OBJECT ORIENTED PROGRAMMING

Affefah Qureshi

Department of Computer Science

Iqra University, Islamabad Campus.

INTERFACES

- Interfaces in Java are a crucial part of its type system and are used to define a contract that classes can implement. They allow for a form of multiple inheritance, which Java does not support through classes, and provide a way to achieve abstraction.
- Key Characteristics of Interfaces:

1. Abstract Methods: Interfaces can contain abstract methods, which are methods without a body. Any class that implements the interface must provide implementations for these methods.
2. Default Methods: Since Java 8, interfaces can have default methods, which are methods with a body. These methods provide default implementations that implementing classes can override.
3. Static Methods: Interfaces can also have static methods, which can be called on the interface itself.
4. Constant Fields: Fields in an interface are implicitly public, static, and final, meaning they are constants.
5. No Constructor: Interfaces cannot have constructors because they cannot be instantiated.

INTERFACES

- An interface is defined much like a class. This is the general form of an interface:

```
access-specifier interface InterfaceName {
  return-type method-name1(parameter-list);
  return-type method-name2(parameter-list);
  type final-varname1 = value;
  type final-varname2 = value;
  // ...
  return-type method-nameN(parameter-list);
  type final-varnameN = value;
}
```

INTERFACES

- access-specifier for interface is either public or not used.
- Notice that the methods which are declared have no bodies.
  - They are, essentially, abstract methods;
- Variables declared inside an interface
  - Are implicitly final and static, meaning they cannot be changed by the implementing class.
  - Must also be initialized with a constant value.
- All methods and variables are implicitly public if the interface, itself, is declared as public.

EXAMPLE

```
// Defining an interface
interface Animal {
    // Abstract method
    void sound();

    // Default method
    default void sleep() {
        System.out.println("Animal is sleeping");
    }

    // Static method
    static void info() {
        System.out.println("This is an Animal
interface");
    }
}

// Implementing the interface in a class
class Dog implements Animal {
    @Override
    public void sound() {
        System.out.println("Dog barks");
    }
}
```

```
class Cat implements Animal {
    @Override
    public void sound() {
        System.out.println("Cat meows");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.sound(); // Outputs: Dog barks
        dog.sleep(); // Outputs: Animal is
sleeping

        Cat cat = new Cat();
        cat.sound(); // Outputs: Cat meows
        cat.sleep(); // Outputs: Animal is sleeping

        Animal.info(); // Outputs: This is an
Animal interface
    }
}
```

INTERFACES

- Once it is defined, any number of classes can implement an interface.
- Also, one class can implement any number of interfaces.
- To implement an interface, a class must create the complete set of methods defined by the interface.
- Interfaces are designed to support dynamic method resolution at run time.
- Interfaces have a different hierarchy from classes, thus, it is possible for classes that are unrelated in terms of the class hierarchy to implement the same interface.
- This is where the real power of interfaces is realized.

