

COMPUTER PROGRAMMING IN JAVA

COLUMBIA UNIVERSITY HIGH SCHOOL SCIENCE HONORS PROGRAM

2007 Mar 24 Sat

1. Review

Object Oriented Programming basics

- Classes
- Objects/instances

Basic software principles

- abstraction
- encapsulation
- modularity
- reuse

2. Constructors

Constructors are a special type of method. A constructor has the following properties:

- 1) The name of the method is the same as the name of the class,
- 2) The method has no return type; instead, it returns a reference to a new instance of the class.

For example:

```
public class Pie
{
    protected String typeOfPie = null;

    /**
     * This is an EMPTY constructor
     */
    public Pie()
    {
        this.typeOfPie = "generic";
    }

    /**
     * This is a constructor that allows you to specify
     * the type of pie at initialization time.
     */
    public Pie(String newType)
    {
        this.typeOfPie = newType;
    }
}
```

```

    }

    public String toString()
    {
        return this.typeOfPie;
    }
}

```

```

Pie applePie = new Pie("apple");
Pie genericPie = new Pie();

```

```

System.out.println(applePie.toString());
System.out.println(genericPie.toString());

```

2. Inheritance

The OO programming paradigm allows for a technique called inheritance. Inheritance is exactly what it sounds like: you have some “thing” which is somehow related to another “thing” in such a way that the “properties” of one is “passed down” to the other.

This is a very natural concept that everybody intuitively grasps. Consider any hierarchy, let's say of animals. We'll start at mammals. Represent this as a class, “Mammal”. What are some properties of all mammals? Now name some specific mammals. Are there any properties that they share? [Discuss]

In OO terminology, classes can inherit from other classes. A class can be both a parent class or a child class (synonyms: superclass and subclass). [Construct a class hierarchy together]

At the root of the Java class hierarchy is the *Object* class. All other classes implicitly inherit from *Object* if they don't explicitly inherit from another class.

Methods and data members are also inherited (assuming you declare it correctly; we'll get into this later). Inheritance has a slightly different meaning in each case: when a method is inherited, it means that all subclasses automatically inherit this method. That is, you can do the following [example]. When a data member is inherited, it means that this variable automatically exists in every instance of that subclass (important: remember what it means to have multiple instances of a class!).

Now how do references work with inheritance? Answer: any reference can refer to any instance that is the same type as it, or if any of the subclasses of that instance match the type of the reference.

In Java, we are restricted to single inheritance: that is, any class can only inherit one class (C++ for example does not have this restriction; it (in)famously supports multiple

inheritance). To provide the same functionality and flexibility as multiple inheritance, Java has the concept of interfaces, which we will also get into later.

OVERRIDE

3. Polymorphism

The other major concept OO brings to the table is polymorphism. Polymorphism is essentially the ability for the system to dynamically (at runtime) determine what method to call. [example]

Polymorphism is a very powerful concept whose use often results in clean, easier to debug, and easier to maintain code.

4. Assignments

Easy

For this assignment you will finish implementing a class hierarchy. First, download the following classes:

- Shape.java
- Triangle.java
- Rectangle.java
- Circle.java

By examining the classes you will see that *Shape* is the parent class of *Triangle*, *Rectangle*, and *Circle*.

Next, do the following:

- a. Each shape subclass has its own unique parameters: triangles have a base and a height, rectangles have a width and height, and circles have a radius. Added each of these properties to each of the classes (i.e. add them as class data members and provide getters and setters for them).
- b. *Shape* has a method called *calculateArea()*, which currently returns 0. Each subclass should override this method with the correct implementation (i.e. copy the method from *Shape* into each subclass, and then use the properties from the previous step to calculate the area).
- c. You can use *ShapesTest* to test your code.

Widgets