Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра ЕОМ



**Звіт**

з лабораторної роботи №7

з дисципліни: «Кросплатформенні засоби програмування»

на тему: «Параметризоване програмування»

Виконав: ст.гр. КІ-34

Турчин П.Р.

Прийняв:

викл. каф. ЕОМ

Іванов Ю. С.

**Львів 2022**

**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 екземплярі розробленого класу-контейнеру. Програма має розміщуватися в пакеті Група.Прізвище.Lab7 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 фрагмент згенерованої документації.
4. Дати відповідь на контрольні запитання.

**Варіант 23**

**Хлівище**

**Лістинг програми:**

**Файл BarnApp.java**

*/\*\*  
 \* lab 7 package  
 \*/*package KI34.Turchyn.Lab7;  
  
*/\*\*  
 \* Trash can Application class implements main method for Barn  
 \* class possibilities demonstration  
 \** ***@author*** *Pavlo Turchyn  
 \** ***@version*** *1.0  
 \*/*public class BarnApp  
{  
 */\*\*  
 \** ***@param*** *args param  
 \*/* public static void main(String[] args) {  
 Barn <? super Data> Barn = new Barn <Data>();  
 Barn.AddData(new Animal("Pig", 1, 350));  
 Barn.AddData(new Grain("Wheat" , 198));  
 Barn.AddData(new Grain("Oat" , 68));  
 Barn.AddData(new Animal("Chicken" , 1, 5));  
 Data res = Barn.findMax();  
 System.*out*.print("The biggest data in barn is: \n");  
 res.print();  
 }  
}

**Файл Barn.java**

package KI34.Turchyn.Lab7;  
import java.util.\*;  
  
public class Barn <T extends Data>  
{  
 private ArrayList<T> arr;  
 */\*\*  
 \* Constructor  
 \*/* public Barn() {  
 arr = new ArrayList<T>();  
 }  
  
 */\*\*  
 \* Method simulates finding the largest information  
 \*/* public T findMax()  
 {  
 if (!arr.isEmpty())  
 {  
 T max = arr.get(0);  
 for (int i=1; i< arr.size(); i++)  
 {  
 if ( arr.get(i).compareTo(max) > 0 )  
 max = arr.get(i);  
 }  
 return max;  
 }  
 return null;  
 }  
  
 */\*\*  
 \* Method simulates adding data  
 \*/* public void AddData(T data)  
 {  
 arr.add(data);  
 System.*out*.print("Element added: ");  
 data.print();  
 }  
  
 */\*\*  
 \* Method simulates deleting data  
 \*/* public void DeleteData(int i)  
 {  
 arr.remove(i);  
 }  
}

**Файл Grain.java**

*/\*\*  
 \* lab 7 package  
 \*/*package KI34.Turchyn.Lab7;  
  
*/\*\*  
 \* Class <code>Grain</code> implements Data  
 \** ***@author*** *Pavlo Turchyn  
 \** ***@version*** *1.0  
 \*\*/*public class Grain implements Data {  
 private String GrainType;  
 private int weight;  
 */\*\*  
 \* Constructor  
 \** ***@param*** *grainType type of grain  
 \** ***@param*** *Weight the weight from all grain's type  
  
 \*/* public Grain(String grainType, int Weight) {  
 GrainType = grainType;  
 weight = Weight;  
  
 }  
 */\*\*  
 \* Method returns grain's type  
 \** ***@return*** *grain's type  
 \*/* public String getGrainType() {  
 return GrainType;  
 }  
 */\*\*  
 \* Method sets the grain's type  
 \** ***@param*** *grainType grain's type  
 \*/* public void setGrainType(String grainType) {  
 GrainType = grainType;  
 }  
 */\*\*  
 \* Method returns grain's weight  
 \** ***@return*** *grain's weight  
 \*/* public int getWeight() {  
 return weight;  
 }  
  
 */\*\*  
 \* Method sets the grain's weight  
 \** ***@param*** *weight grain's type  
 \*/* public void setWeight(int weight) {  
 weight = weight;  
 }  
  
 */\*\*  
 \* Method simulates comparing thing`s weight  
 \*/* public int compareTo(Data p) {  
 Integer j = weight;  
 return j.compareTo(p.getWeight());  
 }  
  
  
 */\*\*  
 \* Method simulates printing info about thing  
 \*/* public void print() {  
 System.*out*.print("Type of grain: " + GrainType + ", Grain Weight: " + weight + ";\n");  
 }  
  
