Java Programming 2 – Lecture #5 – Jeremy.Singer@glasgow.ac.uk

Classes and Objects

Classes are *types* and objects are *instances* of types. Compare with the `boolean` primitive type, which has `true` and `false` instance values. An example class is `Planet`, with objects such as `mercury`, `venus`, or `skaro`. A class contains *members*, which are either data *fields* or *methods*. The data fields store state that describes some attributes of the object. The methods represent behaviour that processes and transforms the object state. Effectively a class is an abstract description of a set of real-world entities. (This is the object-oriented principle of *abstraction*.) Each instance of a class has data and behaviour associated with it. (This is the object-oriented principle of *encapsulation*.)

Example Case Study

Below is a simple class representing a bank account.

```
public class BankAccount {
  int balance;
  String name;
  int id;
  static int nextId = 0;

  void deposit(int value) { this.balance += value; }
  void withdraw(int value) { this.balance -= value; }
  BankAccount(String name, int initialAmount) {
    this.name = name;
    this.balance = initialAmount;
    this.id = BankAccount.nextId++;
  }
}
```

Object Instantiation

The *constructor* for an object looks like a method that has the same name as the class. The constructor sets up initial values for the data fields to initialize the object state. If no constructor is explicitly defined, then a default *no-args* constructor is automatically included. To invoke the constructor, use the `new` keyword in Java. e.g. `BankAccount b = new BankAccount("test", 0);`

Static members

A static member is associated with a class, rather than with any object created from that class. So for static data fields, there is exactly *one* variable, no matter how many objects of the class have been instantiated. A static method performs a general task for the class, rather than for any specific object. A static method can only access static variables and call static methods in the class. A static method is generally invoked via the class name, rather than via an object reference. e.g. `BankAccount.setInterestRate(0.5);`

Inherited Methods

All objects inherit some methods from the class at the root of the inheritance tree, which is `java.lang.Object`[1]. One such method is `toString()`, which generates a String representation of the object. By default, this String displays the name of the class type and the address in memory of the object. However you can *override* this behaviour by supplying a custom `toString()` definition. Another method inherited by all objects is `equals()` which compares two objects for equality and returns a `boolean` value. Note that the `==` operator implements value equality for primitives, and reference equality for objects (i.e. an object is only equal to itself). A custom `equals()` method allows us to implement some kind of value equality for objects of a specific class.

Here is an example equality test for `BankAccount` objects. We assume that if two `BankAccount` objects have equal `int` ids then the corresponding accounts are equal.

```
public boolean equals(Object o) {
  boolean equal = false;
  if (o instanceof BankAccount) {
    BankAccount b = (BankAccount)o;
    equal = (this.id == b.id);
  }
  return equal;
}
```

Questions

For the `BankAccount` class as defined above, how do we stop client code from resetting the `nextId` static field to allow multiple accounts to share the same id? Also, how do we stop client code from directly modifying the id fields of individual `BankAccount` instances?

[1] See http://docs.oracle.com/javase/tutorial/java/IandI/objectclass.html for details.