Lecture 9: Objects and Classes in Java

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

```java
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” return type

“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

```java
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

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)
- Instance variables and methods in an object can be accessed using `.`
  ```java
  s.IDNum = 123456789;
  s.setIDNum(123456789);
  ```
Constructors

- Special “methods” in class definitions to specify how objects are created
- Form of a constructor definition:
  ```java
  Student (String nameDesired, int IDDesired, int tokensDesired) {
      name = nameDesired;
      id = IDDesired;
      tokenLevel = tokensDesired;
  }
  ```
- Can have more than one constructor, provided argument lists are different
  ```java
  Student (int IDDesired) {
      id = IDDesired;
  }
  ```
- Java includes default constructor (no arguments), which you can redefine (`override`)
  ```java
  Student () {
      tokenLevel = 3;
  }
  ```

Equality Testing

- Need to defined what it means for two students to be equal
- You can define it however you like
  ```java
  public boolean equals (Student otherStudent) {
      return id == otherStudent.id;
  }
  ```

Objects to Strings

- What happens if we try to print a Student object?
  - invoke `println` using a `Student` object as an argument?
    ```java
    Student s1 = new Student ();
    System.out.println (s1);
    ```
  - Something like this prints:
    ```java
    Student@82ba41
    ```
Java Knows “How” To Print Any Object

- Why?
  - Every class has a default toString method
  - toString converts objects into strings
  - System.out.println calls this method to print an object
  - Default: object type and address

- toString can be overridden!

```java
// The method for converting Students to strings
public String toString () {
    return (name + ";" + id);
}
```

Static Data Members and Static Methods

- Not contained in or associated with an object of that type
- Accessed by the ClassName-variableName or by ClassName-methodName
- rather than by objectName-variableName or by objectName.methodName

for Loops

- Three kinds of loops in Java
  - while
  - do ... while
  - for
  - A common programming idiom
    ```java
    int i=0;
    while (i <= 10) {
        j += i;
        i++;
    }
    ```
  - Equivalent for-loop
    ```java
    for (int i=0; i<=10; i++)
        j += i;
    ```
for -Loop Form

for (<init>; <continue>; <increment>) <stmt>
- Equivalent to:
  <init>,
  while (<continue>) {
    <stmt>,
  <increment>;
}
- Any of the three <init> / <continue> / <increment> components may be omitted
  for (;;) {
  }
  Runs forever!

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!

Set / Get Methods

- We have been using = to modify instance variables and accessing variables directly to read values
- Generally, this is not good practice because it imposes restrictions on class implementation
- Better
  set methods to set values (mutators)
  get methods to read values (accessors)
Set Methods (Mutators)

public void setID (int newID) {
    id = newID;
}

- Can also do consistency checking

public void setTokenLevel (int newTokenLevel) {
    if (newTokenLevel <= 3) {
        tokenLevel = newMonth;
    } else {
        System.out.println (
            "Bad argument to setTokenLevel: " + newTokenLevel);
    }
}

Get Methods (Accessors)

- Sole purpose is to return values of state
  public int getID () {
    return id;
  }

- Why use them?
  - The state information may not always be stored in a single instance variable, since implementations may change
  - You give designers option of changing instance variables