COJ :: Abstract Classes & Interfaces

TalentSprint

Licensed To Skill

Version 1.0.4

Learning Objectives

The content in this presentation is aimed at teaching learners to:

- Define abstract class
- Explain why abstract classes are needed
- Use abstract classes in writing small java applications
- Define interface
- Use interfaces in writing small java applications
- Use “final” keyword

Abstract Classes & Interfaces

Abstract class

- Is a conceptual class
- Provides a common root for a group of classes, nicely tied together in a package
- Cannot be instantiated – objects cannot be created

Abstract Classes & Interfaces

Abstract class syntax

```
abstract class ClassName {
    ...
    abstract Type MethodName1();
    ...
    Type Method2() {
        // method body
    }
}
```

- When a class contains zero or more abstract methods, it should be declared as abstract class
- The abstract methods of an abstract class must be defined in its subclass
- We cannot declare abstract constructors or abstract static methods

Abstract Classes & Interfaces

Abstract Class - Example


```
public abstract class Shape {
    public abstract double area();
    public void move() { // non–abstract method
        // implementation
    }
}
```

Is the following statement valid?

Shape s = **new** Shape();

No. It is illegal because the Shape class is an abstract class, which cannot be instantiated to create its objects

Abstract classes & Interfaces

Abstract Classes

```
public Circle extends Shape {
    protected double r;
    protected static final double PI = 3.1415926535;
    public Circle() {
        r = 1.0;
    }
    public double area() {
        return PI * r * r;
    }
}

public Rectangle extends Shape {
    protected double w, h;
    public Rectangle() {
        w = 0.0, h = 0.0;
    }
    public double area() {
        return w * h;
    }
}
```

Abstract Classes & Interfaces

Abstract Classes Properties

A class with one or more abstract methods is automatically abstract and it cannot be instantiated

```
public abstract class Shape {
    public abstract double area();
    public void move() { // non−abstract method
        // implementation
    }
}
```

Shape s = **new** Shape(); // *Error*

Abstract Classes & Interfaces

Abstract Classes Properties

A class declared abstract, even with no abstract methods can not be instantiated

```
public abstract class Triangle {
    private int radius;
    public double area() { // non−abstract method
        // implementation
    }
    public void move() { // non−abstract method
        // implementation
    }
}
```

Triangle triagle = **new** Triangle(); // *Error*

Abstract Classes & Interfaces

Abstract Classes Properties

A subclass of an abstract class can be instantiated if it overrides all abstract methods by implementation them

```
public abstract class Shape {
    public abstract double area();
    public void move() { // non−abstract method
        // implementation
    }
}
public Rectangle extends Shape {
    protected double w, h;
    public Rectangle() { w = 0.0; h=0.0; }
    public double area() { return w * h; }
}
```

Rectangle rectangle = **new** Rectangle(); // *Legal*

Abstract Classes & Interfaces

Abstract Classes Properties

A subclass that does not implement all of the superclass abstract methods is itself abstract; and it cannot be instantiated

```
public abstract class Shape {
    public abstract double area();
    public void move() { // non−abstract method
        // implementation
    }
}
public abstract Rectangle extends Shape {
    protected double w, h;
    public Rectangle() { w = 0.0; h=0.0; }
}
```

Rectangle rectangle = **new** Rectangle(); // *Illegal*

Abstract Classes & Interfaces

Interfaces

- Interface is a conceptual entity similar to an Abstract class
- Can contain only constants (final variables) and abstract method (no implementation) - Different from Abstract classes
- Use when a number of classes share a common interface. Each class should implement the interface

