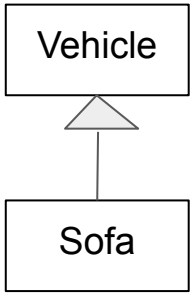
Introduction to Programming I

Lab 7

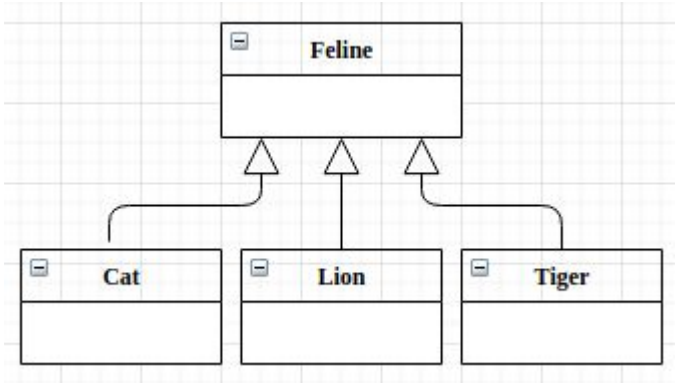
Alexey Shikulin, Munir Makhmutov, Sami Sellami and Furqan Haider

Inheritance and Polymorphism

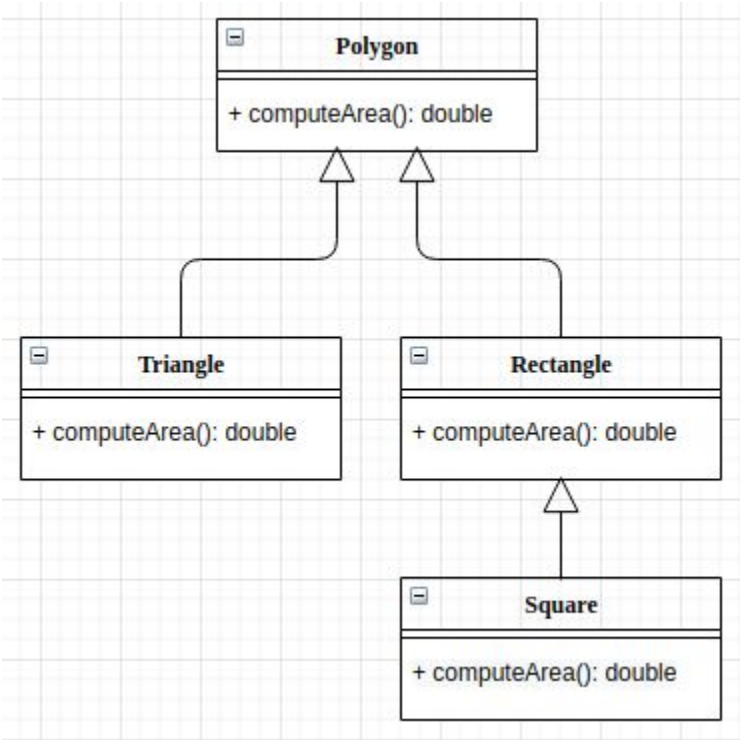


Inheritance

When one object acquires all the properties and behaviors of another object, it is known as inheritance.



Polymorphism



If one task is performed in different ways, it is known as polymorphism.

Abstraction

Data Abstraction is defined as the process of reducing the object to its essential details so that only the necessary characteristics are exposed to the users.

For example phone call, we don't know the internal processing.

Encapsulation

Encapsulation is defined as the wrapping up of data under a single unit. It is a protective shield that prevents the data from being accessed by the code outside this shield.

A java class is the example of encapsulation.

Encapsulation vs Abstraction

1. Encapsulation is data hiding (information hiding) while Abstraction is detailed hiding (implementation hiding).

2. While encapsulation groups together data and methods that act upon the data, data abstraction deal with exposing the interface to the user and hiding the details of implementation.

Interface

The interface is a blueprint that can be used to implement a class. All the methods of an interface are abstract methods.

