Java Course

Object - Oriented Programming

Table of Content

Abstract Class

- Abstract class is a restricted class that cannot be used to create objects
- To access it, it must be inherited from another class
- Abstract class can have both abstract and regular (non-abstract) methods

- `abstract` keyword
- It cannot be instantiated (objects cannot be created from abstract class)
- Abstract and regular methods

```
2
3   // create an abstract class
4   public abstract class Animal {
5     // fields and methods
6
7     // abstract method (no body)
8     public abstract void sound();
9
10    // regular method
11    public void sleep() {
12      System.out.println("zzz...");
13    }
14  }
15  ...
16
17  // try to create an object Animal
18  // throws an error
19  Animal obj = new Animal();
20
21
```

Abstract methods

- Abstract methods have only method declaration and **no body**
- `abstract` keyword is used for abstract methods as well
- Abstract methods can only be defined in Abstract classes
- If an abstract class contains abstract method, that method has to be implemented in all child classes
- All the methods and attributes from the abstract class can be accessed from child classes

```java
// create an abstract class
public abstract class Animal {
  // fields and methods

  // abstract method (no body)
  public abstract void sound();

  // regular method
  public void sleep() {
    System.out.println("zzz...");
  }

}

public class Dog extends Animal {

  @Override
  public void sound() {
    System.out.println("Woof");
  }
}

public class Snake extends Animal {

  @Override
  public void sound() {
    System.out.println("Ssssss");
  }
}
```

Constructors in abstract classes

- Abstract classes can have constructors
- Even though abstract class cannot be instantiated (we cannot create objects), we can call constructor of the abstract class from child classes

```java

// create an abstract class
public abstract class Animal {

  public Animal() {

  }

  // abstract method (no body)
  public abstract void sound();

}

public class Dog extends Animal {

  public Dog() {
    // calling construcstor of
    // abstract class
    super()
  }

  @Override
  public void sound() {
    System.out.println("Woof");
  }
}
```

Abstraction as a concept

- Abstract classes and abstract methods (along with interfaces) in Java are used to achieve abstraction
- Abstraction allows us to hide unnecessary details and show only the needed information
- It allows managing complexity by omitting or hiding details with a simpler and higher-level idea

Abstraction Example

- Let’s check the example with the motorbike brakes
- We know what brake does - when we apply the brake, the motorbike will stop
- However, the working of brake is kept hidden from us
- The major advantage of hiding the working of the brake is that now the manufacturer can implement the brake differently for different types of motorbikes (*method body*), however, what brake does will be the same (*method declaration*)

- `MotorBike` has an abstract method `brake`
- The `brake()` method cannot be implemented in `MotorBike` class because every bike has different implementation of brakes
- The implementation of brake in `MotorBike` is kept hidden
- Every specific bike (classes that extend `MotorBike`) will have it's own implementation of `brake`

```
2
3   // abstract class
4   public abstract class MotorBike {
5
6     // hidden complexity - no implementation
7     public abstract void brake();
8
9   }
10
11  public class MountainBike extends MotorBike {
12
13    // implement mountain brakes
14    @Override
15    public void brake() {
16      System.out.println("Mountain brakes");
17    }
18  }
19
20  public class SportsBike extends MotorBike {
21
22    // implement sport brakes
23    @Override
24    public void brake() {
25      System.out.println("Sport brakes");
26    }
27  }
28
```

Interface

- Another way of achieving abstraction in Java
- Like abstract classes, interfaces **cannot** be used to create objects
- An Interface is a *completely abstract "class"* that is used to group related methods with empty bodies
- To access the interface methods, the interface must be *implemented* by another class
- The body of the interface method is implemented in the class that `implements` an interface
- With interfaces, we use `implements` keyword instead of `extends`

- We have a Printable interface
- Interface has only one method - `print()`
- Methods in interfaces do not have implementation
- Methods in interfaces are by default `abstract` and `public`
- Interface can have attributes
- Attributes are by default `public`, `static` & `final`
- Implementation of interface methods is provided in the child class

```java

// interface
public interface Printable {

  // interface method - no body
  public void print();

}

public Dog implements Printable {

  @Override
  public void print() {
    System.out.println("I am a dog!");
  }
}
```

Interface

- When implementing interface, you must override (implement) all of its methods
- Interface cannot contain constructor - interface and abstract class are not the same
- Interface can extend another interface, but the class that implements extended interface has to implement methods from both interfaces

Interface over Abstract Class?

- Interfaces are used for two reasons:
  - Achieving abstraction
  - Achieving multiple inheritance

Multiple Inheritance

```
2
3 public interface Polygon {
4
5     public double calculateArea();
6
7 }
8
```

```
2
3 public interface Printable {
4
5     public void print();
6 }
7
```

```
2
3 public class Square implements Polygon, Printable {
4
5     private double a;
6
7     public Square() {
8
9     }
10
11    public Square(double a) {
12        this.a = a;
13    }
14
15    @Override
16    public void print() {
17        System.out.println("I am a square with a=" + this.a);
18
19    }
20
21    @Override
22    public double calculateArea() {
23        return this.a * this.a;
24    }
25 }
```

Exercise 1

- Let’s create an interface called `Instrument` and a class `MusicalNote`
- `MusicalNote` has the name and frequency in hertz (double) (for example: note named D has the frequency of 293.6)
- Create constructors and getters & setters for `MusicalNote`
- Interface `Instrument` should have two methods:
  - `tune` - this method has one parameter: note (of `MusicalNote` type)
  - - this method has one parameter: note (of `MusicalNote` type)
- Let’s create some classes that implement `Instrument` (for example Piano, Guitar,…) and add missing implementation for `tune` and `play` methods
  - `play` should print what `MusicalNote` we are playing, and what is the instrument that is played on
  - `tune` should print “`Tuning the D note to 293.6Hz on Piano…`”