Activity 4A: Creating Classes I

Reading:
Chapter 4.1 – 4.5 pages 160-181

Terminology:

attribute / state	UML class diagram	calling method
behavior	encapsulation	method declaration
class	client	method invocation
method header	visibility (or access) modifier	return statement
class header	accessor method	parameters
instance variable	mutator method	constructor

Due:

Activity: Monday, September 10, 2012. At the end of the lab

Goals:
By the end of this activity you should be able to do the following:
- Create a class with methods that accept parameters and return a value
- Understand the constructor and the toString method of a class
- Know how to generate a UML diagram and documentation in jGRASP

Program Description:
In this activity you will create a class called ProfileInformation, which will hold basic information about a person.

In-lab Directions:

Part 1: Create Project and Generate UML Class Diagram
- Create your ProfileInformation class and add the following comments (there will be no main method in this class):

```
public class ProfileInformation {
  // declare instance variables here
  // constructor
  // methods
  // toString method (for String output)
}
```

- **Instance variables represent all of the information that an object of the class needs to store**. Declare the following variables:
  - firstName: **String** that will later reference the person's first name
  - lastName: **String** that will later reference the person's last name
  - location: **String** that will later reference the person's location
  - age: **int** that will contain the person's age
  - status: **int** representing whether the person is online or offline

  Hint: Use the private *access modifier* so that values cannot be changed from outside of the object:
  `private String firstName;`

- **Constants store values that cannot be changed**. Use constants to represent the two possible statuses of the user.
  ```
  private static final int OFFLINE = 0, ONLINE = 1;
  ```

- **The toString method returns a String representation of the object**. Create and return a local variable called output with all of the profile information:

  ```
      public String toString() {
        String output = "Name: " + firstName + " "
           + lastName + "\n";
        output += "Location: " + location + "\n";
        output += "Age: " + age + "\n";
        output += "Status: " + status;
        return output;
     }
  ```

  After you have successfully compiled your class, open the *interactions* pane in jGRASP. Create a new `ProfileInformation` object using the new operator and print it out.

  ```
      ProfileInformation p = new ProfileInformation();
      System.out.println(p);
      Name: null null
      Location: null
      Age: 0
      Status: 0
  ```

  Because the String values haven't been initialized they have a value of null. In order to avoid this, you will add a constructor to initialize your instance variables. In the example above, you can also see the value p by just entering `p` in interactions rather than `System.out.println(p)`.

- **The constructor includes code that will be run when the object is first created**. It should be placed before the methods in the class. The constructor *does not have a return type*, and always has the *same name as the class*. When a profile is created, the constructor will take two parameters as input: first name and last name.

  ```
     public ProfileInformation(String first, String last){

    }
  ```

  In the constructor, store the first and last name in the appropriate instance variables:

  ```
          firstName = first;
          lastName = last;
  ```

  Because you don't have inputs for age, location, and status, you'll have to set those to default values:

  ```
          location = "Not specified";
          age = 0;
          status = OFFLINE;
  ```

Compile your program, open up the interactions pane, and create a new profile with the following input. Now that your constructor has stored your values, you should get the following output:

```
ProfileInformation p = new ProfileInformation("Jane", "Lane");
System.out.println(p);
Name: Jane Lane
Location: Not specified
Age: 0
Status: 0
```

- **The methods of your class describe what your object can do (the behaviors of an object)**. For a profile, you would want to be able to set the location and age off the user. You'd also want to allow the user to change their status. First, create a set method for location:

```
 public void setLocation(String locationName){
   location = locationName;
}
```

Test your program in the interactions pane:

```
ProfileInformation p = new ProfileInformation("Jane", "Lane");
p.setLocation("Auburn");
System.out.println(p);
Name: Jane Lane
Location: Auburn
Age: 0
Status: 0
```

Add a method to change age. This will only change the age if the age is over 0 years. It will return a boolean representing whether the age was set:

```
 public boolean setAge(int ageInYears) {
   boolean isSet = false;
   if(ageInYears > 0) {
      age = ageInYears;
      isSet = true;
   }
   return isSet;
}
```

Add a method to return the age. Notice this method takes no parameters, but returns an int value (instead of void).

```
 public int getAge() {
   return _________;
}
```

Represents what the method returns (an int value).

Finally, add a method to return the location of the user.

```
public ________ getLocation() {
   return ________;
}
```

Add the following two methods to allow the user to log off and on:

```
public void logOff() {
   status = OFFLINE;
}

public void logOn() {
   status = ONLINE;
}
```

The actual values (0 and 1) of offline status and online status are not used in the code because they are declared as constants. This makes code easier to read and modify later. You should also hide the values when printing the class information. Modify the toString method as follows:

```
public String toString(){
   String output = "Name: " + firstName + " "
       + lastName + "\n";
   output += "Location: " + location + "\n";
   output += "Age: " + age + "\n";
   output += "Status: ";
   if(status == OFFLINE) {
       output += "Offline";
   }
   else{
       output += "Online";
   }

   return output;
}
```

Prints Offline or Online rather than 0 or 1 for status.

Test each of your methods in the interactions pane. Your output should be as follows**:**

```
ProfileInformation p = new ProfileInformation("Jane", "Lane");
p
Name: Jane Lane
Location: Not specified
Age: 0
Status: Offline
p.setAge(23);
p.setLocation("Auburn");
p.logOn();
```

```
p
Name: Jane Lane
Location: Auburn
Age: 23
Status: Online
```

Part 2: Create Project and Generate UML Class Diagram

- In addition to writing the class above, you are to create another class as your driver program (Activity4A.java). The main method in Activity4A.java should create several instances of `ProfileInformation` and exercise the methods to test your code. Then you are to create a **jGRASP project file** that includes the ProfileInformation class and the class with your main program – Activity4A. After you have created the project file, you should generate (and layout) the UML class diagram for the project. Since this is your first "project" assignment, it may be easier for you to do it after you have completed the program. However, in the future you should be comfortable adding classes to the project file as you implement them and generating/updating the UML class diagram with each new addition. Below is the “quick” version for creating a jGRASP project.

  1. With Activity4A in the edit window, click the UML button on the toolbar.
  2. In the Add File to Project dialog, click the “Create New Project” button.
  3. In the New Project dialog, click the “Create” button to create a project named Activity4A_Project.
  4. In the In the Add File to Project dialog, click the “Add” button to add Activity4A.java to the project. This should open the UML window, and you should see “Activity4A” in a green rectangle.
  5. Now you need to add the ProfileInformation class to the project. You can do this by dragging ProfileInformation.java from the Browse tab to the UML window.
  6. Your UML class diagram should now have two classes. You can rearrange these by dragging them around on the screen.
  7. Right-click on one of the classes, and select “Show Class Info” to see the class member information in the UML Info tab.
  8. Right-click a red dashed dependency and you see the dependencies in the UML Info tab.
  9. You can also create instances of the class by right-clicking and selecting “Create New Instance”. This places an instance on the workbench.
  10. To invoke a method for an object on the workbench, you can right-click on the object (not the class) and select “Invoke Method”.