An interface cannot be instantiated. However, classes that implement interfaces can be instantiated. Interfaces never contain instance variables but, they can contain public static final variables (i.e., constant class variables).

Exercise 1

1) Create a class which represents Animal class and its basic properties: height, weight, color, and basic operations: eat, sleep, animalSound. Also create child classes which represent the exact animals: cow, cat, dog and override properties / methods.
2) Implement classes for different shapes: Circle, Rectangle, Triangle, Square, Ellipse. Add corresponding members and methods to calculate the area and perimeter of the shapes.

Note: Use inheritance for minimizing amount of code

Solution 1.1: Animal.java

```java
package ru.makhmutov.task1.animals;

public class Animal {

    protected float height;
    protected float weight;
    protected String color;

    /**
     * This is a constructor of a class
     *
     * @param height the height of an animal
     * @param weight the weight of an animal
     * @param color the color of an animal
     */
    public Animal(float height, float weight, String color) {
        this.height = height;
        this.weight = weight;
        this.color = color;
    }

    /**
     * This method allows an animal to eat
     */
    protected void eat(){
        System.out.println("Animal is eating");
    }

    /**
     * This method allows an animal to sleep
     */
    protected void sleep(){
        System.out.println("Animal is sleeping");
    }

    /**
     * This method allows an animal to talk
     */
    protected void animalSound(){
        System.out.println("Animal is talking");
    }

    @Override
    public String toString() {
        return this.getClass().getSimpleName() + "{" +
                "height=" + height +
                ", weight=" + weight +
                ", color='" + color + '\'' +
                '}';
    }
}
```

Solution 1.1: Cat.java

```java
package ru.makhmutov.task1.animals;

public class Cat extends Animal {

    /**
     * This is a constructor of a class
     *
     * @param height the height of an animal
     * @param weight the weight of an animal
     * @param color  the color of an animal
     */
    public Cat(float height, float weight, String color) {
        super(height, weight, color);
    }

    /**
     * This method allows an animal to eat
     */
    @Override
    protected void eat() {
        System.out.println("The Cat is eating");
    }

    /**
     * This method allows an animal to sleep
     */
    @Override
    protected void sleep() {
        System.out.println("The Cat is sleeping");
    }

    /**
     * This method allows an animal to talk
     */
    @Override
    protected void animalSound() {
        System.out.println("The Cat is meowing");
    }
}
```

Solution 1.1: Cow.java

```java
package ru.makhmutov.task1.animals;

public class Cow extends Animal {

    /**
     * This is a constructor of a class
     *
     * @param height the height of an animal
     * @param weight the weight of an animal
     * @param color  the color of an animal
     */
    public Cow(float height, float weight, String color) {
        super(height, weight, color);
    }

    /**
     * This method allows an animal to eat
     */
    @Override
    protected void eat() {
        System.out.println("The " + this.getClass().getSimpleName() + " is eating");
    }

    /**
     * This method allows an animal to sleep
     */
    @Override
    protected void sleep() {
        System.out.println("The " + this.getClass().getSimpleName() + " is sleeping");
    }

    /**
     * This method allows an animal to talk
     */
    @Override
    protected void animalSound() {
        System.out.println("The " + this.getClass().getSimpleName() + " is humming");
    }
}
```

Solution 1.1: Dog.java

```java
package ru.makhmutov.task1.animals;

public class Dog extends Animal {

    /**
     * This is a constructor of a class
     *
     * @param height the height of an animal
     * @param weight the weight of an animal
     * @param color  the color of an animal
     */
    public Dog(float height, float weight, String color) {
        super(height, weight, color);
    }

    /**
     * This method allows an animal to eat
     */
    @Override
    protected void eat() {
        System.out.println("The " + this.getClass().getSimpleName() + " is eating");
    }

    /**
     * This method allows an animal to sleep
     */
    @Override
    protected void sleep() {
        System.out.println("The " + this.getClass().getSimpleName() + " is sleeping");
    }

    /**
     * This method allows an animal to talk
     */
    @Override
    protected void animalSound() {
        System.out.println("The " + this.getClass().getSimpleName() + " is barking");
    }
}
```

