Lecture 21: Inheritance and Polymorphism

CS 170, Section 000
12 November 2009
Lecture Plan

- Logistics

- More on Inheritance:
  - constructor chaining, casting, more examples

- Polymorphism
Logistics

- **Homework 1-5**: solutions posted
- **Labs 1-10**: solutions posted
- **HW6**: solution to be posted by tomorrow (Fri)

- **Midterm 2**: Tuesday, Nov 17th
  - **Review Session**: Monday Nov 16, 4:15-6pm, W301
  - **Practice exam**: will try to circulate by tomorrow (Fri)
Inheritance – basic idea

GeometricObject
- color
- isFilled

Triangle
- sideLength

Circle
- radius

Rectangle
- width
- height
Superclasses and Subclasses

<table>
<thead>
<tr>
<th>GeometricObject</th>
</tr>
</thead>
<tbody>
<tr>
<td>- color: String</td>
</tr>
<tr>
<td>- filled: boolean</td>
</tr>
<tr>
<td>- dateCreated: java.util.Date</td>
</tr>
<tr>
<td>+ GeometricObject()</td>
</tr>
<tr>
<td>+ getColor(): String</td>
</tr>
<tr>
<td>+ setColor(color: String): void</td>
</tr>
<tr>
<td>+ isFilled(): boolean</td>
</tr>
<tr>
<td>+ setFilled(filled: boolean): void</td>
</tr>
<tr>
<td>+ getDateCreated(): java.util.Date</td>
</tr>
<tr>
<td>+ toString(): String</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Circle</th>
</tr>
</thead>
<tbody>
<tr>
<td>- radius: double</td>
</tr>
<tr>
<td>+ Circle()</td>
</tr>
<tr>
<td>† Circle(radius: double)</td>
</tr>
<tr>
<td>+ getRadius(): double</td>
</tr>
<tr>
<td>+ setRadius(radius: double): void</td>
</tr>
<tr>
<td>+ getArea(): double</td>
</tr>
<tr>
<td>+ getPerimeter(): double</td>
</tr>
<tr>
<td>+ getDiameter(): double</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rectangle</th>
</tr>
</thead>
<tbody>
<tr>
<td>- width: double</td>
</tr>
<tr>
<td>- height: double</td>
</tr>
<tr>
<td>+ Rectangle()</td>
</tr>
<tr>
<td>+ Rectangle(width: double, height: double)</td>
</tr>
<tr>
<td>+ getWidth(): double</td>
</tr>
<tr>
<td>+ setWidth(width: double): void</td>
</tr>
<tr>
<td>+ getHeight(): double</td>
</tr>
<tr>
<td>+ setHeight(height: double): void</td>
</tr>
<tr>
<td>+ getArea(): double</td>
</tr>
<tr>
<td>+ getPerimeter(): double</td>
</tr>
</tbody>
</table>
Inheritance

• A subclass inherits all fields and methods from the superclass
• A subclass can also:
  – Add new fields
  – Add new methods
  – Override the methods of the superclass

• Superclass’s constructor are not inherited
  – Unlike fields and methods
• They are invoked explicitly or implicitly
Using the Keyword `super`

- `super` refers to the superclass of the class in which `super` appears
- This keyword can be used in few ways:
  - To call a superclass constructor
  - To call a superclass method
  - To access a superclass public data field
Invoking Superclass Constructor

• Superclasses’ constructors can (only) be invoked from subclasses' constructors explicitly
• Use the keyword `super` to call the superclass constructor
• Java requires that the statement that invokes superclass’ constructor using the keyword `super` appear first in the constructor
Superclass’s Constructor Is Always Invoked

• If no superclass constructor is explicitly invoked, the compiler puts `super()` as the first statement in the constructor

```
public A() {
}
```

is equivalent to

```
public A() {
    super();
}
```

```
public A(double d) {
    // some statements
}
```

is equivalent to

```
public A(double d) {
    super();
    // some statements
}
```
Constructor Chaining

public class Faculty extends Employee {
    public static void main(String[] args) {
        Faculty f = new Faculty("Brad Pitt");
    }

    public Faculty(String name) {
        System.out.println("(3) Faculty's constructor");
    }
}

class Employee extends Person {
    public Employee() {
        System.out.println("(2) Employee's no-arg constructor");
    }
}

class Person {
    public Person() {
        System.out.println("(1) Person's no-arg constructor");
    }
}
public class Faculty extends Employee {
    public static void main(String[] args) {
        Faculty f = new Faculty(“Brad Pitt”);
    }
    public Faculty(String name) {
        super();
        System.out.println("(3) Faculty’s constructor");
    }
}

class Employee extends Person {
    public Employee() {
        super();
        System.out.println("(2) Employee's no-arg constructor");
    }
}

class Person {
    public Person() {
        super();
        System.out.println("(1) Person's no-arg constructor");
    }
}
Why bother with super()?

