CSCI 112
Thursday, July 8, 2010
Class Notes

Review of Last Class

- New class can be derived from current classes using "**extends**."
- Similar terms: base class, **parent class,** super class
- Similar terms: inherited class, **child class**, derived class
- Child Class must hold an "**is-a**" relationship with the parent
- A class can have multiple constructors.
- A constructor with no parameters is the **default constructor**
- To keep a class field private to outside classes, but public to derived classes, make the access modifier "**protected."**
- In a child class, to call the parent's constructor, we use the keyword **super**.
- Java is a single inheritance language, meaning that a class can only be derived from at most one parent class. C++ is a multiple inheritance language, meaning that a class can be derived from multiple classes.

Overriding Methods

When creating a subclass, that class inherits all the methods of the parent class. Sometimes when we create a subclass we want to change the methods that are inherited. We can do this be overriding them. When a method is called, first Java will check all of the methods in the current class and attempt to execute the best match. If none are found, it moves up to the parent class and checks those methods. This process continues until it reaches the topmost class. If none are found in that class, the program errors out.

Example (TestQuote.java):

```
class Quote{
	public void message(){
		System.out.println( "Whatever you are, be a good one. -Abraham Lincoln" );
	}
}
class NewQuote extends Quote{
	public void message(){
		System.out.println( "A nickel ain't worth a dime anymore. -Yogi Berra" );
	}
}
class TestQuote{
	public static void main( String[] args ){
		Quote q = new Quote();
		NewQuote p = new newQuote();
```

```
                q.message();
                p.message();
        }
}
```

Sometimes you don't want a method to be overridden. You show this by using the keyword **final:**

```
public final void message(){}
```

Use this if you're certain that the implementation you create will never need to be overridden.

Abstract Classes

- **Abstract Class:** represents a concept object (i.e. there is no such thing as an animal, there are types of animals).
  - In order to create an abstract class, you prefix it with the keyword abstract.

By itself this isn't very useful, but it has properties that can be passed on to other classes. Thus, it can be used as a building block. Usually it contains a mixture of non-abstract methods and abstract methods.

Abstract classes are useful for creating templates of work.

- **Abstract Methods:** must be overridden by a child class
  - In order to make an abstract method, you prefix the method with the word **abstract** then after the closing parentheses, you end with a semicolon.

Example Code

Work.java

```
abstract public class Work{
        public void result(){
                System.out.println( "Get Paid!" );
        }
        abstract public void action();
}
```

Teaching.java

```
public class Teaching extends Work{
        public void action(){
                System.out.println( "I am teaching." );
        }
}
```

AbstractTest.java

```
class AbstractTest{
	public static void main( String[] args ){
		Teaching t = new Teaching();
		t.action();
		t.result();
	}
}
```

Class Hierarchies

There is no limit to how many times a class can be derived, so a parent class can have as many child classes as you'd like. Those classes can then have their own child classes that inherit from the entire chain above them. However, it doesn't work the other way around, remember, Java is *single-inheritance*.

So, when creating methods, you want to put the methods that have the broadest possible use as high in the chain as you can. This allows for keeping your code "DRY." It also makes for a cleaner class hierarchy.

There is one object that is the master of all other objects, thus highest on any hierarchy. This is known as **Object.** All objects derive from Object. In Object there are six or seven methods, so every object you make in Java has these six or seven methods.

Example Code

Dull.java

```
class Dull extends Object{ //even if you don't write "extends Object," it is done automatically
	public String toString(){
		return "I'm an object.";
	}
}
```

Designing for Inheritance

- Every child class should hold an "is-a" relationship with its parent
- Child classes should always be more specific than their parent class
- Design a class hierarchy around reuse. Put common, broad use functionality as high in the hierarchy as possible.
- Override methods to tailor the functionality of currently written methods
- Avoid using old variable names from class to class
- Each class should be responsible for maintaining its own data.
- It is a good idea to override "toString," even if you don't plan on using it
- Abstract classes are concept classes. They aren't actually created, but they contain functionality

that will be shared among child classes.
- Use access modifiers to prevent breaking encapsulation.

Polymorphism

- **“poly”** → Latin term for 'many'
- **“tics”** → Blood sucking creatures
- **“morph”** → 'form'
- **“polymorphic”** → the ability for a data type to hold many forms

Say we have objects **A, B,** and **C**. If **B** inherits from **A**, and **C** from **B**, **A** can store references from **A**, **B** can store references from **A** and **B**, and **C** can store references from **A**, **B**, and **C**.

- **Polymorphism:** A reference may point to any data type based on its own class of derived from that class.

Example Code

PolyTest.java

```
class Parent{
	public void say1(){
		System.out.println( "Parent" );
	}
	public void say2(){
		System.out.println( "Still parent" );
	}
}
class Child extends Parent{
	public void say1(){
		System.out.println( "Child" );
	}
	public void say3(){
		System.out.println( "Still child" );
	}
}
class PolyTest{
	public static void main( String[] args ){
		Parent p = new Parent(); //see "*Late Binding"
		Child c = new Child();

		p.say1(); // Prints "Parent" to the screen
		p.say2(); // Prints "Still parent" to the screen

		c.say1(); // Prints "Child" to the screen
		c.say2(); // Prints "Still parent" to the screen
		c.say3(); // Prints "Still child" to the screen
```

```
                Parent x = new Child(); //see "*Rules for Polymorphism"

                x.say1(); // Prints "Child" to the screen
                x.say2(); // Prints "Still parent" to the screen
                // x.say3(); // Compile error!
        }
}
```

***Late Binding**

Late Binding is in Java so that you can change the binding of something is Java to something else, necessary for polymorphism.

***Rules For Polymorphism**

- Parent thinks it's a parent class.
- Methods overridden by the child are overridden.
- New methods in the child class are not accessible