Activity 7A: Static Class Members

Due:
Activity (in-lab): Monday, October 15, 2012 by the end of lab

Goals:
By the end of this activity you should be able to do the following:

- ➢ Understand and implement static variables and methods
- ➢ Be able to create, compile, and run a JUnit test file

Description:
For this assignment, you will need to download BankLoan.java from the course website and save it to an appropriate folder.

Directions:

Part 1: Static methods (40%)

- You don't have to read this bullet now, but make sure that you understand it before this week's quiz and project. Three of the properties of static methods:
  - Static methods can be invoked using the name of the class.
  - Static methods can be invoked before an object of the class is instantiated.
  - Static methods cannot access instance data. If the method is called before an object is created, then the instance data does not exist yet. Consider the following examples:
    - The trim method of the String class is an instance method. You have to create a String object before you call the method so that the method knows what to trim:
      ```
      String s = "   Red Sox";
      System.out.println(s.trim());
      Red Sox
      ```
    - The pow method of the Math class is given both values that it needs in the parameter and so it is defined as static so that you don't have to go through the additional steps of creating an object first:
      ```
      System.out.println(Math.pow(2, 3));
      8.0
      ```

  If a method that you create doesn't need to access any instance variable or instance method in that class, then you should consider making the method static.

- Suppose that you want to add a method to BankLoan that returns true if a loan amount is valid and false otherwise. If the loan amount (a double) is taken as a parameter, then the method would not have to access any instance data. Create the following method header:

  ```
  public static _________ isAmountValid(_________ amount)
  ```

  Add code to the method to return true if the amount is greater than or equal to 0 and false otherwise:

  ```
  return amount ___ 0;
  ```

- Make sure that BankLoan.java compiles after you add your isAmountValid method. It will be tested later in this activity.
- Now suppose that you want a method that returns true if a loan's balance is greater than 0 and false otherwise. The user will pass in a BankLoan object as a parameter, so the method won't need to access any instance data and can be static.

```
public static ________ isInDebt(________ loan) {
```

- Now add an if statement that returns true if the specified object has a balance greater than 0:

```
        if (_________.getBalance() > 0) {
           return true;
        }
        return ________;
```

- Compile your program and test it in the interactions pane:

```
    BankLoan b = new BankLoan("Bob", 0.08);
    BankLoan.isInDebt(b)
    false
    b.borrowFromBank(1); // borrow $1.00
    BankLoan.isInDebt(b)
    true
```

Part 2: Static variables (35%) – a.k.a. class variables as opposed to instance variables

- You don't have to read this bullet now, but make sure that you understand it before this week's quiz and project. Properties of static variables / constants:
  - Public constants (which are static) can be accessed using the name of the class if they are public.
  - Public static constants can be accessed before an object of the class is instantiated.
    ```
    System.out.println(Math.PI);
    3.141592653589793
    ```
  - Public static variables (not declared as final) violate encapsulation. Private static variables should be accessed and modified through static <u>or</u> instance methods.
  - A static variable is <u>a single value in memory</u> that can be accessed / modified through any instance of that class. **Why are constants declared as static as well as final?**

  If one object modifies a static variable, then it will be modified for all other objects as well because they are all accessing the same variable (i.e., a class variable). If a variable needs to be specific to one object (such as customer name), then it should <u>not</u> be static.

- Add a static variable to the BankLoan class that will count the number of BankLoan objects that have ever been instantiated in a program. The value will default to 0 when the driver program begins and only change when a loan object is created.

```
private static int loansCreated = 0;
```

- You want to increment the variable only when an object is instantiated, so you'll need to add the following code to the constructor:

```
loansCreated++;
```

- You'll also need to add a method to access the variable. The method will only access a static variable, so it can and should be static as well.

```
     public static ________ getLoansCreated() {
       return loansCreated;
    }
```

- Also add a method that will reset the number of loans created to 0. The method only modifies a static variable, so it can be static.

```
     public static _________ resetLoansCreated() {
       ________ = _________;
    }
```

Test your class in the interactions pane:

```
    BankLoan.getLoansCreated()
    0
    BankLoan jane = new BankLoan("Jane L", 0.09);
    BankLoan bob = new BankLoan("Bob S", 0.09);
    BankLoan.getLoansCreated()
    2
    bob = new BankLoan("Bob Parker", 0.02);
    BankLoan.getLoansCreated()
    3
    BankLoan.resetLoansCreated();
    BankLoan.getLoansCreated()
    0
```

