

Inheritance and Polymorphism

(2004.06.06)

Inheritance

- basing a new class on an existing class
- all classes are based on an existing class
- classes can be given an explicit parent with *extends*

```
class class-name extends parent-class {  
    ...  
}
```

Inheritance (continued)

- **classes created without an explicit parent are given class *java.lang.Object* as parent**
- **example:**

```
class class-name { ... }
```

means:

```
class class-name extends java.lang.object {  
    ...  
}
```

<http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Object.html>

Inheritance Terms

- class - a data type
- extend - make a new class based on the inherited contents of an existing class
- superclass - parent or base class; class above a class; inherited class
- subclass - child class; class that inherits

Single Inheritance

- **Java uses single inheritance, i.e. only one parent allowed per class**
- **multiple inheritance partially supported via *implements* (to be covered in chapter 8)**

Inheritance, Practically Speaking

- public and protected members of the parent class become members of the child class
- private members are not inherited
- public members are inherited as public members
- protected members are inherited as protected members
- *super* calls members of the the parent (not private)

super() // calls parent constructor

super.*member* // calls parent member

Inheritance Example

```
public class Parent {
    public    int p_pub;
    protected int p_prot;
    private  int p_priv;

    public Parent() {
        p_pub  = 1;
        p_prot = 2;
        p_priv = 3;
    }

    public void m_pub() {
        System.out.println("m_pub");
    }

    protected void m_prot() {
        System.out.println("m_prot");
    }

    private void m_priv() {
        System.out.println("m_priv");
    }
}
```

Inheritance Example (continued)

```
public class Child extends Parent {  
  
    public Child() {  
        super();  
  
        System.out.println(String.valueOf(p_pub)); // 1  
        System.out.println(String.valueOf(p_prot)); // 2  
  
        m_pub(); // m_pub  
        m_prot(); // m_prot  
    }  
}
```

Name Collisions

- **If child class contains a field identifier named the same as a parent field identifier, the parent field is hidden.**
- **If a child class contains a method with a signature the same as the signature of a method of the parent class, the parent method is hidden. (Hence, use of *super.*)**

A Second Inheritance Example

```
public class Parent {
    protected int i;

    public Parent() {
        i = 1;
    }

    public void m_pub() {
        System.out.println(String.valueOf(i));
    }
}
```

A Second Inheritance Example (continued)

```
public class Child extends Parent {
    protected int i;

    public Child() {
        super();
        i = 2;
        System.out.println(String.valueOf(super.i)); // 1
        m_pub(); // 1
        System.out.println(String.valueOf(i)); // 2
    }
}
```

Polymorphism

- **name reuse**
- **two forms**
 - **overloading**
 - **overriding**

Overloading Polymorphism

- **multiple methods with the same name**
- **differentiated on signature (name; then number, type and order of parameters)**
- **resolved at compile time**
- **example:**

```
public int Sum(int a, int b) {  
    return a + b;  
}
```

```
public int Sum(int a, int b, int c) {  
    return a + b + c;  
}
```

Overriding Polymorphism

- a method in a sub class has the same signature as a member of the superclass, i.e. a method in sub class (derived class) overrides the method in the superclass

Note: method in the sub class can become more public, not more private

Example of Overriding Polymorphism

```
public class Parent {  
  
    protected int i;  
  
    public Parent() {  
        i = 1;  
    }  
  
    public void m_pub() {  
        System.out.println(String.valueOf(i));  
    }  
}
```

Example of Overriding Polymorphism (continued)

```
public class Child extends Parent {  
  
    protected int j;  
  
    public Child() {  
        super();  
        j = 42;  
        super.m_pub();           // outputs:      1  
        m_pub();                // outputs:      42  
    }  
  
    public void m_pub() {  
        System.out.println(String.valueOf(j));  
    }  
  
}
```

Preventing Overriding: *final*

- *final* classes cannot be inherited

example: **public final class Bowling {**

- *final* methods cannot be overridden

example: **public final Sum(int a, int b);**

Forcing Overriding: Abstract

- ***abstract* classes**
 - **must be inherited**
 - **cannot be used to create objects**
 - **must contain at least one *abstract* method**
 - **example:**

```
public abstract class Bowl {
```

Forcing Overriding: Abstract (continued)

- ***abstract* methods must be overridden**

example:

```
public abstract int Sum(int a, int b);
```

- **any class inheriting the class that has the abstract *Sum* as a method must create a method *Sum* with the same signature**

example:

```
public int Sum(int a, int b) {  
    return a + b;  
}
```

Object

- **root class of Java**
- **all classes in Java descend from class *Object***
- **i.e.** `class Example { ...`

really means

`class Example extends Object { ...`

<http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Object.html>

Object Method *toString*

- returns a string representation of any object
- **syntax:** `String toString()`
- very useful function
- may want to override on occasion

Example of not Overriding *toString*

```
public class Parent {
    protected int i;

    public Parent() {
        i = 1;
    }

    public void m_pub() {
        System.out.println(String.valueOf(i));
    }
}

public class Child extends Parent {
    protected int j;

    public Child() {
        super();
        j = 42;
    }
}
```

Example of not Overriding *toString* (continued)

- **Use**

```
Child c = new Child();
```

```
System.out.println(c.toString());
```

- **Output:** **Child@a18aa2**

Example of Overriding *toString*

```
public class Child extends Parent {  
  
    protected int j;  
  
    public Child() {  
        super();  
        j = 42;  
    }  
  
    public String toString() {  
        return ("i=" + i + ", j=" + j);  
    }  
  
}
```

Example of Overriding *toString* (continued)

- **Use**

```
Child c = new Child();
```

```
System.out.println(c.toString());
```

- **Output: i=1, j=42**