People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University M'hamed Bouguerra - Boumerdes
Institute of Electrical and Electronics Engineering
Electronics Department

Option: Computer engineering

LAB REPORT n°5

December 28, 2022

Title

EE423: Advanced Programming/ Working with Inheritance, Interfaces and Packages

---

Authored by :
Agli Wafa
Zidane Aymen

Instractor :
Dr.A Zitouni

Session : 2022/2023

Contents

Introduction

Within this lab, you will, in the first part, learn about the organization of classes by putting them into packages. Classes should be grouped and put together inside packages in a coherent manner and each package should be given a meaningful and representative name of the group. You will also understand inheritance by coding it. In the second part, using inheritance, you will see two cases of polymorphisme namely overloading and overriding methods.

Tools and Software:

1. A PC with ECLIPSE IDE V8.
2. Online LaTeX Editor for writing the report.

1 Part 1: *Create the following structure of classes*

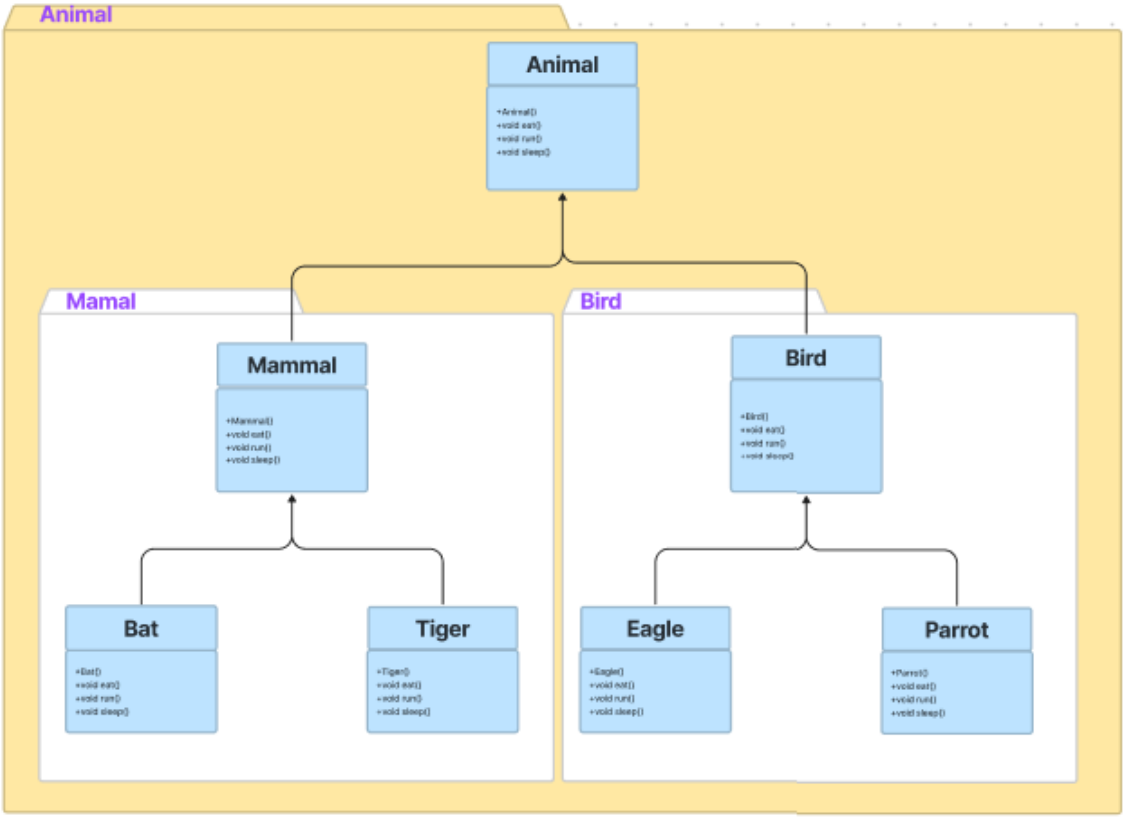

Figure 1: Corresponding class diagram

1.1 Classes design

- **Animal class**

```
1  public class Animal {
2    public Animal() { System.out.println("An animal is created"); }
```

```
    public void eat() {System.out.println("The animal is eating.");};
    public void run() {System.out.println("The animal is running.");};
    public void sleep() {System.out.println("The animal is sleeping.");};
    public static void main(String[] args){
    System.out.println("\n-------------------------------------\n");
    Mammal mammal = new Mammal();
    System.out.println("\n-------------------------------------\n");
    Parrot parrot = new Parrot();
    System.out.println("\n-------------------------------------\n");
    Tiger tiger= new Tiger();
    System.out.println("\n-------------------------------------\n");
    Animal animal = new Tiger();
    animal.sleep();
    System.out.println("\n-------------------------------------\n");
    Tiger tiger2 = new Tiger();
    Parrot parrot2 = new Parrot();
    tiger2.eat(parrot2);
    System.out.println("\n-------------------------------------\n");
    Tiger tiger3 = new Tiger();
    Parrot parrot3 = new Parrot();
    parrot3.run(tiger3);
    }
}

