Unit III

Implementing OOP Concepts (Abstract, Interface, Polymorphism)

Tutor: Pema Galey

Learning Outcomes

In this session, you will learn about:

- Abstract Class and Method
- Interfaces
- Polymorphism
- Encapsulation

Abstract Class

- The abstract class in Java cannot be instantiated (we cannot create objects of abstract classes).
- We use the ***abstract*** keyword to declare an abstract class.
- Example:

```
// create an abstract class
abstract class Language {
        // fields and methods
}
...
// try to create an object Language, throws an error
Language obj = new Language();
```

Abstract Class

- An abstract class can have both the regular methods and abstract methods.
- For example,

```
abstract class Language {
        // abstract method
        abstract void method1();
        // regular method
        void method2() {
                System.out.println("This is regular method");
        }
}
```

Abstract Method

- A method that doesn't have its body is known as an abstract method.
- We use the same ***abstract*** keyword to create abstract methods.

- For example,

```
abstract void display();
```

Abstract Method

- If a class contains an abstract method, then the class should be declared abstract. Otherwise, it will generate an error.
- For example,

```
// error
// class should be abstract
class Language {  // abstract class Language
        // abstract method
        abstract void method1();
}
```

Abstract Class and Method

- Though abstract classes cannot be instantiated, we can create ***subclasses*** from it.
- We can then access members of the abstract class using the object of the subclass.

- For example,

```
abstract class Language {
        // method of abstract class
        public void display() {
                System.out.println("This is Java Programming");
        }
}
class Main extends Language {
        public static void main(String[] args) {
                Main obj = new Main();
                obj.display();
}}
```

Implementing Abstract Methods

- If the abstract class includes any abstract method, then all the child classes inherited from the abstract superclass must provide the implementation of the abstract method.
- Example (next slide)

Implementing Abstract Methods

```
abstract class Animal {
        abstract void makeSound();
        public void eat() {
                System.out.println("I can eat.");
} }
class Dog extends Animal {
        // provide implementation of abstract
        // method
        public void makeSound() {
                System.out.println("Bark bark");
        }
}
```

```
class Main {
        public static void main(String[] args) {
                Dog d1 = new Dog();
                d1.makeSound();
                d1.eat();
} }
```

Accesses Constructor of Abstract Classes

- An abstract class can have constructors like the regular class. And, we can access the constructor of an abstract class from the subclass using the ***super*** keyword.
- For example,

```
abstract class Animal {
        Animal() { …. }
 }
 class Dog extends Animal {
        Dog() {
                super();
                ...
        }
 }
```

Java Abstraction

- The major use of abstract classes and methods is to achieve abstraction in Java.
- Abstraction is an important concept of object-oriented programming that allows us to hide unnecessary details and only show the needed information.
- This allows us to manage complexity by omitting or hiding details with a simpler, higher-level idea.
- Example:
  - Motorbike brakes
  - Animal sound (dog, cat, etc.)

Abstraction: Key Points to Remember

- We use the abstract keyword to create abstract classes and methods.
- An abstract method doesn't have any implementation (method body).
- A class containing abstract methods should also be abstract.
- We cannot create objects of an abstract class.
- To implement features of an abstract class, we inherit subclasses from it and create objects of the subclass.
- A subclass must override all abstract methods of an abstract class. However, if the subclass is declared abstract, it's not mandatory to override abstract methods.
- We can access the static attributes and methods of an abstract class using the reference of the abstract class.
- Example: *Animal.staticMethod();*

Interface

- An interface is a fully abstract class.
- It includes a group of abstract methods (methods without a body).
- We use the ***interface*** keyword to create an interface in Java.
- For example,

```
interface Language {
        public void getType(); //abstract method
        public void getVersion(); //abstract method
}
```

Implementing an Interface

- Like abstract classes, we cannot create objects of interfaces.
- To use an interface, other classes must implement it.
- We use the ***implements*** keyword to implement an interface.