Solution 1.1: AnimalCreator.java

```java
package ru.makhmutov.task1.animals;

import java.util.Arrays;

public class AnimalCreator {

    /**
     * The entry point of the Animals program.
     * It allows creating array of different animals
     * and demonstrate inheritance and polymorphism
     * principles
     *
     * @param args Array with parameters of the program
     */
    public static void main(String[] args) {
        Animal[] animals = addAnimals();
        System.out.println(Arrays.toString(animals));
    }

    /**
     * This method allows adding animals of different
     * type to the array
     *
     * @return the array of animals
     */
    private static Animal[] addAnimals() {
        Animal[] animals = new Animal[4];

        Animal animal = new Animal(50, 35.5F, "black");
        animals[0] = animal;

        Animal dog = new Dog(40, 28F, "brown");
        animals[1] = dog;

        Animal cat = new Cat(10, 5.5F, "orange");
        animals[2] = cat;

        Animal cow = new Cow(160.5F, 450, "white");
        animals[3] = cow;

        return animals;
    }
}
```

Solution 1.2: Shape.java

```java
package ru.makhmutov.task1.shapes;

public abstract class Shape {
    protected abstract double calculatePerimeter();
    protected abstract double calculateArea();
}
```

Solution 1.2: Circle.java

```java
package ru.makhmutov.task1.shapes;

public class Circle extends Shape {

    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    @Override
    protected double calculatePerimeter() {
        return 2 * radius * Math.PI;
    }

    @Override
    protected double calculateArea() {
        return Math.PI * Math.pow(radius, 2);
    }
}
```

Solution 1.2: Rectangle.java

```java
package ru.makhmutov.task1.shapes;

public class Rectangle extends Shape {

    private double firstSideLength;
    private double secondSideLength;

    public Rectangle(double firstSideLength, double secondSideLength) {
        this.firstSideLength = firstSideLength;
        this.secondSideLength = secondSideLength;
    }

    @Override
    protected double calculatePerimeter() {
        return firstSideLength * 2 + secondSideLength * 2;
    }

    @Override
    protected double calculateArea() {
        return firstSideLength * secondSideLength;
    }
}
```

Solution 1.2: Triangle.java

```java
package ru.makhmutov.task1.shapes;

public class Triangle extends Shape {

    private double firstSideLength;
    private double secondSideLength;
    private double thirdSideLength;
    private double angleBetweenFirstAndSecondSides; // in degrees

    public Triangle(double firstSideLength, double secondSideLength,
                    double thirdSideLength, double angleBetweenFirstAndSecondSides) {
        this.firstSideLength = firstSideLength;
        this.secondSideLength = secondSideLength;
        this.thirdSideLength = thirdSideLength;
        this.angleBetweenFirstAndSecondSides = angleBetweenFirstAndSecondSides;
    }

    @Override
    protected double calculatePerimeter() {
        return firstSideLength + secondSideLength + thirdSideLength;
    }

    @Override
    protected double calculateArea() {
        return 0.5 * firstSideLength * secondSideLength
                * Math.sin(Math.toRadians(angleBetweenFirstAndSecondSides));
    }
}
```

Solution 1.2: Square.java

```java
package ru.makhmutov.task1.shapes;

public class Square extends Shape {

    private double sideLength;

    public Square(double sideLength) {
        this.sideLength = sideLength;
    }

    @Override
    protected double calculatePerimeter() {
        return sideLength * 4;
    }

    @Override
    protected double calculateArea() {
        return Math.pow(sideLength, 2);
    }
}
```