  
}

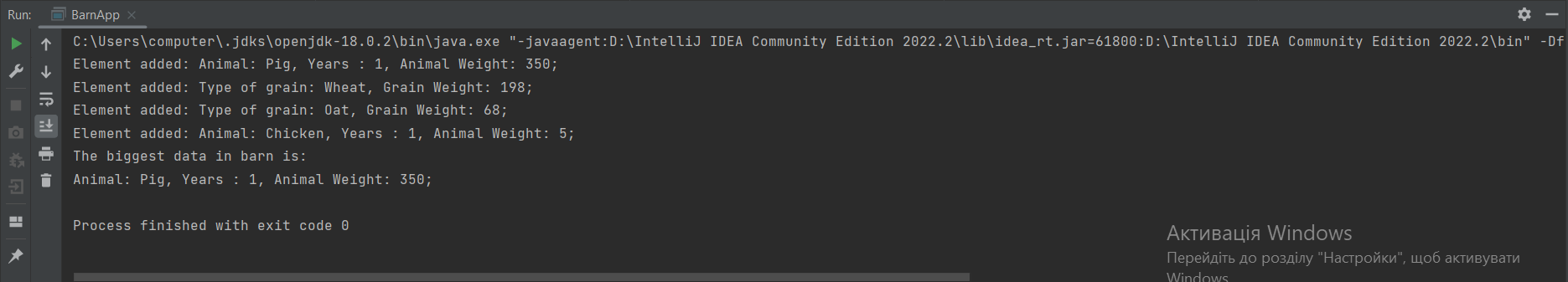
**Файл Animal.java**

*/\*\*  
 \* lab 7 package  
 \*/*package KI34.Turchyn.Lab7;  
*/\*\*  
 \* Class <code>Animal</code> implements Data  
 \** ***@author*** *Pavlo Turchyn  
 \** ***@version*** *1.0  
 \*\*/*import java.time.Year;  
  
public class Animal implements Data {  
 private String AnimalName;  
 private int years;  
 private int weight;  
  
 */\*\*  
 \* Constructor  
 \** ***@param*** *tAnimalName Name of Animal  
 \** ***@param*** *tYears Animal's age  
 \** ***@param*** *tWeight Animal's weight  
 \*/* public Animal(String tAnimalName, int tYears, int tWeight) {  
 AnimalName = tAnimalName;  
 years = tYears;  
 weight = tWeight;  
  
 }  
 */\*\*  
 \* Method returns animal's name  
 \** ***@return*** *animal's name  
 \*/* public String getAnimalName() {  
 return AnimalName;  
 }  
 */\*\*  
 \* Method sets the new animal's name  
 \** ***@param*** *animalName animal's name  
 \*/* public void setAnimalName(String animalName) {  
 AnimalName = animalName;  
 }  
 */\*\*  
 \* Method returns animal's age  
 \** ***@return*** *animal's age  
 \*/* public int getYears() {  
 return years;  
 }  
 */\*\*  
 \* Method sets the new animal's weight  
 \** ***@param*** *years animal's age  
 \*/* public void setYears(int years) {  
 this.years = years;  
 }  
 */\*\*  
 \* Method returns animal's weight  
 \** ***@return*** *animal's weight  
 \*/* public int getWeight() {  
 return weight;  
 }  
  
 */\*\*  
 \* Method simulates comparing animal's weight  
 \*/* public int compareTo(Data p) {  
 Integer s = weight;  
 return s.compareTo(p.getWeight());  
 }  
 */\*\*  
 \* Method simulates printing info about animal  
 \*/* public void print() {  
 System.*out*.print("Animal: " + AnimalName + ", Years : " + years +  
 ", Animal Weight: " + weight + ";\n");  
 }  
  
  
  
