## Lecture 6: Objects and Classes

Based on *Introduction to Java Programming*, Y. Daniel Liang, Brief Version, 10/E

### Topics Covered

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Creating Classes and Objects</td>
</tr>
<tr>
<td>2</td>
<td>Constructors</td>
</tr>
<tr>
<td>3</td>
<td>Reference Variables ↔ Objects</td>
</tr>
<tr>
<td>4</td>
<td>Accessing Object’s Data and Methods</td>
</tr>
<tr>
<td>5</td>
<td>Default Values for Data Fields</td>
</tr>
<tr>
<td>6</td>
<td>Using Classes from Java Library</td>
</tr>
<tr>
<td>7</td>
<td>static Variables and Methods, and Constants</td>
</tr>
<tr>
<td>8</td>
<td>Visibility Modifiers: Four P’s</td>
</tr>
<tr>
<td>9</td>
<td>Data Field Encapsulation</td>
</tr>
<tr>
<td>10</td>
<td>Passing Objects to Methods</td>
</tr>
<tr>
<td>11</td>
<td>Arrays of Objects</td>
</tr>
</tbody>
</table>

---
Classes give us a way of defining custom data types and associating data with operations on that appropriate for such data.

1 Creating Classes and Objects

An object represents an entity in the real world. For example: student, table, car, circle, university, book, bookstore.

Objects have:

- **state** represented by **data fields**
- **action/behavior** defined by **methods**

For example:

<table>
<thead>
<tr>
<th>circle has:</th>
<th>student has:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• state</td>
<td>• state</td>
</tr>
<tr>
<td>– radius</td>
<td>– name</td>
</tr>
<tr>
<td>• action/behavior</td>
<td>– id</td>
</tr>
<tr>
<td>– getArea()</td>
<td>– grades</td>
</tr>
<tr>
<td>– getPerimeter()</td>
<td>– address</td>
</tr>
<tr>
<td>– setRadius()</td>
<td>– ...</td>
</tr>
<tr>
<td>– ...</td>
<td>– action/behavior</td>
</tr>
<tr>
<td></td>
<td>– computeGPA()</td>
</tr>
<tr>
<td></td>
<td>– computeMajorGPA()</td>
</tr>
<tr>
<td></td>
<td>– getID()</td>
</tr>
<tr>
<td></td>
<td>– changeAddress()</td>
</tr>
<tr>
<td></td>
<td>– ...</td>
</tr>
</tbody>
</table>

A class is a template for creating objects of the same type. For example, a Circle class can be used to create multiple Circle objects.

A constructor is a special kind of method that is used to construct an object. A class can have multiple constructors (i.e. different ways of creating objects).

See TestSimpleCircle.java for an example of a class.
// Define the circle class with two constructors

class SimpleCircle {
  private double radius;

  /** Construct a circle with radius 1 */
  SimpleCircle() {
    radius = 1;
  }

  /** Construct a circle with a specified radius */
  SimpleCircle(double newRadius) {
    radius = newRadius;
  }

  /** Return the area of this circle */
  double getArea() {
    return radius * radius * Math.PI;
  }

  /** Return the perimeter of this circle */
  double getPerimeter() {
    return 2 * radius * Math.PI;
  }

  /** Set a new radius for this circle */
  void setRadius(double newRadius) {
    radius = newRadius;
  }
}

2 Constructors

A constructor is a special method that is used to construct/create an object. Constructors have to obey several special rules:

- The name of the constructor is always the same as the name of the class, even if multiple constructors are present (uses method overloading).
- There is no return type, not even void.
- The constructor is called automatically when a new operator is used to create an object, ex. Circle c = new Circle();

The constructor that can be called with no arguments is called the default constructor or no-arg constructor. If a class has no constructors explicitly defined, Java provides a default constructor with empty body. It is provided only if NO constructors are defined in the class. Classes do not have to have default constructors, but it is usually a good idea to provide one. (Although there might be good reasons for to have such a constructor in some situations.)
3 Reference Variables ⇔ Objects

The statement

```
ClassName referenceVariable;
```

declares a reference variable that can be used to store a memory address at which the actual object is stored.

In order to create an object, you need to use the `new` operator

```
referenceVariable = new ClassName(...);
```

This is similar to how an array-name stores the memory addresses of the location where the array is stored. The actual array storage needs to be allocated using the `new` operator.

As with all the other declarations/creation statements, the above two lines can be combined into a single statement

```
ClassName objectRefVariable = new ClassName(...);
```

NOTE: observe the parenthesis in the last two statements. This is call to the constructor that may or may not take parameters.