Solution 1.2: Ellipse.java

```java
package ru.makhmutov.task1.shapes;

public class Ellipse extends Shape {

    private double majorRadius;
    private double minorRadius;

    public Ellipse(double majorRadius, double minorRadius) {
        this.majorRadius = majorRadius;
        this.minorRadius = minorRadius;
    }

    @Override
    protected double calculatePerimeter() {
        return Math.PI * 2 * Math.sqrt((Math.pow(majorRadius, 2) + Math.pow(minorRadius, 2)) / 2);
    }

    @Override
    protected double calculateArea() {
        return majorRadius * minorRadius * Math.PI;
    }
}
```

Solution 1.2: ShapeCreator.java

```java
package ru.makhmutov.task1.shapes;

public class ShapeCreator {

    /**
     * The entry point of the Shapes program.
     * It allows creating array of different shapes
     * and demonstrate inheritance and polymorphism
     * principles
     *
     * @param args Array with parameters of the program
     */
    public static void main(String[] args) {
        Shape[] shapes = addShapes();
        displayShapeSizes(shapes);
    }

    /**
     * This method allows calculating perimeters and
     * areas of all array shapes
     * @param shapes the array of shapes
     */
    private static void displayShapeSizes(Shape[] shapes) {
        for (Shape shape : shapes) {
            System.out.println(shape.getClass().getSimpleName() +
                    " area is " + shape.calculateArea() + ", " +
                    shape.getClass().getSimpleName() +
                    " perimeter is " + shape.calculatePerimeter());
        }
    }

    /**
     * This method allows adding shapes of different
     * types into the array
     *
     * @return the array of shapes
     */
    private static Shape[] addShapes() {
        Shape[] shapes = new Shape[5];

        Shape circle = new Circle(5);
        shapes[0] = circle;

        Shape square = new Square(3);
        shapes[1] = square;

        Shape rectangle = new Rectangle(5, 2);
        shapes[2] = rectangle;

        Shape ellipse = new Ellipse(3, 6);
        shapes[3] = ellipse;

        Shape triangle = new Triangle(2, 2, 2, 60);
        shapes[4] = triangle;

        return shapes;
    }
}
```

Exercise 2

- Create the abstract class *Creature* with abstract methods *bear()* and *die()* and String field *name* equal to *null* and boolean *isAlive* equal to *false*. Also, create non-abstract method *shoutName()*, which should print the name, if it's not equal to null. Otherwise, it should print error message
- Create classes *Human*, *Dog* and *Alien* which should inherit the *Creature*. Override all abstract methods for all 3 classes differently
- For *bear()* method each of them should assign the *name* and print the message "*[class name]* called *[name]* has born"
- For *die()* method each of them should print the message "*[class name]* called *[name]* has died"
- Add a method *bark()* to a class *Dog*

Exercise 2 (cont.)

- Create the *AbstractClassDemonstration* class, to demonstrate the functionality
- Modify Exercise 2 *AbstractClassDemonstration* class, so that array of creatures of different types (*Human, Dog, Alien*) is created. For each element of the array call methods *bear()* and *die().*

Hint: you can use ArrayList instead of array

Exercise 2 solution: Creature.java

```java
package ru.makhmutov.task2;

public abstract class Creature {

    private String name = null;
    private boolean isAlive = false;

    public abstract void bear();

    public abstract void die();

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public boolean isAlive() {
        return isAlive;
    }

    public void setAlive(boolean alive) {
        isAlive = alive;
    }

    /**
     * This method prints out the name of a Creature
     */
    void shoutName() {
        if (name != null) {
            System.out.println("My name is " + name);
        } else {
            System.out.println("Error: the name is not assigned");
            System.exit(1);
        }
    }
}
```

Exercise 2 solution: Human.java

```java
package ru.makhmutov.task2;

public class Human extends Creature {

    @Override
    public void bear() {
        if (!this.isAlive()) {
            this.setAlive(true);
            this.setName("John");
            System.out.println("The " + this.getClass().getSimpleName() +
                    " " + getName() + " was born");
        }
    }

    @Override
    public void die() {
        if (this.isAlive()) {
            this.setAlive(false);
            System.out.println("The " + this.getClass().getSimpleName() +
                    " " + getName() + " has died. RIP");
            this.setName(null);
        }
    }
}
```