}

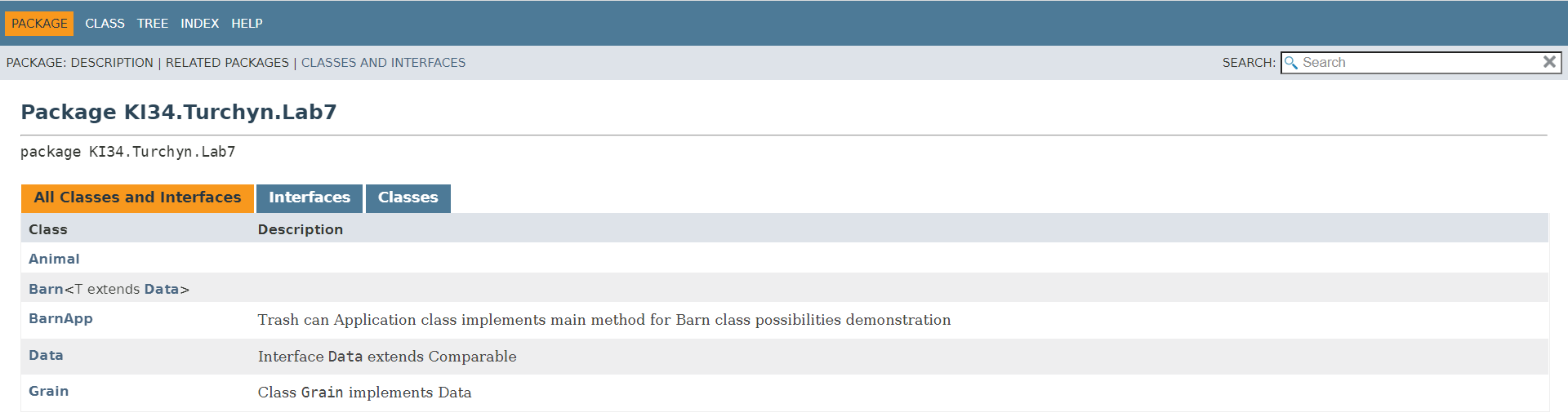
**Файл Data.java**

*/\*\*  
 \* lab 7 package  
 \*/*package KI34.Turchyn.Lab7;  
*/\*\*  
 \* Interface <code>Data</code> extends Comparable  
 \** ***@author*** *Pavlo Turchyn  
 \** ***@version*** *1.0  
 \*\*/*public interface Data extends Comparable<Data>{  
 public int getWeight();  
 public void print();  
  