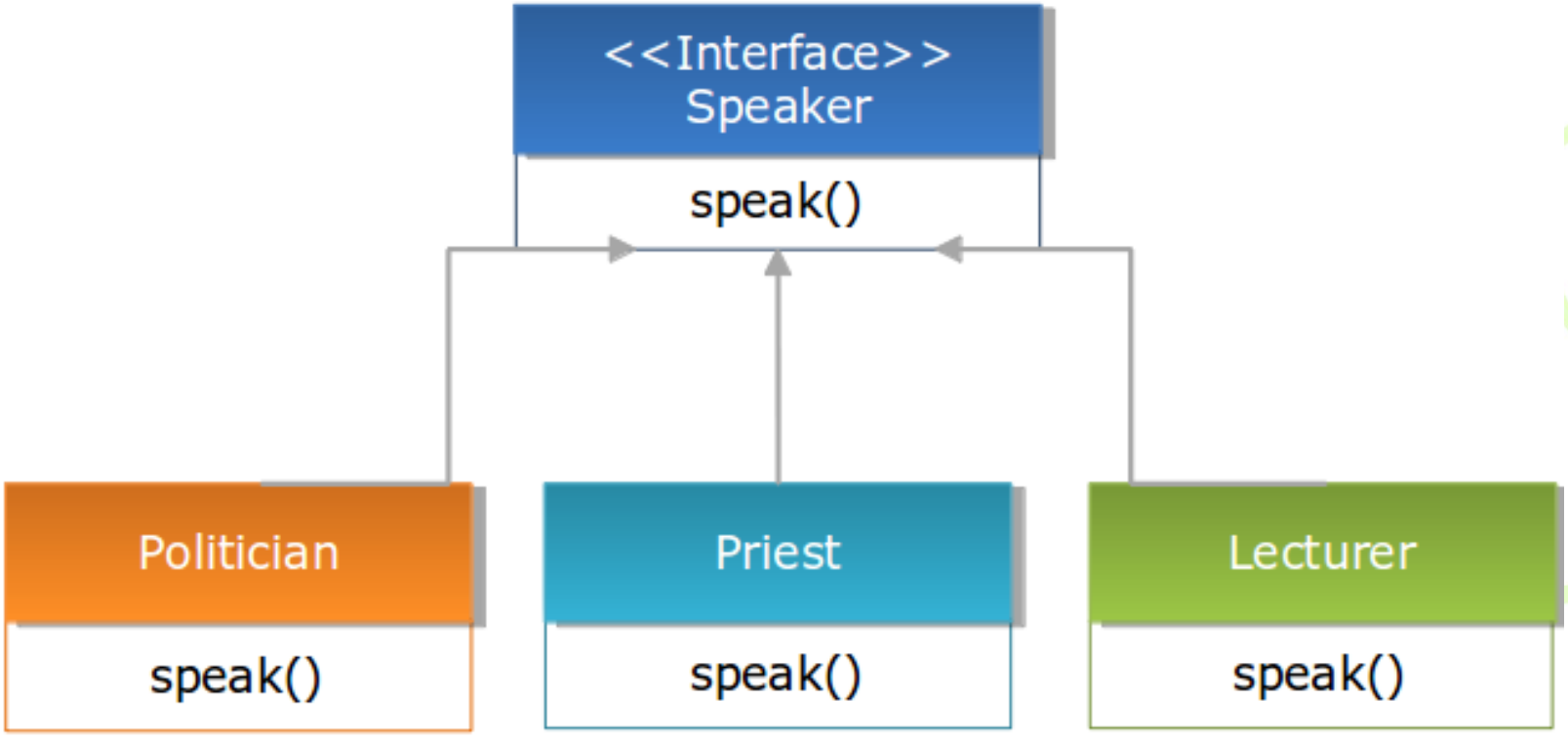
Abstract Classes & Interfaces

`Interfaces: An informal way of realising multiple inheritance`

- An interface is basically a kind of class - it contains methods and variables, but they have to be only abstract classes and final fields/variables
- Therefore, it is the responsibility of the class that implements an interface to supply the code for methods
- A class can implement any number of interfaces, but cannot extend more than one class at a time. Therefore, interfaces are considered as an informal way of realising multiple

Abstract Classes & Interfaces

Interface - Example



Abstract Classes & Interfaces

Interface Syntax

```
interface InterfaceName {
    // Constant/Final Variable Declaration
    // Methods Declaration – only method body
}
```

