1.00 Lecture 7

Java Classes and Objects

Reading for next time: Big Java: sections 2.6-2.11

Classes

- **A class is a pattern or template from which objects are made**
  - **You may have many birds in a simulation**
    - **One bird class (or more if there's more than one type of bird)**
    - **Many bird objects (actual instances of birds)**
    - **Simulation**
- **Objects are instances of classes**
  - **Class: Student Object: Joe Smith**
  - **Class: Building Object: Building 10**
  - **Class: Street Object: Mass Ave**

Class Definition

- **Classes contain:**
  - **Data (members, fields)**
    - **Simple data types, like int or double (e.g. bird weight)**
    - **Objects (e.g. bird beak)**
  - **Methods (functions, procedures)**
    - **Actions that an object can execute (e.g. bird flies/moves)**
- **Classes come from:**
  - **Java class libraries: JOptionPane, Array, Math, etc. There are several thousand classes (Javadoc)**
  - **Class libraries from other sources: Web, fellow students…**
  - **Classes that you write yourself**
- **Classes are usually the nouns in a problem statement (e.g. bird)**
  - **Data members are also nouns (e.g., weight)**
- **Methods are usually the verbs (e.g. flies)**

Building Classes

- **Classes hide their implementation details from the user (programmer using the already-written class):**
  - **Their data is not accessed directly, and the details are not known to 'outside' objects or programs.**
  - **Data is almost always `private` (keyword).**
- **Objects are used by calling their methods.**
  - **The outside user knows what methods the object has, and what results they return. Period. (Usually.)**
    - **The details of how their methods are written are not known to 'outsiders'**
  - **Methods are usually `public` (keyword).**
    - **By insulating the rest of the program from each object's details, it is much easier to build large programs correctly, and to reuse objects from previous work.**
    - **This is called encapsulation or information hiding.**
- **Access: public, private, (package, protected)**

Using an Existing Class

```
public class BusRoute {
   // Data members--private
   private int rteNumber;
   private int passengers;
   private double pctTransfer;
   // Constructor or existence method–public (no return type ever)
   public BusRoute(int r, int p, double pct) {
        rteNumber = r;
        passengers = p;
        pctTransfer = pct;
   }
   // 'Get' methods–respond to messages from other objects--public
   public int getRteNumber() {return rteNumber;}
   public int getPassengers() {return passengers;}
   public double getPctTransfer() {return pctTransfer;}
   public double getConnectionPassengers() {
        return passengers * pctTransfer / 100.0;}
   // 'Set' methods-respond to messages from other objects--public
   public void setPassengers(int i) { passengers = i;}
   public void setPctTransfer(double d) {pctTransfer = d;}
   public void setRteNumber(int i) {rteNumber = i;}
} // 'void' means they don't send a response, just do what's asked
```

Using an Existing Class

```
public class BusTransfer {
   public static void main(String[] args) {
        // Create bus route objects
        BusRoute bus1 = new BusRoute(1, 300, 80.0);  // 'new' keyword
        BusRoute bus2 = new BusRoute(47, 400, 30.0);
        BusRoute bus3 = new BusRoute(70, 500, 50.0);

        // Send messages to the routes asking for connecting psgrs
        double psgr1= bus1.getConnectionPassengers();
        double psgr2= bus2.getConnectionPassengers();
        double psgr3= bus3.getConnectionPassengers();

        // Print out the results
        System.out.println("Route "+ bus1.getRteNumber()+ " "+ psgr1);
        System.out.println("Route "+ bus2.getRteNumber()+ " "+ psgr2);
        System.out.println("Route "+ bus3.getRteNumber()+ " "+ psgr3);

        double totalPsgr= psgr1 + psgr2 + psgr3;
        System.out.println("Total passengers: "+ totalPsgr);
   }
}
```

Exercise, part 1

- Download BusRoute and BusTransfer
- In BusTransfer's main method:
  - Immediately after the three bus routes are created:
    - Get the number of passengers from routes 1 and 47
    - (You can just use the bus1 and bus2 objects directly; you don't have to figure out which routes are numbers 1 and 47)
    - Set these routes' passengers to be 100 more than the current level
- Save, compile and read with the debugger to make sure it's working correctly

Exercise, part 2

- In BusRoute:
  - Add a variable connectingTime to the BusRoute class
    - Choose an appropriate data type for it
  - Change the constructor
    - Add a fourth parameter (call it c)
    - Use c to set the value of connectingTime in the bus route
    - (Make the parameter name different than the variable name)
  - Add 'set' and 'get' methods for connectingTime
    - Use the other 'set' and 'get' methods as a guide
- Save and compile BusRoute (not BusTransfer)
  - Eclipse will give you 3 error message saying your new constructor is undefined in BusTransfer. That's ok for now.
  - You can't run BusRoute to test it yet
    - You could write a main() method in BusRoute strictly as a test method. This is called 'unit test' to check each class in isolation.

Exercise, part 3

- In BusTransfer's main method:
  - Alter the 'new' statements to send another parameter
    - Make the connectingTime 5 for rte 1, 10 for rte 47, 7 for rte 70
    - Parameters must be sent in the order the constructor is expecting them
  - Get the connectingTime from route 70 and print it
    - Do this right after the bus routes are created
- Save, compile and read it with the debugger