Exercise 2 solution: Dog.java

```java
package ru.makhmutov.task2;

public class Dog extends Creature {

    @Override
    public void bear() {
        if (!this.isAlive()) {
            this.setAlive(true);
            this.setName("Spike");
            System.out.println("The " + this.getClass().getSimpleName() +
                    " " + getName() + " was born");
        }
    }

    @Override
    public void die() {
        if (this.isAlive()) {
            this.setAlive(false);
            System.out.println("The " + this.getClass().getSimpleName() +
                    " " + getName() + " has barked and died");
            this.setName(null);
        }
    }

    public void bark() {
        System.out.println(this.getName() + " is barking");
    }
}
```

Exercise 2 solution: Alien.java

```java
package ru.makhmutov.task2;

public class Alien extends Creature {

    @Override
    public void bear() {
        this.setAlive(true);
        this.setName("Xenomorph");
        System.out.println("The " + this.getClass().getSimpleName() +
                " " + getName() + " has hatched from an egg. Beware of it!!!");
    }

    @Override
    public void die() {
        if (this.isAlive()) {
            this.setAlive(false);
            System.out.println("The " + this.getClass().getSimpleName() +
                    " " + getName() + " was killed by Sergeant Ripley");
            this.setName(null);
        }
    }
}
```

Exercise 2 solution: AbstractClassDemonstration.java

```java
package ru.makhmutov.task2;

public class AbstractClassDemonstration {

    /**
     * The entry point of the Creatures program.
     * Iw allows creation of creatures and showing
     * their results of overridden methods bear(),
     * shoutName() and die()
     *
     * @param args Array with parameters of the program
     */
    public static void main(String[] args) {
        Creature[] creatures = addCreatures();
        runCreatureMethods(creatures);
    }

    /**
     * This method allows seeing how creatures are born,
     * shout their names and die
     *
     * @param creatures the array of creatures
     */
    private static void runCreatureMethods(Creature[] creatures) {
        for (Creature creature : creatures) {
            creature.bear();
            creature.shoutName();
            creature.die();
            System.out.println();
        }
    }

    /**
     * This method allows creating creatures of different types
     * and adding them to the array
     *
     * @return the array with creatures
     */
    private static Creature[] addCreatures() {
        Creature[] creatures = new Creature[3];

        Creature human = new Human();
        creatures[0] = human;

        Creature dog = new Dog();
        creatures[1] = dog;

        Creature alien = new Alien();
        creatures[2] = alien;

        return creatures;
    }
}
```

Exercise 3

- Create an interface *Swimmable* with methods *swim()* and *stopSwimming()*
- Create an interface *Flyable* with methods *fly()* and *stopFlying()*
- Create an interface *Living* with default method *live()* that prints "*[class name]* lives"
- Create class *Submarine* which implements *Swimmable* and override methods
- Create class *Duck* which implements *Swimmable, Flyable* and *Living,* and override non-default methods
- Create class *Penguin* which implements *Swimmable* and *Living,* and override non-default methods
- Create the *InterfaceDemonstration* class, to demonstrate the functionality.

Hint: to stop swimming/flying creature has to be swimming/flying

Exercise 3 (cont.)

- Modify Exercise 3 *InterfaceDemonstration* class, so that array of living objects of different types (*Duck, Penguin*) is created. For each element of the array call method *live().*

Discussion

- What should happen if *swim()* is called for the elements of this array?
- Can instance of a *Submarine* be added to this array?

