Inheritance and Polymorphism

• Professor Evan Korth
• New York University

Road map

• Object class
• Polymorphism
• Superclass / subclass references and casting

review

• Define superclass.
• Define inheritance.
• Why do you need the keyword super?
  – What are the two uses of super?
• If you want to call a superclass constructor (with non
  zero parameters) in your subclass' constructor where
  must you call it?
• What is the difference between overloading and
  overriding?

Review (cont)

• Given the following class:
public class Bicycle extends Vehicle
  When you call one of Bicycle's constructors, which
  constructors are called and in which order do they do
  their work?
• When a client program invokes an object's methods, how
  does it use the keyword super?
• When a client program invokes an object's methods,
  what is the difference in syntax between invoking a
  method declared in the object vs. one that is directly
  inherited? How about indirectly inherited?

protected members and accessing a
superclass member

• protected access
  – Intermediate level of protection between public and
    private
  – protected members accessible to
    • superclass members
    • subclass members
    • Class members in the same package

• Subclass access superclass member
  – Keyword super and a dot (.)

Relationship between Superclasses and
Subclasses (Cont.)

• Using protected instance variables
  – Advantages
    • subclasses can modify values directly
    • Slight increase in performance
  – Avoid set/get function call overhead
  – Disadvantages
    • No validity checking
    • subclass can assign illegal value
    • Implementation dependent
  – subclass methods more likely dependent on superclass
    implementation
  – superclass implementation changes may result in subclass
    modifications
    • Fragile (brittle) software
Visibility modifiers for data and methods

- **Weak access privileges**
  - `private` – only seen in class (UML -)
  - `default` (no visibility modifier) – private access plus classes in the package (UML none)
  - `protected` – default access plus subclasses and superclasses (UML #)
- **Strongest access privileges**
  - `public` – protected access plus everything else (UML +)

Overload vs override (review)

- Overloading a method refers to having two methods which share the same name but have different signatures.
- Overriding a method refers to having a new implementation of a method with the same signature in a subclass.

Object class

- In Java, all classes are derived from other classes except the Object class which is the top of Java’s class hierarchy.
- Therefore, if a new class does not explicitly extend another class, it implicitly extends the Object class.
- Several of the methods provided in the Object class are provided with the intention that they will be overridden.

Object class: equals method

```java
public boolean equals(Object object)
```

- Object’s `equals()` method returns true if the Objects are the same Object (i.e., the two variables refer to the same position in memory).
- Since you can already check for that condition with the `==` operator, you are meant to override the `equals()` method with one that will check to see if the two objects (explicit and implicit) have the same fields (or some other definition of equality).

Object class: toString method

```java
public String toString()
```

- Object’s `toString()` method returns the name of the class of the object plus an @ sign and a number representing the object.
- You should override `toString()` to return a String that more closely represents the object.
- Whenever you print an object, the result of method `toString()` is what will be printed.

Introduction to Polymorphism

- Polymorphism
  - “Program in the general”
  - Treat objects in same class hierarchy as if all are superclass objects
  - Abstract class (next session)
  - Common functionality
  - Cannot instantiate
  - Makes programs extensible
  - New classes added easily, can still be processed
Superclass / Subclass Relations

- Try to reference a superclass object with a subclass variable:
  - For example:
    - Circle c = new GeometricObject();
    - Is a GeometricObject a Circle?

- No, a superclass object is not necessarily subclass object:
  - For example:
    - A GeometricObject is a GeometricObject
    - A GeometricObject is not a Circle (i.e. a Circle is more specialized GeometricObject)
    - Does not have an "is-a" relationship.

Superclass / Subclass Relations

- Try to reference a subclass object with a superclass variable:
  - For example:
    - GeometricObject g = new Circle();
    - Is a Circle a GeometricObject?

- Yes, a subclass object is a superclass object:
  - For example:
    - A Circle is a GeometricObject (i.e. a Circle is more specialized GeometricObject)
    - A Circle is a Circle
    - Does have an "is-a" relationship.

Limitation of using superclass reference

- You cannot call a subclass’ method using a superclass reference if the method has not been inherited from the superclass.
  - For example:
    - Object o = new Circle(6);
    - o.getArea()
    - Causes a compile error
Polymorphism Examples

- **Examples**
  - Suppose `Rectangle` derives from `Quadrilateral`
    - `Rectangle` more specific than `Quadrilateral`
    - Any operation on `Quadrilateral` can be done on `Rectangle` (i.e., perimeter, area)
  - Suppose designing video game
    - Superclass `SpaceObject`
      - Subclasses `Martian`, `SpaceShip`, `LaserBeam`
      - Contains method `draw`
    - To refresh screen
      - Send `draw` message to each object
      - Same message has "many forms" of results

- **Video game example, continued**
  - Easy to add class `Mercurian`
    - Extends `SpaceObject`
    - Provides its own implementation of `draw`
  - Programmer does not need to change code
    - Calls `draw` regardless of object's type
      - `Mercurian` objects "plug right in"