Interface Example

```
interface Speaker {
    public void speak( );
}
```

Abstract Classes & Interfaces

Implementing Interfaces

Interfaces are used like super-classes who properties are inherited by classes. This is achieved by creating a class that implements the given interface as follows:

```
class ClassName implements
    InterfaceName [InterfaceName1,
    InterfaceName2, ...] {
      // Body of Class
}
```

Abstract Classes & Interfaces

Implementing Interfaces Examples

```
class Politician implements Speaker {
    public void speak() {
        System.out.println("Talk politics");
    }
}
```

```
class Priest implements Speaker {
    public void speak() {
        System.out.println("Religious Talks");
    }
}
```

```
class Lecturer implements Speaker {
    public void speak() {
        System.out.println("Talks Object Oriented Design and Programming!");
    }
}
```

Abstract Classes & Interfaces

Inheritance and Interface Implementation

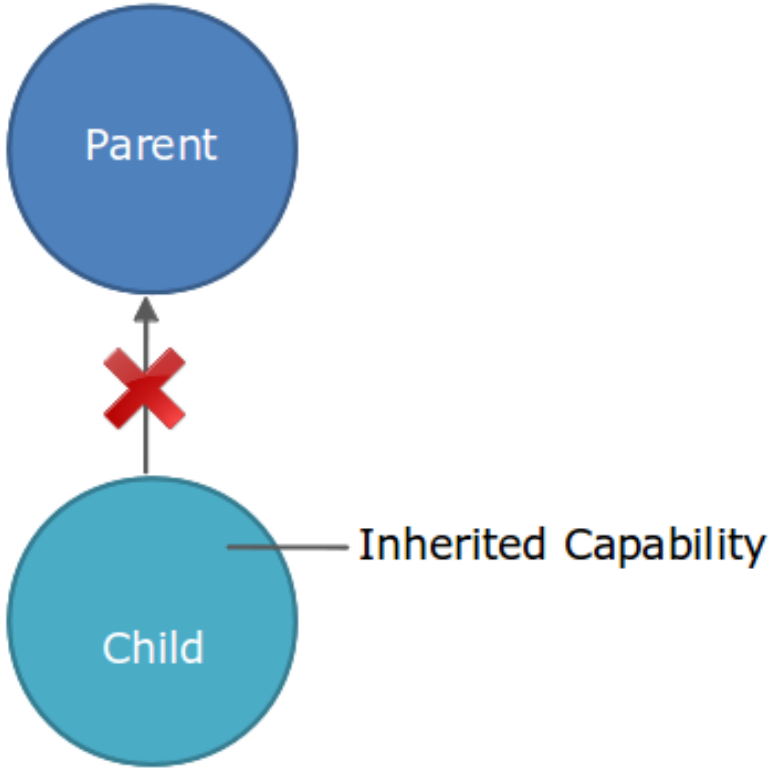
A general form of interface implementation:

```
class ClassName extends SuperClass implements InterfaceName [, InterfaceName2, ...] {
    // Body of Class
}
```

This shows a class can extended another class while implementing one or more interfaces. It appears like a multiple inheritance (if we consider interfaces as special kind of classes with certain restrictions or special features)

Abstract Classes & Interfaces

Final



Abstract Classes & Interfaces

Final Classes: A way for Preventing Classes being extended

We can prevent an inheritance of classes by other classes by declaring them as final classes. This is achieved in Java by using the following keyword final:

```
final class Marks {
    // members
}
final class Student extends Person {
    // members
}
```

Any attempt to inherit these classes will cause an error.

Abstract Classes & Interfaces

Final Members: A way for Preventing Overriding of Members in Subclasses

- All methods and variables can be overridden by default in subclasses
- This can be prevented by declaring them as final using the keyword “final” as a modifier
- Example:
  final int marks = 100;
  final void display();
- This ensures that functionality defined in this method cannot be altered any. Similarly, the value of a final variable cannot be altered

Abstract Classes & Interfaces