Object-oriented Software Design and Development

CCP114N

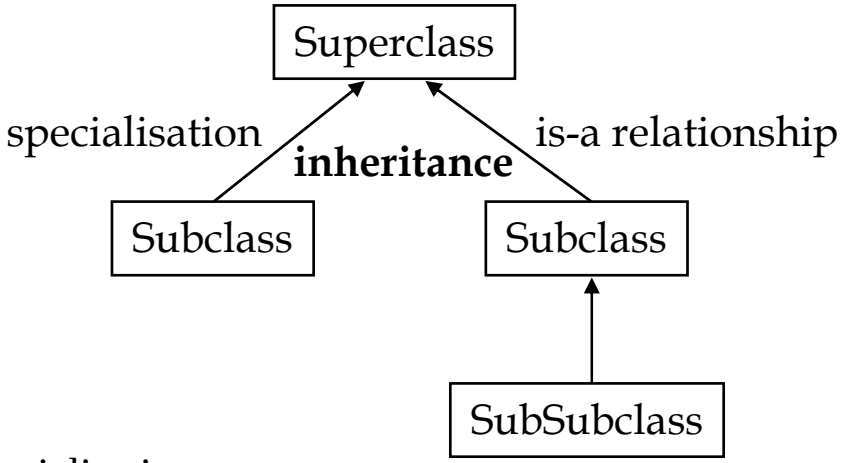
Week 10

- OO Features Revisited
  - Inheritance
  - Polymorphism
    - Dynamic binding
    - Via Inheritance vs. Via Interface types

Inheritance revisited

- A mechanism to achieve software reuse
- Newly created class inherits the non-private members of an existing class
  - Superclass / base class / parent class
  - Subclass / derived class / child class
- Inheritance saves time and efforts of writing and debugging the inherited code
- The subclass is normally added with its own members (data & functions). This is the whole point to create a new, specialised subclass from a superclass

Inheritance: the class hierarchy



- Relationship between superclass and subclasses may be represented in a tree-like diagram
  - Also called inheritance tree
- The arrow represents IS-A relationship
  - Because an object of a subclass is also an object of its superclass
  - The subclass-superclass relationship is also a specialisation relationship e.g.
    - Postgraduate Student Class is a specialisation of Student Class
- Single inheritance vs. multiple inheritance
  - Java doesn't support multiple inheritance
- Inheritance vs. Composition (or Aggregation)
  - Computer may be a subclass of IT equipment class
  - A Computer object is composed of objects of CPU, Monitor, Keyboard and Mouse
    - HAS-A relationship

Multiple inheritance

- A subclass inherits from more than one superclass
  - Not ‘directly’ supported in Java
- Java allows ‘extending’ only one class

```
public class myApp extends JFrame { …}
```

- But you can implements as many ‘interfaces’ as you need

```
public class myApp extends JFrame implements
ActionListener, WindowListener { …}
```

Beyond Inheritance …

- OO programming allows subclasses not only to inherit but also to extend and alter the members of the superclass
- Overriding
  - Redefined inherited methods with own implementation, keeping the method's signature unchanged e.g.
    - area( ) and circumference( ) in the shape example
  - The **final** keyword
    - A qualifier of a method that indicates the method cannot be overridden any further.
- Overloading
  - Overloading is used to define NEW methods with the same name, but different list of parameters in subclasses of a class e.g.
    - Multiple constructors, as in the Square class in the Shape example
- Extending with new data and function members
  - e.g. new method getRadius( ) in the Circle class example

What is Polymorphism?

- Polymorphism means many forms
  - Recall
    - Overriding
    - Overloading
  - Same method could behave differently depending on which object it is applied to e.g. area() is calculated differently for rectangles and circles
- Polymorphism vs. Inheritance
  - Inheritance is about sharing & commonality whereas Polymorphism - differentiation & distinction
  - Inheritance allows an object to be treated as its own type or its base type i.e. objects of many types that are subclasses of the same base type can be treated as if they were one type. And single piece of code will work on all those different types equally.
  - Polymorphism: the polymorphic method call allows one type to express its distinction from another, similar type, as long as they're both derived from the same base type. This distinction is expressed thru differences in behaviour of the polymorphic methods that are called thru the base class.