Exercise 3 solution: Swimmable.java

```
package ru.makhmutov.task3.interfaces;

public interface Swimmable {

    boolean isSwimming = false;

    void swim();

    void stopSwimming();
}
```

Exercise 3 solution: Flyable.java

```java
package ru.makhmutov.task3.interfaces;

public interface Flyable {

    void fly();

    void stopFlying();
}
```

Exercise 3 solution: Living.java

```java
package ru.makhmutov.task3.interfaces;

public interface Living {

    default void live() {
        System.out.println(this.getClass().getSimpleName() + " lives");
    }
}
```

Exercise 3 solution: Submarine.java

```java
package ru.makhmutov.task3.objects;

import ru.makhmutov.task3.interfaces.Swimmable;

public class Submarine implements Swimmable {

    boolean isSwimming = false;

    @Override
    public void swim() {
        if (!isSwimming) {
            System.out.println("The " + this.getClass().getSimpleName() +
                    " is swimming in the military zone");
            isSwimming = true;
        }
    }

    @Override
    public void stopSwimming() {
        if (isSwimming) {
            System.out.println("The " + this.getClass().getSimpleName() +
                    " stopped swimming in the military zone");
            isSwimming = false;
        }
    }
}
```

Exercise 3 solution: Duck.java

```java
package ru.makhmutov.task3.objects;

import ru.makhmutov.task3.interfaces.Flyable;
import ru.makhmutov.task3.interfaces.Living;
import ru.makhmutov.task3.interfaces.Swimmable;

public class Duck implements Living, Swimmable, Flyable {
    boolean isSwimming = false;
    boolean isFlying = false;

    @Override
    public void swim() {
        if (!isSwimming) {
            System.out.println("The " + this.getClass().getSimpleName() + " is swimming in the river");
        }
    }

    @Override
    public void stopSwimming() {
        if (isSwimming) {
            System.out.println("The " + this.getClass().getSimpleName() + " stopped swimming in the river");
            isSwimming = false;
        }
    }

    @Override
    public void fly() {
        if (!isFlying) {
            System.out.println("The " + this.getClass().getSimpleName() + " is flying in the air");
        }
    }

    @Override
    public void stopFlying() {
        if (isFlying) {
            System.out.println("The " + this.getClass().getSimpleName() + " stopped flying in the air");
        }
    }
}
```

Exercise 3 solution: Penguin.java

```java
package ru.makhmutov.task3.objects;

import ru.makhmutov.task3.interfaces.Living;
import ru.makhmutov.task3.interfaces.Swimmable;

public class Penguin implements Living, Swimmable {

    boolean isSwimming = false;

    @Override
    public void swim() {
        if (!isSwimming) {
            System.out.println("The " + this.getClass().getSimpleName() +
                    " is swimming between icebergs");
            isSwimming = true;
        }
    }

    @Override
    public void stopSwimming() {
        if (isSwimming) {
            System.out.println("The " + this.getClass().getSimpleName() +
                    " stopped swimming between icebergs");
            isSwimming = false;
        }
    }
}
```

Exercise 3 solution: InterfaceDemonstration.java

```java
package ru.makhmutov.task3;

import ru.makhmutov.task3.interfaces.Living;
import ru.makhmutov.task3.objects.Duck;
import ru.makhmutov.task3.objects.Penguin;

public class InterfaceDemonstration {

    /**
     * The entry point of the DifferentInterfaces program.
     * It allows creating objects implementing different set
     * of interfaces including Living, Swimmable and Flyable
     *
     * @param args Array with parameters of the program
     */
    public static void main(String[] args) {
        Living[] livings = addLivingObjects();
        letObjectsLive(livings);
    }

    /**
     * This method allows all elements of the array live
     *
     * @param livings the array of living objects
     */
    private static void letObjectsLive(Living[] livings) {
        for (Living living : livings) {
            living.live();
        }
    }

    /**
     * This method allows adding living objects into
     * the array
     *
     * @return the array of living things
     */
    private static Living[] addLivingObjects() {
        Living[] livings = new Living[2];

        Duck duck = new Duck();
        livings[0] = duck;

        Penguin penguin = new Penguin();
        livings[1] = penguin;

        return livings;
    }
}
```

References

- Inheritance, abstract classes, interfaces https://medium.com/@isaacjumba/overview-of-inheritance-interfaces-and-abstract-classes-in-java-3fe22404baf8
- Polymorphism https://codegym.cc/groups/posts/99-how-to-use-polymorphism