JAC444 - Lecture 3

Object-Oriented Concepts

Segment 2 – Inheritance

Classes – Segment 2 – Inheritance

In this segment you will be learning about:

- Inheritance
- Overriding
- Final Methods and Classes
- Implementing and Extending Interfaces with Default Methods
- Abstract Classes

Inheritance

Definition:

A *subclass* is a class that extends another class.

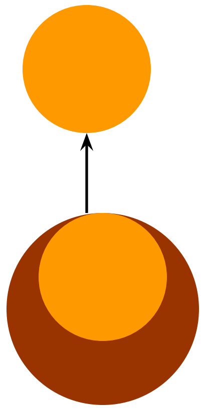
A subclass inherits state and behavior from all its ancestor.

The *superclass* refers to a direct ancestor.

Subclass inherits all, but private superclasses members

```
public class SuperClass { … }

public class SubClass extends SuperClass {
      …
}
```



Overriding

- Definition:
  Replacing the superclass's implementation with a new method in a subclass is called *overriding.*

  - The signature should be identical.
  - Only accessible non-static method can be overridden.
  - Access modifier could be different in overridden method as long as the subclass modifier is less restrictive than the superclass.
  - A subclass can change whether a parameter in an overridden method is final (final is not part of method signature).
  - Fields cannot be overridden; they can only be hidden.

super acts as a reference accessing fields and method of superclass.

- Ex*:* `super.superclassField;`

Overriding and Hiding - Example

```
 class Base {
      public String s = “X”;
      public void show() { System.out.println(s); } }

 class Extended extends Base {
      public String s = “Y”;
      public void show() { System.out.println(s); }
      public static void main(String[] args) {
           Extended e = new Extended();
           Base b = e;
           b.show();
           e.show();
           System.out.println(b.s + “ “ + e.s); }
}
```

Results: **Y Y X Y**

When invoke a method on an object, the actual class of the object governs which implementation is used.

When access a field the declared type of the referenced is used.

Final Methods and Classes

- A method could be declared as ***`final`***
  - A final method **cannot be overridden**.
- A class could be declared as ***`final`***
  - A final class **cannot be subclassed**.

Example: Immutable class like ***`String`*** class

Implementing / Extending

- Implementing Interface I

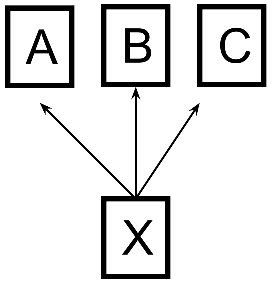
```
interface I { void m(); }
class A implements I { void m1() { … } }
```

- Extending Interface I

```
interface J extends I { void m2(int i); }
class A implements J {
     void m1() { … }
    void m2(int i) { … }
}
```

Interface accepts multiple inheritance

```
interface X extends A, B, C { … }
```

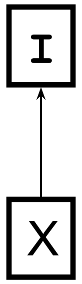


Default and Static Methods

- Interface could contain Static Methods and Default Methods

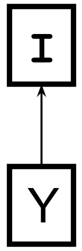
One can add new methods to an old interface, without breaking old code

```
interface I { void m(); }
class X implements I { … }
```



- Evolving the `I` Interface

```
interface I  {
    void m();
    default String n(int k){
        if (k % 2)
         return "It is OK";
    }
}
class Y implements I { … }
```



Abstract Methods / Classes

- Abstract Method is a method without implementation

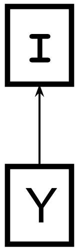
```
absract void movePoint(int deltaX, int deltaY);
```

Abstract Class is a class with at least an abstract method

```
public abstract class X { //fields //other methods
    absract void movePoint(int deltaX, int deltaY);
}
```

- Evolving the `I` Interface

```
interface I  {
    void m();
    default String n(int k){
       if (k % 2)
         return “It is OK”;
    }
}
class Y implements I { … }
```



Extending Interfaces - revisited

- Extending an interface with default methods

- Three options:
  1. Ignore the default methods – inherit the default methods
  2. Redeclare the default method (makes the method an abstract method)
  3. Redefine the default method – overrides it.

- Interface declaration can contain:
  4. Method signatures
  5. Default methods
  6. Static methods
  7. Constant definitions

Annotations

- Annotation does not affect the program semantics

- Annotations are used by development tools to generate new artifacts or to check the properties of class / methods, etc.

- Previous annotation were defined in JavaDoc such as:
  - `@author`
  - `@version`

Annotation types are imported in the same fashion as classes and interfaces

Annotation Example and Use

Example of using the annotation `@Override`

```
public class Example {
   @Override
   public int hashCode { return toString().hashCode(); }
}
```

Annotation can be used anywhere you use a type (starting with Java SE 8)

```
@NotNull String str;
```

Annotation type definitions

```
public @interface Preliminary {//Marker annotation }

public @interface Copyright { String value(); //Single member annotation }

public @interface Name { String first(); String last(); }
```