Another form of Polymorphism

Using Interface types in Java

The use of Interface types allows

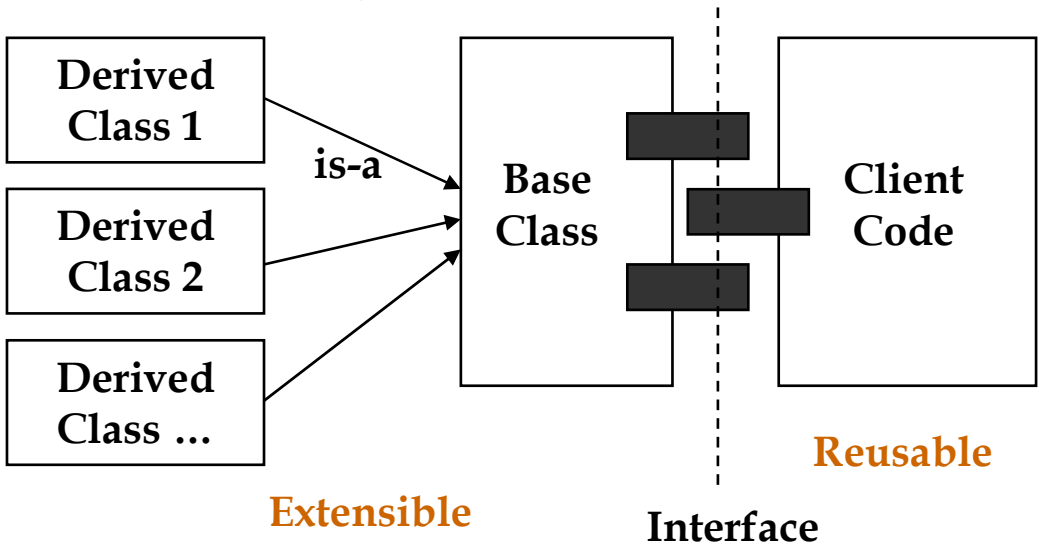
- brand new code of service provider class to implement an Interface type
  - Example: the Comparable interface
  - Example: the Comparator interface and anonymous object and class
- re-use the client's code
  - The client code can make calls to polymorphic methods defined by an Interface type
  - The client code can use objects of a class that implements an Interface type
    - Java.util.Arrays.sort(T[ ] array, Comparator a)
    - JOptionPane.showMessageDialog(…, Icon anIcon) see more in Horstmann p.140

Polymorphism: Dynamic Binding

- The object (or subtype) of polymorphic calls may not be known until the run-time
  - E.g. which implementation of the method area( ) should be used for a Shape object?
    - See example.
- Dynamic binding vs. Static binding
  - static binding: at compile time
  - dynamic binding: at run time
    - the compiler doesn't know which object's code will be run when compiling
  - benefits
    - re-use the client's code (the code that makes calls to polymorphic methods)
    - allow service provider's code extendibility
      - create new sub-class with different implementation of the super class methods
- see the Java working example

Polymorphism: Supporting Reuse and Extensibility

- Polymorphism supports reuse
  - The client code that manipulates the objects of base class's public interface will not need to be changed to accommodate the new derived classes
- Polymorphism supports extensibility
  - The base class may be extended/modified as long as it maintains the interface used by the client code



Summary

- Inheritance
  - The class hierarchy
  - Extensibility with own members
- Polymorphism
  - Overriding, Overloading
  - Meanings and how it is manifested in code
  - Dynamic binding vs. static binding
  - Polymorphism via Interface types in Java