

COMPUTER PROGRAMMING IN JAVA

COLUMBIA UNIVERSITY HIGH SCHOOL SCIENCE HONORS PROGRAM

2007 Mar 09 Sat

1. Review

Insertion Sort
Selection Sort
Merge Sort

2. Administratum

No class next week, Mar 17. Have a nice spring break (or just a nice Saturday morning)!

2. Object Oriented (OO) Programming: Why?

We've spent our entire time so far programming using methods; why change? Answer: as our programs get larger and more complex, procedural programming (essentially programming using only methods as the organizational unit) starts to become unwieldy. Also, there are many problems for which it is very natural to formulate the problem in terms of objects, each of which have their own properties and state.

3. Old Picture

The universe consisted of a bunch of variables, each in their own scope, and methods, which can call each other. These variables had no properties of their own; we could only call operations on them. If we had a particular, well-defined task that we needed to do, we would collect that code into a method, allowing any other method to call this method to help it do its own task.

4. New Picture: Classes and Objects

The main thing OO brings to the table is polymorphism, but we'll save that topic for next lesson. Today we'll focus more on the basic mechanics of OO programming and a few tangential benefits it brings to the table.

First off, in a programming language which supports the OO paradigm, everything is a class. A class is like a blueprint or a Platonic ideal (for those of you philosophically inclined): it describes how each instance of this class looks like and how it behaves. The Java keyword **new** is how we instantiate instances of a class:

```
Dog dog1 = new Dog();  
Cat cat = new Cat();  
// dog1 and dog2 are different instances of the same class  
Dog dog2 = new Dog();
```

Think about the kinds of variables we've been working with: ints, booleans, and chars:

```
int i = 1;
boolean b = false;
char c = 'm';
```

In the above, “i”, “b”, and “c” are all names of variables *themselves*. However, with objects, like in the example before, “dog1”, “cat”, and “dog2” are references to objects (a synonym for instance). That is, there is the object itself, and then there is a reference to it (think of it as an arrow pictorially). So if we did:

```
Dog dog3 = dog1;
```

We're creating a new reference called “dog3” which points to “dog1”. The compiler knows that “dog1” itself points to an object of type Dog, so therefore “dog3” points to the same instance of Dog.

A class contains two kinds of things: variables (in OO terminology, data members) and methods.

Remembering what we know about scopes, data members are at the class scope; that is, all methods can access their class's data members. Methods we are already familiar with, but note the lack of a “static” in the declaration. We'll talk about exactly what “static” truly means later.

Here is a more complete Dog class:

```
public class Dog
{
    private String name;
    private String color;
    private int age;

    public String getName()
    {
        return this.name;
    }

    public void setName(String newName)
    {
        this.name = newName;
    }

    public void setColor(String newColor)
    {
        this.color = color;
    }

    public String getColor()
    {
        return this.color;
    }

    public int getAge()
    {
        return this.age;
    }

    public void setAge(int newAge)
    {
        this.age = newAge;
    }
}
```

In this class example we have 3 data members, and 6 methods, called getters and setters, which get or set a particular data member. Note the naming convention: {get|set}<data_member_name>.

Having these methods allows us to do the following:

```
Dog dog1 = new Dog();

dog.setName("Alfie");
dog.setColor("white");
dog.setAge(4);

Dog dog2 = new Dog();

dog.setName("Yeller");
dog.setColor("Black");
dog.setAge(5);

System.out.println(
    dog1.getName() + " is a " +
    dog1.getColor() + " dog " +
    "who is " + dog1.getAge() + " years old."
);
System.out.println(
    dog2.getName() + " is a " +
    dog2.getColor() + " dog " +
    " who is " + dog2.getAge() + " years old."
);
```

This is just the tip of the iceberg; there are many more topics within OO programming.

5. Assignments

OO Design Practice 1

Write a dice class, carefully thinking about what kinds of data members and methods it needs. This dice class should have some way to specify the number of sides it has (i.e. a 6-sided die, a 20-sided die, etc). Lastly, include some way to specify the probabilities of each side (i.e. create a biased die, e.g. a 6-sided die with 6 having a probability of 50% instead of the usual 16.66%), either one at a time (set the probability of 4 to 25%, leaving the remaining 75% even distributed among the remaining sides) or all at once (set the probability of a 4-sided die to 10%, 5%, 30%, and 55%, respectively).

OO Design Practice 2

Write a player class for Pig. This player class should contain a name, a score, and the methods to get and set these data members.