Lecture 10:
Objects and Classes in Java

Last time:
1. Pseudo-code (from last lecture)
2. Objects and classes
3. Heaps
4. Garbage Collection

Today:
1. Object equality
2. Objects and classes in Java
3. Methods
What Are

- Objects?
- Classes?
- Stack?
- Heap?
- Reference-type variables?
- `new`?
Equality and Objects

- `==` can be used to compare objects, but result is not always what you expect.
- What is output of following?
  ```java
  String a = new String ("abc");
  String b = new String ("abc");
  if (a == b){
      println ("Equal");
  } else {
      println ("Not equal");
  }
  ```
- Not equal
- ???
== for Reference Values

- `==` compares for identical values
- If `x`, `y` are reference variables, then they contain addresses
- Two addresses are equal if they point to exactly the same thing
- In previous example, `a` and `b` are assigned different addresses because `new` is called twice!
Example

String a = new String ("abc");
String b = new String ("abc");
if (a == b) {
    println ("Equal");
} else {
    println ("Not equal");
}

- Not equal is printed
Contrasting Example

```java
String a = new String ("abc");
String b = a;
if (a == b){
    println ("Equal");
} else {
    println ("Not equal");
}
```

- Equal is printed
- This is called ALIASING: Two variables refer to same object.
- Can be DANGEROUS!!
- What if we really want to make a copy?
  ```java
  String a = "abc"
  String b = new String(a);
  ```
“equals”

- `==` checks if two reference variables refer to the same object
- Methods like `str.equals()` check if two different objects have the same “content”
- Other classes will have an `equals` method also
Classes in Java

- Class declarations have the following form in Java:

  ```java
  public class Student {
    class body: instance variables, methods
  }
  ```

- When you create a class in Eclipse, it generates this template for you.
Anatomy of an Instance Variable Declaration

Visibility modifier

Normal variable declaration

public int IDNum;
Method Declarations in Java

- Methods are operations, like +, ++, etc.
  - They can take inputs
  - They can return values
  - They can modify instance variables
- The form varies slightly depending on whether values are returned or not
Anatomy of a Method Declaration (1)

... for methods that do not return values

```java
public void acceptTokens (int tokensPassedIn) {
    tokenLevel = tokenLevel + tokensPassedIn;
    ...
}
```
“void?”

“Parameter List?”

- If a method does not return a value, use the `void` keyword
- The parameter list describes the form of inputs:
  - type
  - name (for use in body)
- Parameter lists may be empty: ()
- Multiple parameters are separated by: ,
Anatomy of a Method Declaration (2)

... for methods that return values

```
public int lastFour () {
    ... return id % 10000;
}
```
“Return Type?”

- Methods that return values must specify the type of the value to be returned
- The bodies of these methods use `return` to indicate when a value is to be returned
- The value being returned must have the same type as the return type
Example

// "getTokens" returns an integer value corresponding to the number of tokens
public int getTokens () {
    return tokens;
}

Comments

- ... allow you to insert explanations in your code

- Two form:
  - `// blah blah blah`
  - `/* blah blah blah */`

- Every:
  - Class declaration
  - Instance variable declaration
  - Method declaration

must have a comment explaining its purpose!
Example

/* Programmer:  Fawzi Emad and Bonnie Dorr
   Date:13 February, 2007

   This class implements simple operations on students. */

public class Student {

   /* First, instance variables. Each student has these things. */
   public String name;           // Student’s name
   public int IDNum;             // Students University ID number
   public int tokens;            // Number of tokens available for submission

   /* Constructors */
   public Student(String nameDesired, int IDDesired){
      name = nameDesired;
      IDNum = IDDesired;
      tokens = 3;
   }

   /* setters */
   public void setName (int nameDesired) {
      name = nameDesired;
   }

}
Object Creation

- Once a class is defined, objects based on that class can be created using `new`:
  ```java
  new Student();
  ```

- To assign an object to a variable, the variable’s type must be the class of the object:
  ```java
  Student s = new Student();
  ```

- Each object has its own copies of all the instance variables in the class (except for certain kinds we’ll study later):
  ```java
  s.IDNum = 123456789;
  s.setIDNum(123456789);
  ```