Example The following line of code creates a reference variable, unless it is followed by creation of an actual object, there is no room to store data.

```
Person p;
```

In fact, you should make a habit of always assigning the null value to a reference variable that does not actually reference any valid data:

```
Person p = null;
```

The `new` operator is used to create the actual object and to assign its memory address to a reference variable.

```
p = new Person();
```

And since we do not really care about the hexadecimal value of the memory address (nor do we really have a way of knowing it in Java), you’ll see arrows used to show that a reference variable refers to or “points to” a memory location that contains the actual object.
Anonymous objects. Occasionally, you may want to create an object in memory (usually temporary object) that is not pointed to by any object reference variable. Those objects are called anonymous objects, since they do not have a name. For example

```java
System.out.printf("The area of circle with radius %f is %f. \n",
    5, new Circle(5).getArea() );
```

After this statement executes the object is still in the memory, but there is no way to access it. The memory that it occupies eventually gets reclaimed by Java garbage collection.

Common error Confusing object assignment with reference variable assignment.

```java
Circle c1 = new Circle(5);
Circle c2 = new Circle(17);
c1 = c2;
```

The last statement results in c1 and c2 pointing to the same Circle object in memory (the one with radius 17). It does not copy one Circle object to another.

4 Accessing Object’s Data and Methods

The dot operator or object member access operator is used to access data fields and methods of an object:

- `objectReferenceName.dataField` accesses specific data field in the object
- `objectReferenceName.method()` invokes a specific method on the object

The instance variables and instance methods are the variables and methods that can be accessed/invoked using a specific instance of the class (not using the class name). The object on which an instance method is invoked is called the calling object.

We have been using the dot operator when we used methods in the String class.
Example: The code

```java
String s = "Hello NYU";
System.out.println( s.length() );
System.out.println( s.substring(6, 8) );
```

prints

9
NYU

because `s.length()` returns the number of characters in the string `s` and `s.substring(6, 8)` returns a string that is a substring of `s` starting at index 6 up to index 8.

5 Default Values for Data Fields

When a variable of any kind is created the memory associated with that variable contains some values. The data fields of a newly created object are filled, by default, with zero bits. Depending on the type of the data field, the interpretation of zeroed memory is slightly different.

- numerical types (for both integers and floating point numbers) - set to number zero
- char variables - set to `\u0000`, which is a null character, `'\0'` or `''`
- Boolean variables - set to false
- reference variables - set to `null`, which is an invalid memory address (from the point of view of the running program).

`NullPointerException` occurs if you try to use a reference variable without assigning an object to it first. You will see a lot of these.

6 Using Classes from Java Library

You have already seen some classes that come with Java. We will be using more and more of these.

- Math [http://docs.oracle.com/javase/8/docs/api/java/lang/Math.html](http://docs.oracle.com/javase/8/docs/api/java/lang/Math.html)
- Character [http://docs.oracle.com/javase/8/docs/api/java/lang/Character.html](http://docs.oracle.com/javase/8/docs/api/java/lang/Character.html)
- String [http://docs.oracle.com/javase/8/docs/api/java/lang/String.html](http://docs.oracle.com/javase/8/docs/api/java/lang/String.html)
- Arrays [http://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html](http://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html)
7  **static Variables and Methods, and Constants**

A **static variable** (**class variables**) is shared by all objects of the class. Such variables represent the whole class of objects, rather than one instance of the class (object).

A **static method** (**class method**) is called on a class, rather than a specific object. Static methods do not have access to instance variables.

**static** keyword in the declaration of a variable or method indicates that it is be a static variable/method.

See CircleWithStaticMembers.java

All methods in the **Math** class are static. In fact, you cannot create an instance of that class.

Constants are shared among all instances of the class. They should be declared with **final** and **static** modifiers.

8  **Visibility Modifiers: Four P’s**

**Data fields and methods**

There are four modifiers that can be used with data fields / methods in a class definitions (only one of these can be specified at a time, but the modifiers can be different for different data fields / methods in the same class):

- **public** any other class in the world can access the data fields / methods
- **package** (default, when no other modifier is specified) any other class in the same package can access the data fields / methods
- **protected** only members of the same class and subclasses (to be discussed in chapter 11) can access the data fields / methods
- **private** only the members of the same class can access the data fields / methods

**Constructors** should be public in general. Exception: to prevent users of the class from creating an instance, for example, Math class has a private constructor.

**Classes**

The class itself can have an access modifier. Either **public** or nothing is used with a class. If no modifier is specified, only classes from within the same package can access the class. **Each public class needs to be specified in its own file.**

9  **Data Field Encapsulation**

The data fields should be **private** to the class. Why?

The class should provide

**getters / accessors / get methods**
and

setters / mutators / set methods

to allow for controlled access to the data fields.

10 Passing Objects to Methods

See IncrementExample.java

11 Arrays of Objects

See SortingCircles.java