DYNAMIC METHOD RESOLUTION

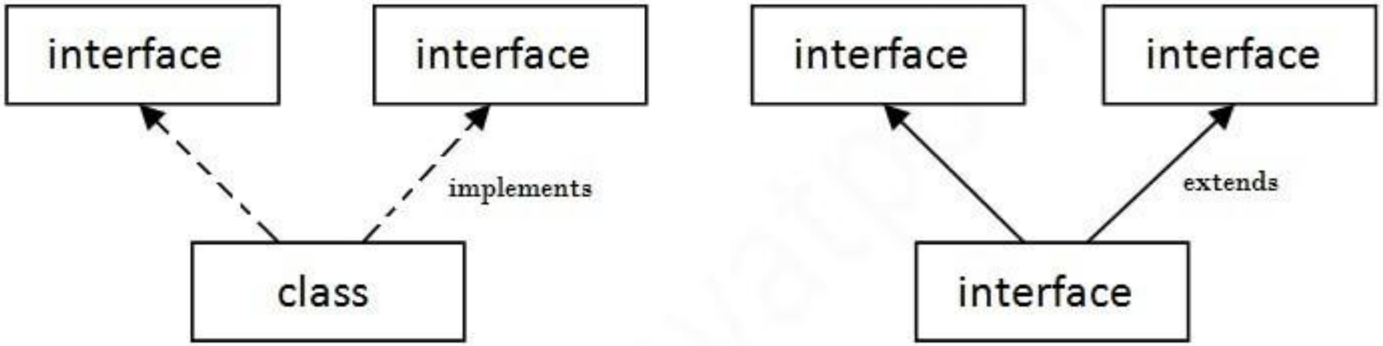
- Dynamic method resolution means that the} method to be invoked is determined at runtime rather than compile-time. This allow for more dynamic and flexible program behavior.

```
interface Callback {
    void callbackMethod1();
    void callbackMethod2();
}

abstract class Incomplete implements
Callback {
    int a, b;

    void show() {
        System.out.println(a + " " + b);
    }

    @Override
    public void callbackMethod1() {
        System.out.println("Callback method 1
implemented in Incomplete.");
    }
```

```
class Complete extends Incomplete {
    @Override
    public void callbackMethod2() {
        System.out.println("Callback method 2
implemented in Complete.");
    }
}

public class Main {
    public static void main(String[] args) {
        Callback obj = new Complete();
        obj.callbackMethod1(); // Calls method
from Incomplete
        obj.callbackMethod2(); // Calls method
from Complete
    }
}
```

WHY DO WE NEED INTERFACE WHEN ABSTRACT CLASSES EXIST IN JAVA?

- If abstract class doesn't have any method implementation, its better to use interface because java doesn't support multiple class inheritance
- The subclass of abstract class in java must implement all the abstract methods unless the subclass is also an abstract class



Multiple Inheritance in Java

IMPLEMENTING INTERFACES

- Once an interface has been defined, one or more classes can implement that interface.
- To implement an interface, include the implements clause in a class definition, and then create the methods defined by the interface.
- The general form of a class that includes the implements clause looks like this:

```
access-specifier class classname [extends superclass]
[implements interface [,interface...]] {
    // class-body
  }
```

- access-specifier is either public or not used.
- If a class implements more than one interface, the interface names are separated with a comma.
- If a class implements two interfaces that declare the same method, but that method should be implemented only once in the class.
- The methods that implement an interface must be declared public.

EXAMPLE OF MULTIPLE INTERFACES

```
// Defining interfaces
interface Flyable {
    void fly();
}

interface Swimmable {
    void swim();
}

// Implementing multiple interfaces
in a single class
class Duck implements Flyable,
Swimmable {
    @Override
    public void fly() {
        System.out.println("Duck flies");
    }
```

```
    @Override
    public void swim() {
        System.out.println("Duck
swims");
    }
}

public class Main {
    public static void main(String[]
args) {
        Duck duck = new Duck();
        duck.fly(); // Outputs: Duck flies
        duck.swim(); // Outputs: Duck
swims
    }
}
```

DIFFERENCES BETWEEN INTERFACES AND ABSTRACT CLASSES:

- Abstract Class:
  - Can have both abstract and concrete methods.
  - Can have member variables.
  - Can have constructors.
  - A class can extend only one abstract class (single inheritance).

- Interface:
  - Can only have abstract methods (until Java 8, now can have default and static methods).
  - Cannot have member variables (except static and final fields).
  - Cannot have constructors.
  - A class can implement multiple interfaces (multiple inheritance).

REAL WORLD EXAMPLE

```
// Defining the Payment interface
interface Payment {
    void pay(double amount);
}

// Implementing the interface in different payment classes
class CreditCard implements Payment {
    @Override
    public void pay(double amount) {
        System.out.println("Paid " + amount + " using Credit Card");
    }
}

class PayPal implements Payment {
    @Override
    public void pay(double amount) {
        System.out.println("Paid " + amount + " using PayPal");
    }
}

public class Main {
    public static void main(String[] args) {
        Payment payment1 = new CreditCard();
        payment1.pay(100.0); // Outputs: Paid 100.0 using Credit Card

        Payment payment2 = new PayPal();
        payment2.pay(200.0); // Outputs: Paid 200.0 using PayPal
    }
}
```