}

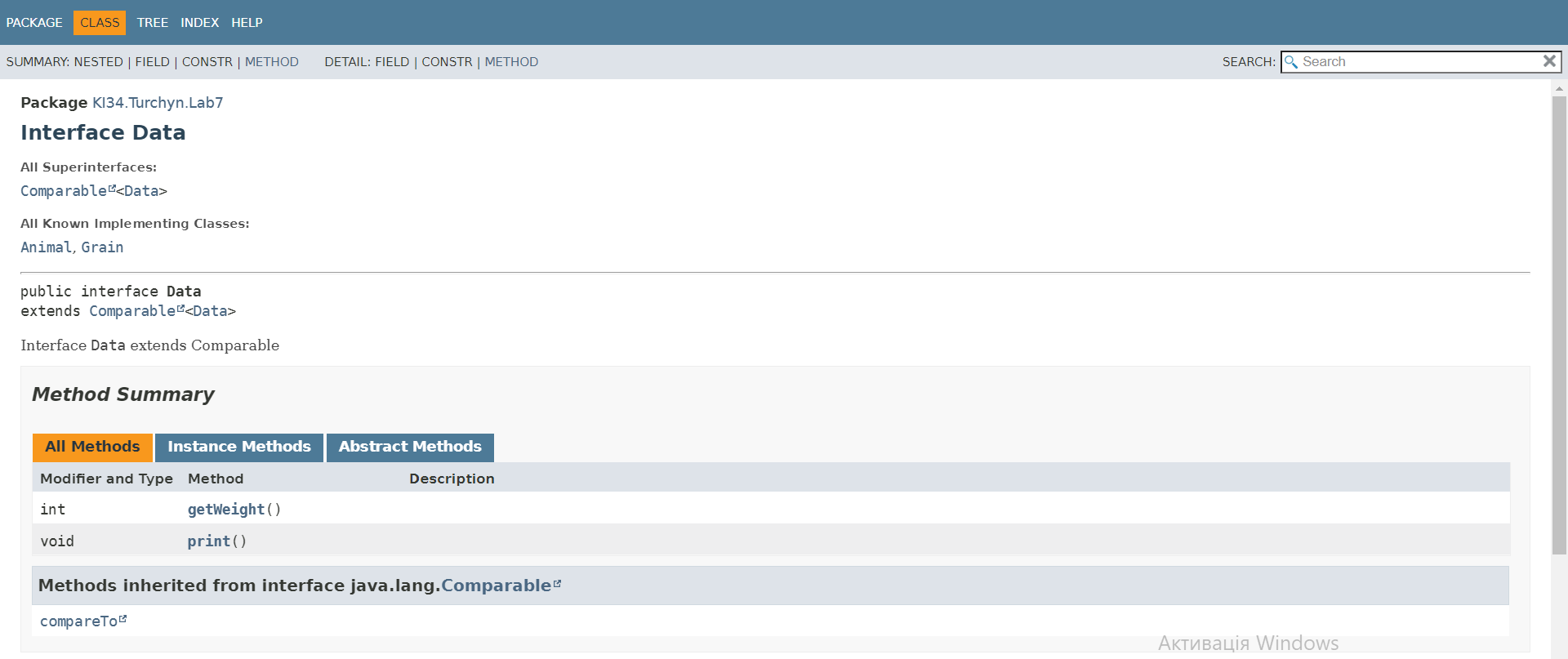
**Результат виконання програми:**

**

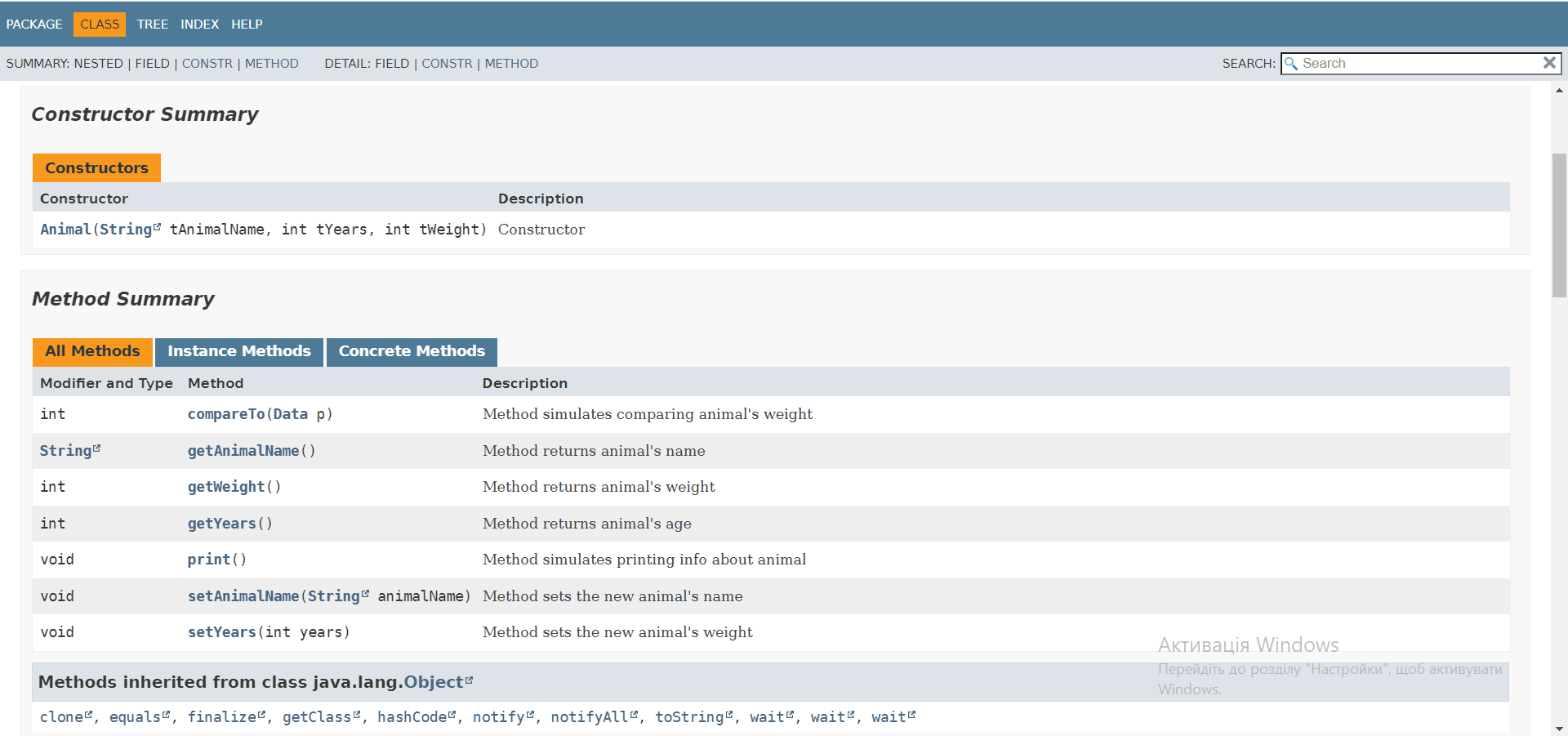
*Успішне виконання програми*

**

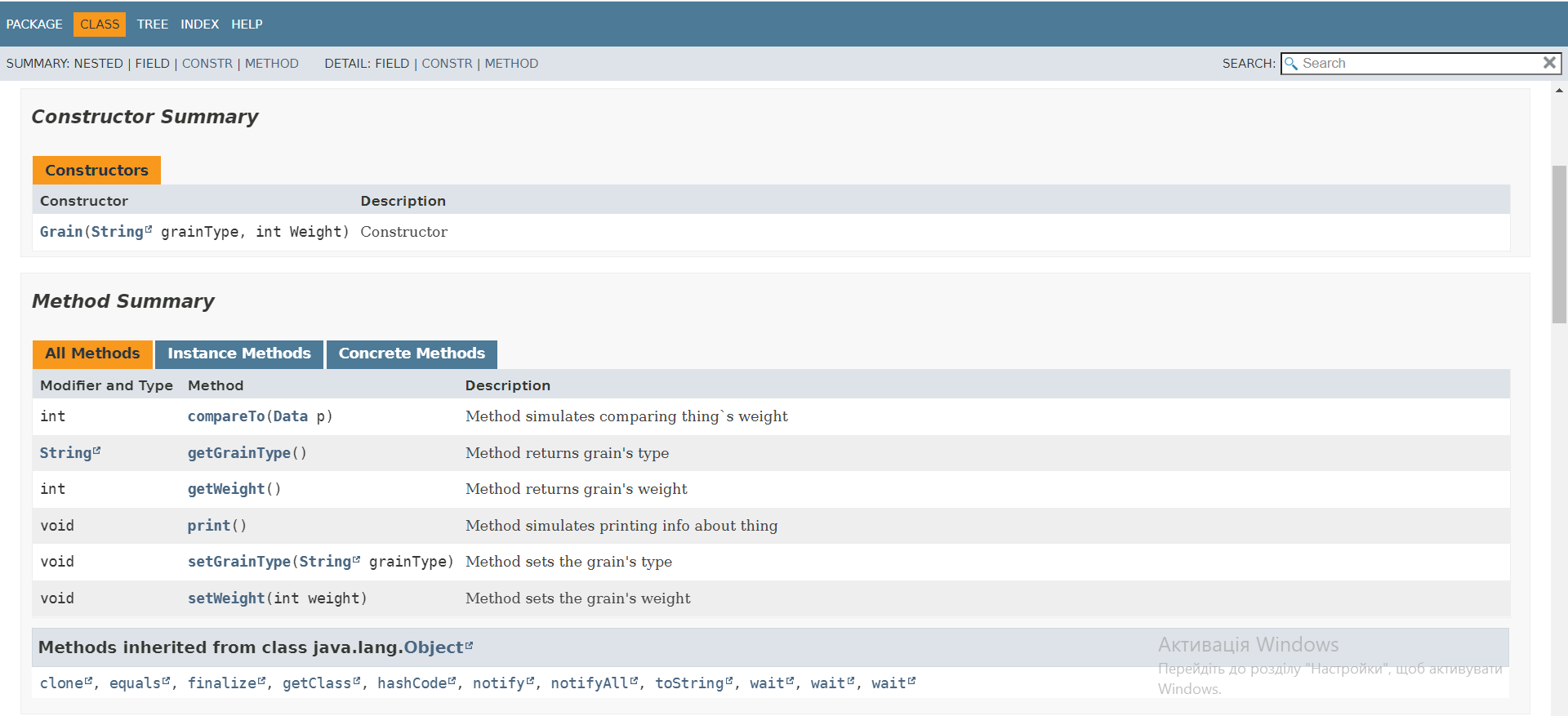
*Згенерована документація*

**

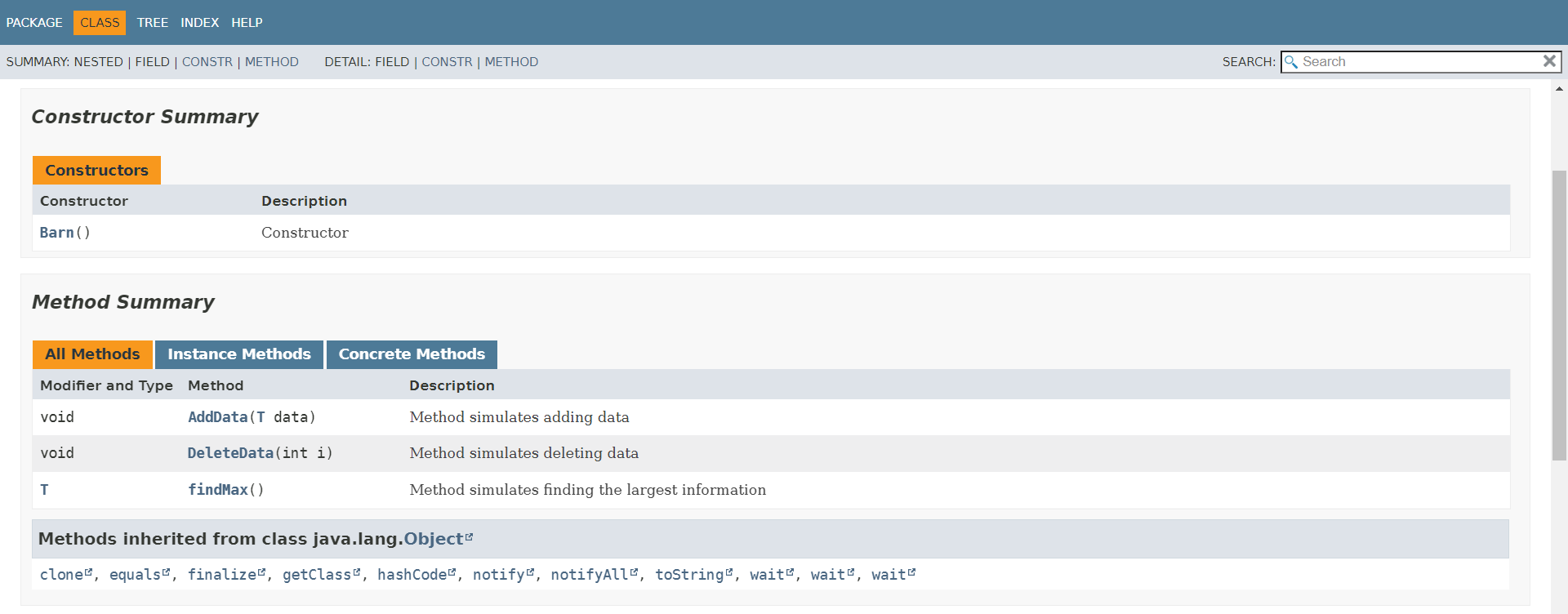
*Інформація про клас Data*

**

*Інформація про клас Animal*

**

*Інформація про клас Grain*

**

*Інформація про клас Barn*

**

*Інформація про клас BarnApp*

**Відповіді на 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.

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

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

Синтаксис оголошення параметризованого класу:

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

{…}

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

Синтаксис створення об’єкту параметризованого класу:

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

1. **Розкрийте синтаксис визначення параметризованого методу.**

Синтаксис оголошення параметризованого методу:

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

1. **Розкрийте синтаксис виклику параметризованого методу.**

Синтаксис виклику параметризованого методу:

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

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.

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

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.Класи DataOutputStream і DataInputStream дозволяють записувати і зчитувати дані примітивних типів.

1. **Розкрийте правила спадкування параметризованих типів.**

Правила спадкування параметризованих типів:

1. 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.
2. Завжди можна перетворити параметризований клас у «сирий» клас, при роботі з яким захист від некоректного коду є значно слабшим, що дозволяє здійснювати небезпечні присвоєння об’єктів параметризованого класу об’єктам «сирого» класу.
3. Параметризовані класи можуть розширювати або реалізовувати інші параметризовані класи. В цьому відношенні вони не відрізняються від звичайних класів.
4. **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.

1. **Застосування підстановочних типів.**

Підстановочні типи дозволяють реалізувати:

1. обмеження підтипу;
2. обмеження супертипу;
3. необмежені підстановки.

**Висновок:**

На цій лабораторній роботі я оволодів навиками параметризованого програмування мовою Java.