- Good practice: define default constructor
- Only the default constructor is called automatically
  - If need to call super(name), must do so explicitly:
    
    ```java
    public Faculty(String name) {
        super(name);
        ...
    }
    ```
Overriding Methods in the Superclass

- A subclass inherits methods from a superclass
- Subclass can modify the implementation of a method defined in the superclass.
- *Method overriding.*

```java
public class Circle extends GeometricObject {
    // Other methods are omitted

    /** Override the toString method defined in GeometricObject */
    public String toString() {
        return super.toString() + "\nradius is " + radius;
    }
}
```
NOTE

• An instance method can be overridden only if it is accessible
• A private method cannot be overridden, because it is not accessible outside its own class
Overriding vs. Overloading

```java
public class Test {
    public static void main(String[] args) {
        A a = new A();
        a.p(10);
    }
}

class B {
    public void p(int i) {
    }
}

class A extends B {
    // This method overrides the method in B
    public void p(int i) {
        System.out.println(i);
    }
}
```

```java
public class Test {
    public static void main(String[] args) {
        A a = new A();
        a.p(10);
    }
}

class B {
    public void p(int i) {
    }
}

class A extends B {
    // This method overloads the method in B
    public void p(double i) {
        System.out.println(i);
    }
}
```
Review: Zoo example (from lab 10)

- /home/cs170000/inclass/nov12/zoo/
Another Example: Person, Student

- Student \textit{extends} Person

- Code: inclass/nov12/student/
Review questions

• Which of the following statements are true?

A. A subclass is a subset of a superclass.
B. A subclass is usually extended to contain more functions and more detailed information than its superclass.
C. "class A extends B" means A is a subclass of B.
D. "class A extends B" means B is a subclass of A.
Review questions

• Which of the following statements are true?

A. A method can be overloaded in the same class.
B. A method can be overridden in the same class.
C. If a method overloads another method, these two methods must have the same signature.
D. If a method overrides another method, these two methods must have the same signature.
Object: The Cosmic Superclass

- All classes defined without an explicit `extends` clause automatically extend `Object`
Object: The Cosmic Superclass

- Most useful methods:
  - `String toString()
  - `boolean equals(Object otherObject)
- Good idea to override these methods in your classes
The `toString()` method in `Object`

- The `toString()` method returns a string representation of the object.
- The method is called whenever the object is converted to a string.

```java
BankAccount account = new BankAccount();
System.out.println(account); // BankAccount@15037e5
```

- The default implementation returns a string consisting of a class name of which the object is an instance, the at sign (`@`), and a number representing this object.
Overriding the `toString` Method

- To provide a nicer representation of an object, override `toString()`.
  ```java
  public String toString()
  {
      return "BankAccount [balance=\" + balance + \"]";
  }
  ```

- This works better:
  ```java
  BankAccount momsSavings = new BankAccount(5000);
  System.out.println(momsSavings);
  //BankAccount [balance=5000]
  ```

- More examples: Card.java
The equals Method

• The equals() method compares two objects.
• The default implementation of the equals method in the Object class is as follows:

```java
public boolean equals(Object obj) {
    return (this == obj);
}
```

- Java classes such as `String` override equals() method so that it compares the content of two objects.
- It is a good idea to override equals() method for your own classes.
public class Circle {

    private double radius;

    public Circle(double r) {
        radius = r;
    }

    public boolean equals(Object o) {
        Circle other = (Circle)o;
        return other.radius == radius;
    }
}

Overriding equals method
Review questions

• What is the output of the following code:

```java
public class Test {
    public static void main(String[] args) {
        Object o1 = new Object();
        Object o2 = new Object();
        System.out.print((o1 == o2) + " " + (o1.equals(o2)));
    }
}
```

A. false false  
B. true true  
C. false true  
D. true false
Review questions

What is the output of the following code:

```java
public class Test {
    public static void main(String[] args) {
        String s1 = "Java";
        String s2 = "Java0".substring(0, 4);
        System.out.println((s1 == s2) + " " + (s1.equals(s2)));
    }
}
```