```

- **Mammal class**

```
    package animal.mammal;
    import animal.Animal;
    public class Mammal extends Animal {
    public Mammal() { System.out.println("A mammal is created"); }
    public void eat() {System.out.println("The mammal is eating.");}
    public void run() {System.out.println("The mammal is running.");}
    public void sleep() {System.out.println("The mammal is sleeping.");}
    }

```

- **Bird class**

```
   package animal.bird;
   import animal.Animal;
   public class Bird extends Animal {
   public Bird() { System.out.println("A bird is created"); }
   public void eat() {System.out.println("The Bird is eating.");}
   public void run() {System.out.println("The Bird is running.");}
   public void sleep() {System.out.println("The Bird is sleeping.");}
   }

```

- **Bat class**

```
  package animal.mammal;
  public class Bat extends Mammal {
  public Bat() { System.out.println("A bat is created"); }
  public void eat() {System.out.println("The bat is eating.");}
  public void run() {System.out.println("The bat is running.");}
```

```
  public void sleep() {System.out.println("The bat is sleeping.");}
  }

```

- **Tiger class**

```
    package animal.mammal;
    import animal.bird.Parrot;
    public class Tiger extends Mammal {
    public Tiger() { System.out.println("A tiger is created"); }
    public void eat() {System.out.println("The tiger is eating.");}
    public void eat(Parrot parrot) {
    parrot.sleep();
    System.out.println("A bad tiger just ate a parrot");
    }
    public void run() {System.out.println("The tiger is running.");}
    public void sleep() {System.out.println("The tiger is sleeping.");}
    }

```

- **Parrot class**

```
    package animal.bird;
    import animal.mammal.Tiger;
    public class Parrot extends Bird {
    public Parrot() { System.out.println("A parrot is created"); }
    public void eat() {System.out.println("The parrot is eating.");}
    public void run(){System.out.println("The parrot is running.");}
    public void run(Tiger tiger){
    System.out.println("A tiger! Run run run!");
    }
    public void sleep() {System.out.println("The parrot is sleeping.");}
    }

```

- **Eagle class**

```
    package animal.bird;
    public class Eagle extends Bird {
    public Eagle() { System.out.println("An eagle is created"); }
    public void eat() {System.out.println("The eagle is eating.");}
    public void run() {System.out.println("The eagle is running.");}
    public void sleep() {System.out.println("The eagle is sleeping.");}
    }

```

1.2 Results and Explanation

1.2.1

Running the code ***Mammal mammal = new Mammal();*** :

```
> An animal is created
> A mammal is created
```

Explanation: since the Mammal class extends Animal class, Java implicitly calls the super() constructor of Mammal which is the constructor of Animal (parent class). The super() is called as soon as the constructor of Mammal starts. I.e, before the System.out.println() line in Mammal constructor, Hence we get "An animal is created" first then "A mammal is created".

1.2.2

Running the code ***Parrot parrot = new Parrot();*** :

```
> An animal is created
> A bird is created
> A parrot is created
```

Explanation: since the Parrot class extends Bird class, Java will implicitly call the super() constructor before continuing to execute Parrot's constructor, And since the Bird class also extends Animal, then Java will do the same (call the super() first). So, the constructor of the Animal class will run first, producing the first line of output. Then, the Bird constructor will run second, hence we get the second line of output indicating a Bird was created. And finally, the Parrot constructor will continue executing last, which gives the last line out output.

1.2.3

Running the code ***Tiger tiger= new Tiger();*** :

```
> An animal is created
> A mammal is created
> A tiger is created
```

Explanation: the reason behind the output above is identical to that of 1.2.2, except that this time the class Tiger extends Mammal and Mammal extends Animal. So to construct a Tiger , the Animal constructor runs first, then Mammal's, and finally that of Tiger.