Example for Interface

```
interface Polygon {
        void getArea(int length, int breadth);
}
// implement the Polygon interface
class Rectangle implements Polygon {
        // implementation of abstract method
        public void getArea(int length, int breadth) {
        System.out.println("The area of the
        rectangle is " + (length * breadth));
        }
}
```

```
class Main {
        public static void main(String[] args) {
                Rectangle r1 = new Rectangle();
                r1.getArea(5, 6);
        }
}
```

Implementing Multiple Interfaces

- In Java, a class can also implement multiple interfaces.
- For example,

```
interface A {
        // members of A
}
interface B {
        // members of B
}
class C implements A, B {
        // abstract members of A
        // abstract members of B
}
```

Extending an Interface

- Similar to classes, interfaces can extend other interfaces. The ***extends*** keyword is used for extending interfaces.
- For example,

```
interface Line {
        // members of Line interface
}
// extending interface
interface Polygon extends Line {
        // members of Polygon interface
        // members of Line interface
}
```

Extending Multiple Interfaces

- An interface can extend multiple interfaces.
- For example,

```
interface A {
        ...
}
interface B {
        ...
}
interface C extends A, B {
        ...
}
```

Multiple Inheritance

- Interfaces able to achieve multiple inheritance.
- Example:

```
interface Line {
        …
}
interface Polygon {
        …
}
class Rectangle implements Line, Polygon {
        …
}
```

Advantages of Interface in Java

- Similar to abstract classes, interfaces help us to achieve **abstraction in Java**.
- Interfaces **provide specifications** that a class (which implements it) must follow.
- Interfaces are also used to achieve multiple inheritance in Java.

Data Members inside Interface

- **Note**: All the methods inside an interface are implicitly *public* and all fields are implicitly *public static final.*
- For example,

```
interface Language {
        // by default public static final
        String type = "programming language";
        // by default public
        void getName();
}
```

Default methods in Java Interfaces

- We can now add methods with implementation inside an interface. These methods are called default methods.
- To declare default methods inside interfaces, we use the **default** keyword.

- For example,

```
public default void getSides() {
        // body of getSides()
}
```

Default Method inside Interface

```
interface Polygon {
        void getArea();
        // default method
        default void getSides() {
                System.out.println("I can get sides of a polygon."); } }
 // implements the interface
class Rectangle implements Polygon {
        public void getArea() {
                int length = 6; int breadth = 5;
                int area = length * breadth;
                System.out.println("The area of the rectangle is " + area);
        }
        // overrides the getSides()
        public void getSides() {
                System.out.println("I have 4 sides.");
} }
```

```
// implements the interface
class Square implements Polygon {
        public void getArea() {
                int length = 5; int area = length * length;
                System.out.println("The area of the square is " + area);
        }
}
```

Why default methods inside Interface?

- We can add the method in our interface easily without implementation. However, that's not the end of the story. All our classes that implement that interface must provide an implementation for the method.
- If a large number of classes were implementing this interface, we need to track all these classes and make changes to them. This is not only tedious but error-prone as well.
- To resolve this, Java introduced default methods. Default methods are inherited like ordinary methods.

private and static Methods in Interface

- Feature to include static methods inside an interface.
- Similar to a class, we can access static methods of an interface using its references.
- For example,

```
// create an interface
interface Polygon {
        staticMethod(){..}
}
// access static method
Polygon.staticMethod();
```

Java Polymorphism

- Polymorphism is an important concept of object-oriented programming. It simply means more than one form.
- That is, the same entity (method or operator or object) can perform different operations in different scenarios.
- We can achieve polymorphism in Java using the following ways:
  - Method Overriding
  - Method Overloading
  - Operator Overloading

Java Encapsulation

- Encapsulation is one of the key features of object-oriented programming. Encapsulation refers to the bundling of fields and methods inside a single class.
- It prevents outer classes from accessing and changing fields and methods of a class. This also helps to achieve **data hiding.**

- **Data Hiding**
  - Data hiding is a way of restricting the access of our data members by hiding the implementation details. Encapsulation also provides a way for data hiding.

Thank you!