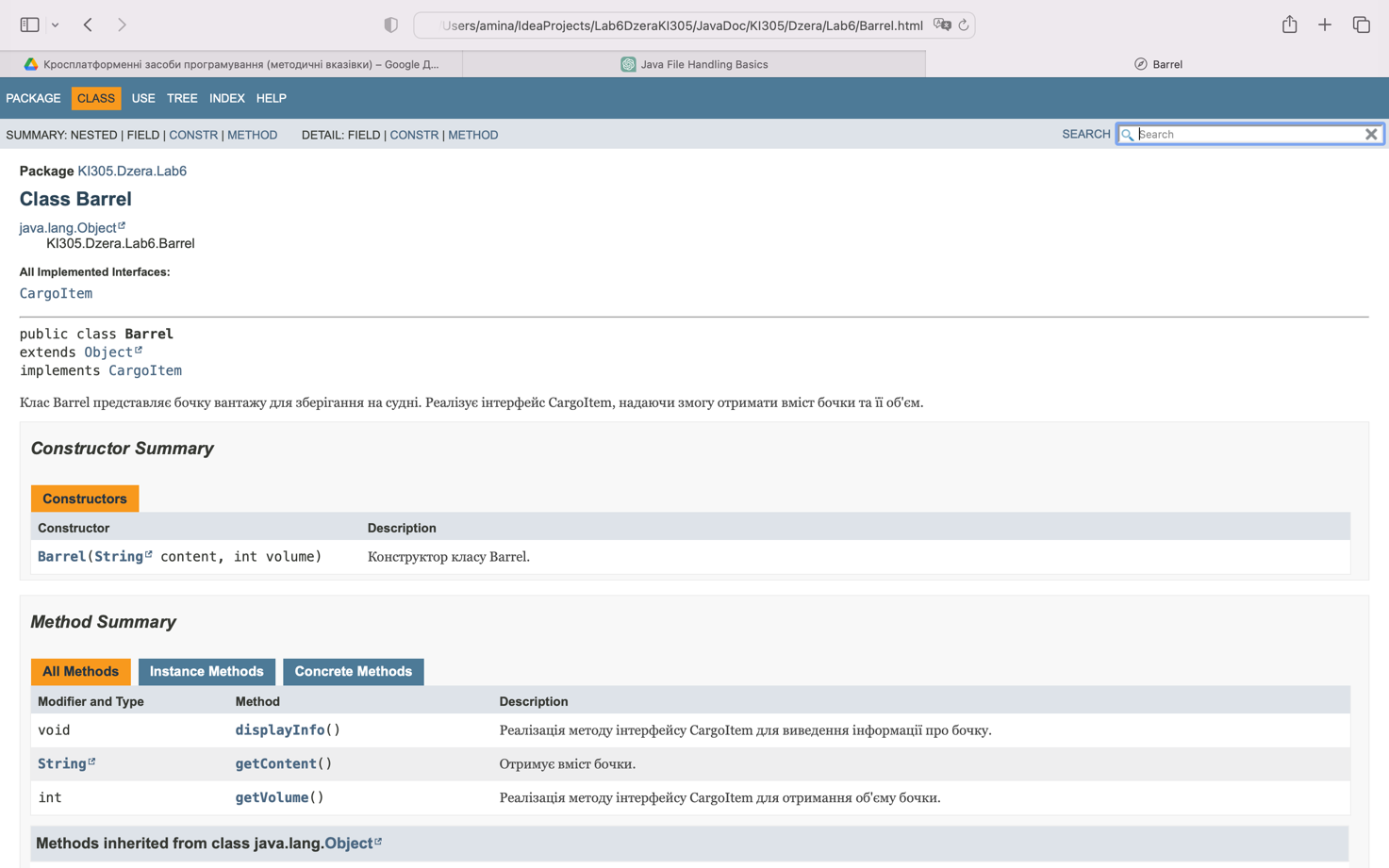


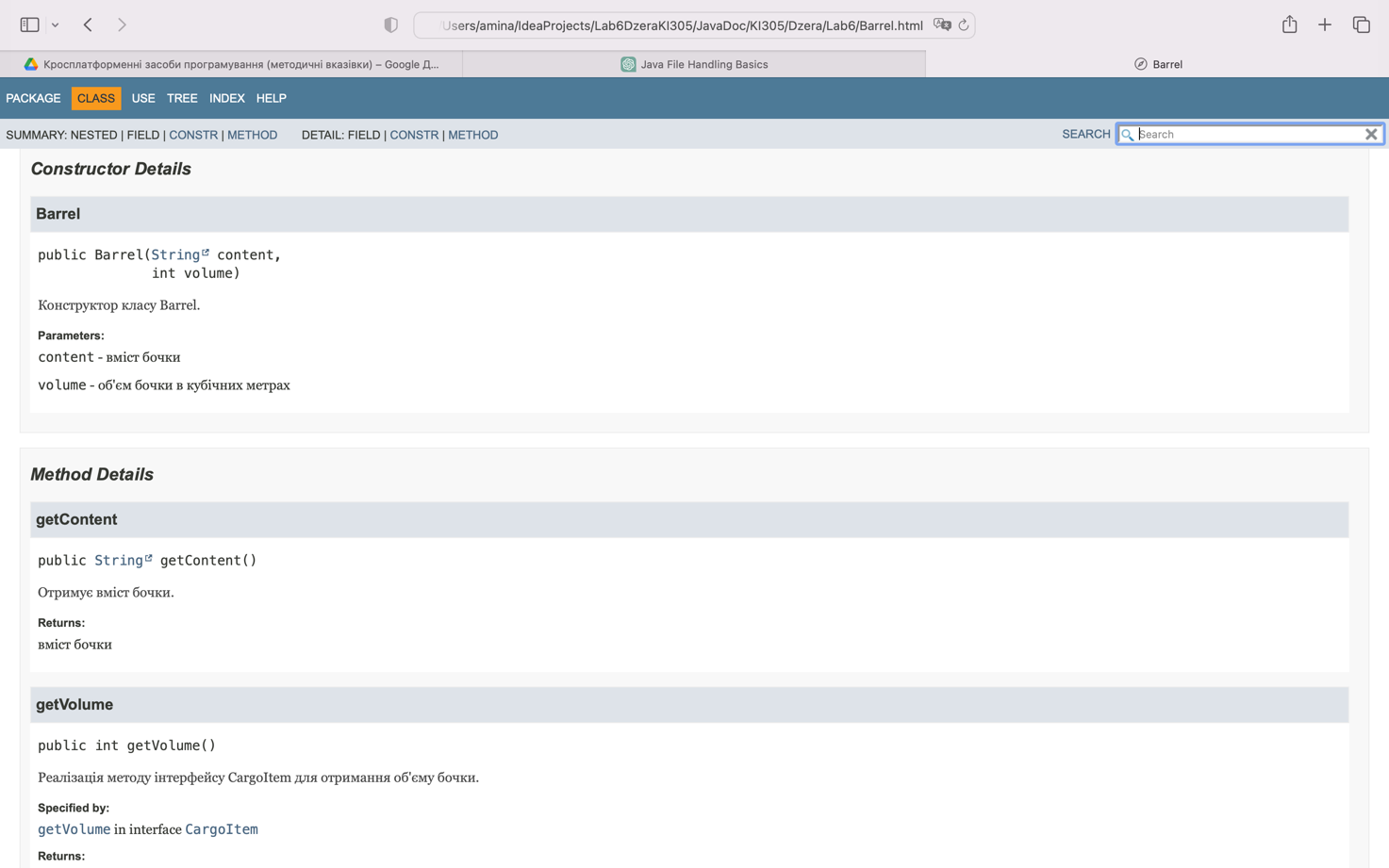
**Фрагмент згенерованої документації**

****

****





****

**Контрольні питання**

1. Дайте визначення терміну «параметризоване програмування».

*Параметризоване програмування* ⎯ це підхід в програмуванні, що дозволяє створювати загальні абстракції (такі як класи, функції або методи), які можуть працювати з будь-яким типом даних, або з декількома типами даних, без прив'язки до конкретного типу.

1. Розкрийте синтаксис визначення простого параметризованого класу.

class MyClass<T> {

// Код класу, де T - це параметр типу

}

1. Розкрийте синтаксис створення об’єкту параметризованого класу.

MyClass<int> obj = new MyClass<int>();

1. Розкрийте синтаксис визначення параметризованого методу.

void MyMethod<T>(T param) {

// Код методу, де T - це параметр типу

}

1. Розкрийте синтаксис виклику параметризованого методу.

MyMethod<int>(5); // Приклад виклику методу з типом даних int

1. Яку роль відіграє встановлення обмежень для змінних типів?

Роль встановлення обмежень для змінних типів ⎯ це забезпечення того, що параметр типу відповідає певним критеріям (наприклад, що він реалізує певний інтерфейс або має певний конструктор).

1. Як встановити обмеження для змінних типів?

Java обмеження для змінних типів можна встановити за допомогою ключового слова *extends* або *super* в контексті параметризованих класів і методів. Використання ключового слова extends дозволяє обмежити тип параметра відповідно до його успадкування, тоді як ключове слово super використовується для обмеження типу параметра його батьківським типом. Наприклад:

class MyClass<T extends SomeClass> {

// Код класу, де T - це параметр типу, який успадковує SomeClass

}

1. Розкрийте правила спадкування параметризованих типів.

Параметризовані класи можуть успадковувати інші класи, але при цьому потрібно передати параметри типу базовому класу.

1. Яке призначення підстановочних типів?

*Призначення підстановочних типів ⎯* це можливість використовувати параметризовані типи в загальних контекстах, не вказуючи конкретний тип даних.

1. Застосування підстановочних типів.

Застосування підстановочних типів допомагає підтримувати гнучкість і повторне використання коду, дозволяючи використовувати загальні класи та методи для різних типів даних, не змінюючи їхнього внутрішнього коду.

**Висновок**

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