A. false false
B. true true
C. false true
D. true false
Converting Between Subclass and Superclass Types

- Ok to convert subclass reference to superclass reference
- Need cast to convert from a superclass reference to a subclass reference
  - This cast is dangerous: if you are wrong, an exception is thrown
TIP

• To help understand casting, consider the analogy of fruit, apple, and orange.
• Fruit class is the superclass for Apple and Orange.
• An apple is a fruit, so you can always safely assign an instance of Apple to a variable of Fruit.
• However, a fruit is not necessarily an apple, so you have to use explicit casting to assign an instance of Fruit to a variable of Apple.
Casting

- Casting from a superclass to a subclass is dangerous: if you are wrong, an exception is thrown.
- Use the `instanceof` operator to test whether an object belongs to a particular type before casting.

```java
object instanceof TypeName
```

**Example:**
```java
if (anObject instanceof Circle) {
    Circle c = (Circle) anObject;
    ...
}
```

**Purpose:** To return `true` if the `object` is an instance of `TypeName` (or one of its subtypes), and `false` otherwise.
Converting Between Subclass and Superclass: Examples

• What happens here?

```java
public static void main(String[] args) {
    GeometricObject1 obj1 = new Circle4(1);
    GeometricObject1 obj2 = new Rectangle1(2,4);

    Rectangle1 r = (Rectangle1) obj2; //?
    GeometricObject1 obj3 = r; //?
    obj3.getPerimeter(); //?
}
```
Polymorphism and Dynamic Binding

- Method calls are determined by type of actual object, not type of object reference.
- Suppose an object $o$ is an instance of classes $C_1, C_2, \ldots, C_{n-1},$ and $C_n,$ where $C_1$ is a subclass of $C_2,$ $C_2$ is a subclass of $C_3,$ $\ldots,$ and $C_{n-1}$ is a subclass of $C_n.$
- In Java, $C_n$ is the Object class. If $o$ invokes a method $p,$ the JVM searches the implementation for the method $p$ in $C_1,$ $C_2,$ $\ldots,$ $C_{n-1}$ and $C_n,$ in this order, until it is found.

Since $o$ is an instance of $C_1,$ $o$ is also an instance of $C_2, C_3, \ldots, C_{n-1},$ and $C_n.$
Polymorphism

- PolymorphismDemo.java
  - Which toString() method to call is determined by the object
  - What’s the output?
Polymorphism (2)

What's the output?

```java
public static void main(String[] args) {
    GeometricObject1 obj1 = new Circle4(1);
    GeometricObject1 obj2 = new Rectangle1(2,4);
    System.out.println(obj1.toString());
    System.out.println(obj2.toString());
}
```
Review questions

• Given the following code:

```java
class C1 {}
class C2 extends C1 {}
class C3 extends C2 {}
class C4 extends C1 {}

C1 c1 = new C1();
C2 c2 = new C2();
C3 c3 = new C3();
C4 c4 = new C4();
```

Which of the following expressions evaluates to false?

A. c1 instanceof C1
B. c2 instanceof C1
C. c3 instanceof C1
D. c4 instanceof C2
The ArrayList

ArrayList class that can be used to store an unlimited number of objects.

TestArrayList.java

```
java.util.ArrayList

+ArrayList()
+add(o: Object) : void
+add(index: int, o: Object) : void
+clear(): void
+contains(o: Object): boolean
+get(index: int) : Object
+indexOf(o: Object) : int
+isEmpty(): boolean
+lastIndexOf(o: Object) : int
+remove(o: Object): boolean
+size(): int
+remove(index: int) : Object
+set(index: int, o: Object) : Object

```

Creates an empty list.

Appends a new element o at the end of this list.

Adds a new element o at the specified index in this list.

Removes all the elements from this list.

Returns true if this list contains the element o.

Returns the element from this list at the specified index.

Returns the index of the first matching element in this list.

Returns true if this list contains no elements.

Returns the index of the last matching element in this list.

Removes the element o from this list.

Returns the number of elements in this list.

Removes the element at the specified index.

Sets the element at the specified index.
Practice questions

• Inheritance
  – 10.1, 10.3, 10.6, 10.7, 10.9

• Object class
  – 10.12

• Polymorphism
  – 10.13

• ArrayList class
  – 10.17