SWE 4201 - OBJECT ORIENTED CONCEPTS I

LAB 05 HANDOUT AND TASKS

Prepared by:
Md. Jubair Ibna Mostafa
Assistant Professor, IUT CSE

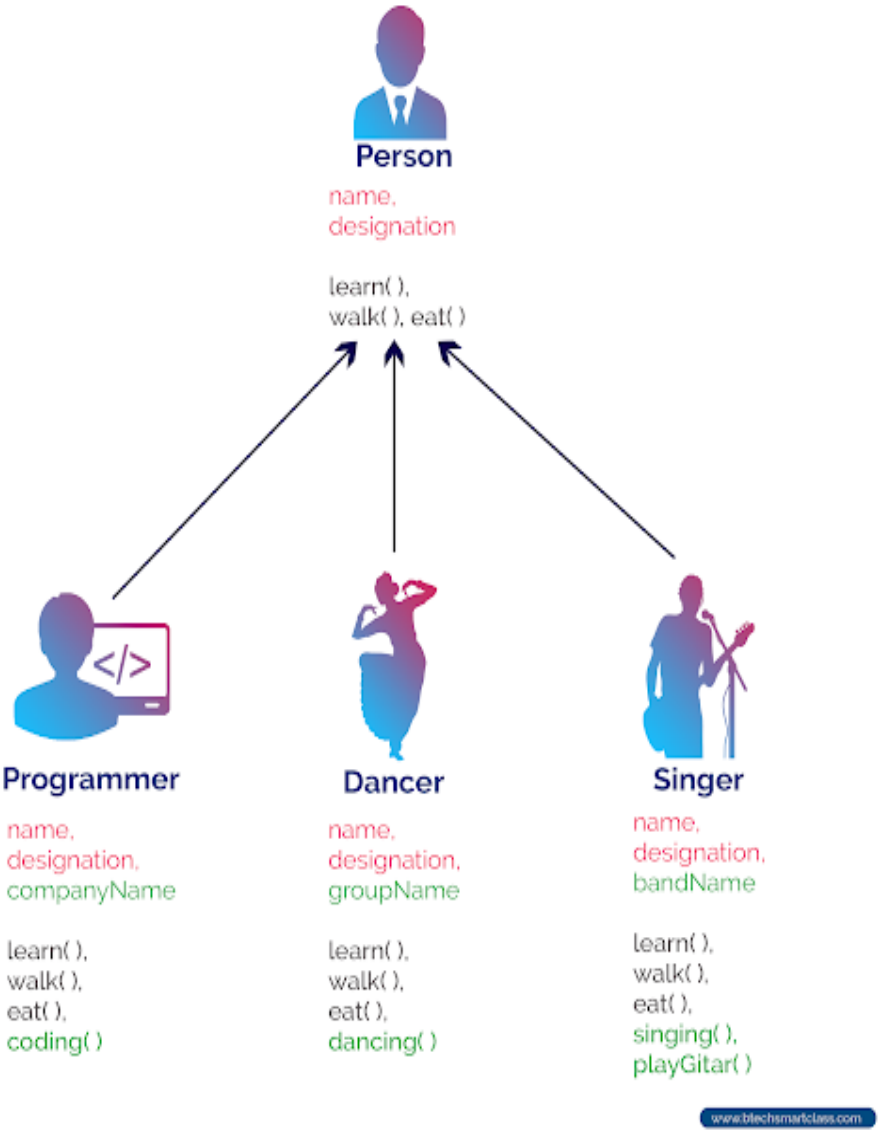
Department of Computer Science and Engineering
Islamic University of Technology

Contents

1 INHERITANCE

Inheritance is one of the fundamental concepts of Object-Oriented Programming (OOP). It enables us to reuse existing code or extend an existing type. Inheritance allows us to create a base class with basic features and behaviors and derive specialized versions by creating classes that inherit from this base class. A base type is also called a super or a parent type. A derived type is referred to as an extended, sub, or child type.

