Due:
Activity (in-lab): Monday, October 17, 2011 by the end of lab
Homework: Wednesday, October 19, 2011 by 11:59 PM on Web-CAT

Goals:
By the end of this activity you should be able to do the following:
- Understand and implement static variables and methods

Description:
For this assignment, you will need the BankLoan class from the course website.

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 have a method in 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 your method compiles. 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%)

- 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. Static variables should be accessed and modified through static or instance methods.
  - A static variable is a single value in memory 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 value. If a variable needs to be specific to one object (such as customer name), then it should not 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 totalLoansCreated = 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:

```
totalLoansCreated++;
```

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

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

- 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() {
  ________ = _________;
}
```

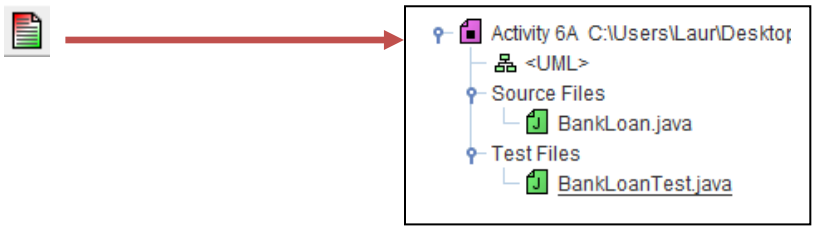
Test your class in the interactions pane:

```
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.

Homework (Due: Wednesday, October 19, 2011 by 11:59 PM on Web-CAT)

- Creating a separate class to carry out testing has advantages over using the interactions pane only:
  - You can create your tests before your method or someone else could write unbiased tests for your code.
  - If you modify another part of the class, you can rerun your old tests to make sure that no bugs were added to existing code. For example, you probably ran the 7A Web-CAT tests before moving to the graded portion of 7B.
- Open your BankLoan class and add it to a new project. Then press the JUnit button in jGRASP:



- Delete the setUp method and defaultTest method. Add a method to test your borrowFromBank and getBalance methods.

```
@Test public void addBalanceTest() {
```

- In the method, create a BankLoan object and add $100 to the balance:

```
BankLoan loan = new BankLoan("Jane", 0.08);
loan.borrowFromBank(100);
```

- The assertEquals method of the Assert class will be used to test your program. The method takes the following parameters:
  - A string representing the error that will be printed if test fails.
  - The value that you expect to get.
  - The actual value that was returned by the method
    - If the last 2 were double values, the method also takes the number of decimals that you want to compare (if you want the test to be accurate to 2 decimals, then you will need 0.01).

- Add the following code to your method.

```
Assert.assertEquals("The getBalance method returned an incorrect "
     + "value after $100 was added.", 100, loan.getBalance(), 0.01);
```

- Add a method to BankLoanTest to test your isAmountValid method.

```
@Test public void amountValidTest() {
```

- If the isAmountValid method is wrong for -1, you want to print the **error message** "isAmountValid failed for value -1.". A value of -1 should be invalid, so the **expected value** is false. The method call to get the **actual value** is BankLoan.isAmountValid().

```
Assert.assertEquals("isAmountValid failed for value -1.",
false, BankLoan.isAmountValid(-1));
```

- Add the following code to the same method:
  - Error message: "isAmountValid failed for value 1."
  - Expected value: true (a loan of 1 is valid)
  - Actual value: BankLoan.isAmountValid(1)
  - There is no 4th parameter since you are testing boolean (not double) values

```
Assert.assertEquals(_________, _________, _________);
```

- Run your test program to make sure that your class passes all of your tests:

```
 ----jGRASP exec: java org.junit.runner.JUnitCore BankLoanTest

JUnit version 4.9b2
..
Time: 0

OK (2 tests)


 ----jGRASP: operation complete.
```

- Submit the entire project (BankLoan and BankLoanTest) to Web-CAT.