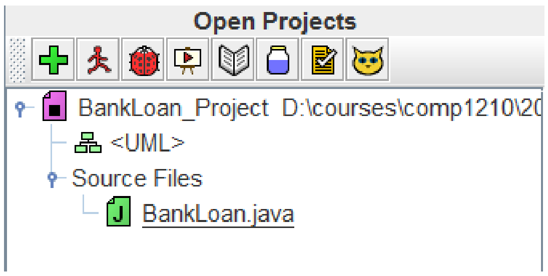
- If you created a static variable that stored the highest loan in creation, it would be modified in the borrowFromBank and payBank methods. A getHighestLoan method could then return the value of the variable. You don't have to create the variable for this activity, but you should try it on your own so that you know where to modify static variables.

Part 3. JUnit Test Files (25%) – NOTE: JUnit is installed on all the computers in the lab. In order to use JUnit on you own computer, you must download and install JUnit (see 1210 web page), and then JUnit must be configured in jGRASP (Tools > JUnit > Configure). The following example will work with the unmodified BankLoan.java file, so you may want to do Part 3 on the lab computer if you don't have JUnit installed on your own machine.

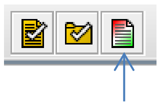
- Now let's create a JUnit test file to allow us to test (and retest) the method in a class. Suppose we want to test the chargeInterest() method. We'll need to create a test method that essentially includes the statements that we would enter as interactions to informally test the chargeInterest() method. For example, we could test this method using interacts by entering the following:

```
BankLoan loan1 = new BankLoan("Jane", .10);
loan1.borrowFromBank(1000.00);
loan1.chargeInterest()
loan1.getBalance()
1100.0
```

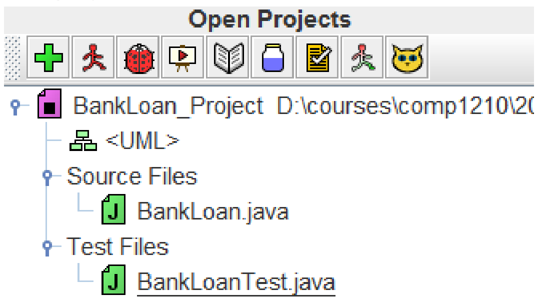
  - Create a project file for the BankLoan class



  - On the Desktop menu click on the Create JUnit Test File button.



  - This creates a JUnit test file name BankLoanTest.java that is listed in the "Test File category in the Project tab.



- BankLoanTest.java contain a class with the same name. The class contains a setup() method and test method named defaultTest() that asserts that 0 equals 1 so it always fails. This is an example that you should either modify or delete.

```
    /** A test that always fails. **/
      @Test public void defaultTest() {
        Assert.assertEquals("Default test added by jGRASP. Delete "
              + "this test once you have added your own.", 0, 1);
     }
```

- Modify the method above or create a new test method as follows. <u>If you create a new method, be sure to delete or comment out the default test method</u>. Note that we won't be using the setup() method, so it can be deleted or left as is since it has an empty body.

```
 @Test public void chargeInterestTest() {
   BankLoan loan1 = new BankLoan("Jane", .10);
   loan1.borrowFromBank(1000.00);
   loan1.chargeInterest();
   Assert.assertEquals(" ", 1100, loan1.getBalance(), .0001);
}
```

Note that the third parameter is the "delta" required when two doubles values are compared for equality (i.e, how close do the values have to be to be considered equal; in this case, we have chosen 0.0001).

- After you have have entered your test method, you can compile the test file or you can compile and run the test file using JUnit buttons on the Project tab or on desktop menu. You should make sure BankLoanTest compiles successfully before you run it.

- A successful test will have output such as the following.

```
 JUnit version 4.8.2
.
Time: 0

OK (1 test)
```

- To test additional methods in the BankLoan class, you simply create additional @Test methods in BankLoanTest.java using the pattern above. Remember creating a test method is only slightly more work that testing your methods in interactions. In fact, many of the same statements used in interactions are used in the test method. These are then followed by an appropriate JUnit assert statement. The beauty of this approach to testing is that (1) as you create new methods in your program, you can also create appropriate test methods; (2) by working to make sure each method passes its test method(s), you can be much more confident that your program is correct; (3) you can re-run all test files with a single click, which is important after you make changes during development.