1.2.4

Running the code
Animal animal = new Tiger();
animal.sleep();:

```
> An animal is created
> A mammal is created
> A tiger is created
> The tiger is sleeping.
```

Explanation: This time we are creating a Tiger instance, but we are storing it in an Animal variable. This is possible in Java since the Tiger is a child of Animal. However, using the 'animal' variable is only going to allows us to access the variables and functions defined in the Animal class. The first 3 lines of output are explained in 3.1.3.
For the last line of output. Since the function sleep() is overriden by the Tiger class. and the 'animal' variable is referring to a Tiger instance. the sleep() function that is going to be executed is the overriden one. Hence, we get "The tiger is sleeping".

1.2.5

Overloading the "eat()" function as described in the lab sheet and as shown in the code above and running the code:
Tiger tiger = new Tiger();
Parrot parrot = new Parrot();
tiger.eat(parrot);

```
> An animal is created
> A mammal is created
> A tiger is created
> An animal is created
> A bird is created
> A parrot is created
> The parrot is sleeping.
> A bad tiger just ate a parrot
```

Explanation:the explanations for the different objects being created are stated in 3.1.2 and 3.1.3. However, we notice that when we pass the parrot instance to the 'eat(Parrot parrot)' function, no new Parrot gets created. Or at least the constructor of that class does not get called, even if we know that Java passes by value and not by reference. This is due to the fact that Java uses another constructor when passing instances through methods, and that constructor is called the Copy Constructor. So that is why we do not observe that a new parrot getting created through the constructor that we defined earlier. The last two lines of output are expected, since the eat() first calls parrot.sleep() which produces the 7th line of output, then the print() call executes producing the last line of output.

1.2.6

Overloading the "eat()" function as described in the lab sheet and as shown in the code above and running the code:
Tiger tiger = new Tiger();
Parrot parrot = new Parrot();
parrot.run(tiger);

```
> An animal is created
> A mammal is created
> A tiger is created
> An animal is created
> A bird is created
> A parrot is created
> A tiger! Run run run!
```

Explanation: the explanation of this part is identical to the previous one. Except that this time, the tiger instance is the one that is passed as a parameter to a Parrot function instead of the reverse.

2 Part 2: *Overloading and overriding methods*

2.1 class Point:

```
1     public class Point {
2     private int x;
3     private int y;
4
5     public Point(int x, int y) { this.x = x; this.y = y;}
6     public void setX(int x) {this.x = x;}
7     public void setY(int y) {this.y = y;}
8     public int getX() {return this.x;}
9     public int getY() {return this.y;}
10    public void print(){System.out.println("Point("+x+","+y+")");}
11
12    public void print(boolean pretty){
13        if(pretty) print();
14        else System.out.println(x+","+y);
15    }
16
17    public static void main(String[] args) {
18        Point pt = new Point(102, 210);
19        pt.print();
20        pt.print(true);
21        pt.print(false);
22        ColoredPoint cpt = new ColoredPoint(103, 330, "black");
23        cpt.print();
24        cpt.print(true);
25        cpt.print(false);
26        ColoredPoint cpt2 = new ColoredPoint();
27        cpt2.print();
28        cpt2.print(true);
29        cpt2.print(false);
30    }
31 }
32
33
```

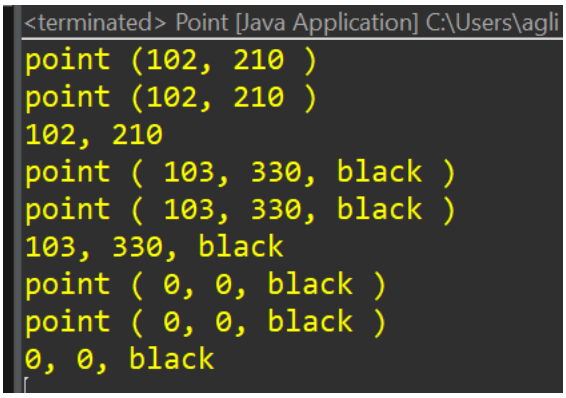

Figure 2: the results of Class Point

2.2 Extending the class Point

```
class ColoredPoint extends Point {
    private String color;

    ColoredPoint(int x, int y, String color) {
        super(x, y);
        this.color = color;
    }

    ColoredPoint() {
        this(0, 0, "black");
    }

    public void print(){
        System.out.println("Point("+getX()+","+getY()+","+color+")");
    }

    public void print(boolean pretty){
        if(pretty) print();
        else System.out.println(getX()+","+getX()+","+color);
    }
}
```

3 Conclusion

1. When creating a new class in Java, you can define the class to inherit from an existing class.
2. Inheritance is the process by which the new child subclass automatically includes any public or protected primitives, objects, or methods defined in the parent class.
3. Java supports single inheritance, by which a class may inherit from only one direct parent class.
4. Java also supports multiple levels of inheritance, by which one class may extend another class, which in turn extends another class.