Figure 1: Inheritance



Example of Inheritance

```
class Car {
    int wheels;
    String model;
    Car(int wheels, String model){
        this.wheels = wheels;
        this.model = model;
    }
    void start() {
         System.out.println(model + " car moving\n");
    }
}

class ArmoredCar extends Car {
    int bulletProofWindows;
    ArmoredCar(int wheels, String model){
        super(wheels, model);
    }

    ArmoredCar(int wheels, String model, int bulletProofWindows){
        super(wheels, model);
        this.bulletProofWindows = bulletProofWindows;
    }

    void remoteStartCar() {
        System.out.println(super.model + " car stated remotely");
    }
}

class Main {
    public static void main(String[] args) {
        Car car = new Car(4, "BMW");
        car.start();

        ArmoredCar arCar = new ArmoredCar(4, "U.S. T17E1 Staghound", 2)
```

```
    ;
35          arCar.remoteStartCar();
36      }
37  }
```

Listing 1: An Example of Inheritance with constructor

In Listing 1 Car is the base or parent class and ArmoredCar class is the sub or child class.

In Java, a class can inherit only another class. Similarly, an interface can inherit other interfaces (not a single interface, multiple is also allowed).

Example of Interface Inheritance

```
1  public interface Floatable {
2      void floatOnWater();
3  }
4
5  public interface Flyable {
6      void fly();
7  }
8
9  public class ArmoredCar extends Car implements Floatable, Flyable{
10     public void floatOnWater() {
11         System.out.println("I can float!");
12     }
13
14     public void fly() {
15         System.out.println("I can fly!");
16     }
17 }
```

Listing 2: An Example of interface inheritance

Another Example of Interface Inheritance

```
1  interface Drawable {
2      void draw();
3  }
4
```

```
5  interface Interactable {
6      void onClick();
7  }
8
9  interface Component extends Drawable, Interactable {
10     void setPosition(int x, int y);
11 }
12
13 class Button implements Component {
14     private int x, y;
15     private String name;
16
17     public Button(String name) {
18         this.name = name;
19     }
20
21     public void draw() {
22         System.out.println("Button '" + name + "' is being drawn at
       position (" + x + ", " + y + ")");
23     }
24
25     public void onClick() {
26         System.out.println("Button '" + name + "' was clicked");
27     }
28
29     public void setPosition(int x, int y) {
30         this.x = x;
31         this.y = y;
32     }
33
34     public String getName() {
35         return name;
36     }
37 }
38
```

```java
public class Main {
    public static void main(String[] args) {
        Button button = new Button("Submit");
        button.setPosition(100, 50);
        button.draw();
        button.onClick();
    }
}
```

Listing 3: Another Example of interface inheritance

1.1 Benefits of Inheritance

- **Code Reusability**: Inheritance allows classes to inherit attributes and methods from a parent class, reducing the need to rewrite code.
- **Extensibility**: Inheritance allows new classes to be derived from existing ones, adding new features or modifying existing behavior without modifying the original class.
- **Modularity**: Inheritance promotes modularity by breaking down complex systems into smaller, more manageable classes that can be reused in different contexts.
- **Polymorphism**: Inheritance enables polymorphic behavior, where a subclass can be used wherever its superclass is accepted, providing flexibility and extensibility in the code.

2 TASKS

2.1 Scenario 1 Marks 10

In a zoo management system, there is a need to model different types of animals, including birds and aquatic animals. Each animal has basic information such as name, species, and age. Birds have additional attributes like wingspan and wing color. while aquatic animals have attributes like fin length, and bone type (light or spongy). All animals can eat, Birds can

fly, and build a nest, and aquatic animals can swim. So, there is a need to model the concepts of fly ability and float ability for certain animals.

Task 1: Implement a class hierarchy for animals ensuring proper encapsulation concepts.

Task 2: Aquatic animals could be Fish, Whale or Seal. Some of these animals live in salty water while others live in fresh water. Some eat zooplankton while others eat other animals. Extend the previous hierarchy to support these relations.

2.2 Scenario 2 Marks 5

You have to model various shapes for drawing purposes. Each shape has common properties such as name and, color along with some methods of area and perimeter. Shapes include circles and rectangles, each with specific properties such as radius for circles and length and width for rectangles. Any shape is drawable. Use inheritance concepts to implement the solution.

3 REFERENCE

- https://www.baeldung.com/java-inheritance