Lab 8  Name:__________________________  Checked:______

Objectives:
Practice creating classes and methods, and using them in your programs.

Preparation A: Exercise using the Die class

In your Lab 8 folder, create a subfolder Dice for your files in this exercise.
(Don't have a Lab 8 folder? Then be sure to create one now, complete the Warmup exercise and review lecture notes for this week before you begin.)

For this exercise we will be modifying an example from the text involving the Die class (simulating rolling dice).

1. Download and compile RollingDice.java and Die.java

   * Try running Die.java, note the error you get here:

   ______________________________________________________________
   ______________________________________________________
   Explain what happened:
   ____________________________________________________________

2. Modify RollingDice.java so that it creates a third die and rolls it along with the others

3. Change the toString() method in Die class so that instead of printing just the number showing on the face of the die, it produces a string containing the number in a new line, inside a box, like this:

   +----+
   | 5  |
   +----+

   * Re-compile Die.java
   * Run RollingDice.java to observe the effect of this change.

6. (optional): Challenge: Make the toString() method more realistic, so that it returns a string that looks like an actual die. For example,

   +----+
   | 5  |
   +----+

   instead of this: it could look like this:

   +----------+
   | *   *    |
   | *   *    |
   | *   *    |
   +----------+
Preparation B: Exercise using the Account class

In your Lab 8 folder, create a subfolder Banking for your files in this exercise. We will be modifying the code of the Transactions class to create and use Account objects.

1. Download and compile the Account.java and Transactions.java files. (Do NOT use the ones from the Warmup exercise; be sure to create a new folder and start again with fresh versions downloaded from the course website.)

Run the Transactions class (NOT the Account class) and answer the following questions:

   a) How many Account objects were created by the Transactions class? __________

   b) What were the variables names (Java identifiers) that referred to these objects?
      ________________________________
      ________________________________

   c) Give an example of a statement that was used to print out the information of an Account object:
      _______________________________________________________

2. Examine the code for the Account class.
   o List the names of all the instance variables: ____________________________

   o List the corresponding types (for inst. vars): ____________________________

   o List the names of all the methods (include the constructor):
      ___________________________________________________________________

3. Create a new application named OnePercent.java

   Similar to Transactions.java) using the Account class as follows:
   • Creates an account for someone named "Donald Trump" with $400 as initial balance and account number: 20230715
   • Creates an account for someone named "Bill Gates" with $500 as initial balance and account number 31558040
   • Creates an account for someone named "Elon Musk" with $600 as initial balance and account number 44003050
   • Prints the information for these three accounts

Compile and run OnePercent.java before proceeding. Make sure it prints the information as you expect it.
4. Add more code to `OnePercent.java` to create one more account and to print its information, along with the other accounts' information:

   * Account name: "Uncle Sam"
   * account number: 999999999.
   * Initial balance: $0

Re-compile and run `OnePercent`. Make sure it prints the information as you expect it.

5. Using the `getBalance()` method.
Add some more code to `OnePercent.java` to use the `getBalance()` method of the `Account` class to get the balances of the four accounts and to add them together to obtain the total amount of money in the bank. Print the total and verify that you are getting the right amount (should be $1500).

   Note: Do NOT make any changes to the `Account` class, we are still working in the `OnePercent` class.
   Remember that, in order to display the total as currency, you will need to use the `NumberFormat` class.

6. Change the `toString()` method in the `Account` class.
In the `Account` class, change the code of the `toString()` method so that the string returned also contains the name of the bank (make something up!) and formats the information slightly differently (your design decision here). Recompile `Account` and then run `OnePercent` (no need to recompile `OnePercent` since it should NOT have changed in this step).

**SUBMIT files for preparation through blackboard under Lab 8 prep:**

- Die.java
- RollingDice.java
- Account.java
- OnePercent.java
Part A: Dice – Modifying the Die class
Use the files in the subfolder Dice from the lab preparation.

1. Create a new method in the Die class:
   
   ```java
   public void nudge()
   ```
   The method should cause the die's value to be incremented. So, for example, if the die's value is 3, it should become 4; if it is 6, it should circle back to 1.
   
   *Hint: use an if statement or think of a clever way to use the % operator to do this. The nudge() method should not return any value. Be sure this method contains appropriate comments.*

2. In RollingDice.java add some statements at the end to "nudge" up the values of the three dice and print them again.

Part B: Banking – Modifying the Account class
Use the files in the subfolder Banking from the lab preparation.

1. Create a new method in the Account class, to add interest to an account.
   
   The method should have a single parameter, a double, for the interest rate. The amount added to the balance of the account should be computed according to the rate given by the parameter. For example, if the acct1 balance is $100.00 and the method is invoked as follows:
   
   ```java
   acct1.addInterest(0.015);
   ```
   the balance of acct1 should increase by 1.5% (thus $100 + $1.50 = $101.50 ).
   
   - Be sure to include comments describing the method’s functionality. Use a similar format to the other methods in the Account class.
   - Test your method by invoking it four times to add 3% interest to all the accounts (including Uncle Sam’s!).

   For the next two exercises, recall that Java allows you to define alternative versions of methods using the same method name as long as the different versions also have a different number or different types of parameters. We will now define alternative withdraw() and constructor methods.

2. Add a second version of the withdraw() method.
   
   This version does NOT charge a withdrawal fee, so it has only one parameter. Use this version of the method in OnePercent to withdraw some money from Elon Musk’s account.
   
   *Reminder: The name of this method should still be withdraw(). See note above.*

3. Add another version of the constructor.
   
   This version takes only two parameters: name and account number (i.e., no initial balance).
   
   This constructor creates an Account object with initial balance $0. Modify OnePercent to use this version of the constructor to create the “Uncle Sam” account.
   
   *Reminder: This constructor should still be named Account(). See note above.*
Part C: Implementing your own class and a client to test it

In your Lab 8 folder, create a subfolder Humans for your files in this exercise. We will be writing all the code from scratch.

1. Implement a Person class.
   a) Copy and paste the Java comments below into a new Java file for a Person class (we will use these comments to build the code for the Person class incrementally).
   b) Start by putting in the class heading and the enclosing braces; write the code for the instance variable declarations and implement the constructor and the toString() method.
   c) Compile your class and fix any errors before proceeding.

2. Implement the client (driver class).
   You can call this class PeopleBeingPeople or another name or your choice.

   Be sure to set up the class and the main() method appropriately

Villanova University  CSC 1051  www.csc.villanova.edu/~map/1051  Dr. Papalaskari
3. In the **Person** class, add another constructor that has only one parameter, the name. Modify your client to use this constructor to create an additional **Person** object **friend4** and to print out info about **friend4**.

4. Let’s now add some more methods to the **Person** class. Copy/paste the comments below into your **Person** class and fill in the code as appropriate.

   ```java
   // ----------------------------------
   // birthday(): increases age by one.
   // ----------------------------------
   
   // -------------------------------
   // getAge(): returns the age of this person
   // -------------------------------
   ```

5. Test your methods by adding some code to your client (**PeopleBeingPeople** class).
   
   - Increase the age of **friend4** twice
   - Compute and print the average for the ages of the four friends (i.e., use `getAge()` to obtain the ages of the four friends, add them together and divide by four).

6. Add more methods to the **Person** class. Copy/paste the comments below into your **Person** class and fill in the code as appropriate.

   ```java
   // ------------------------------------------
   // setName(String x): changes the name of this Person to x
   // ------------------------------------------
   
   // ------------------------------------------
   // getName(): returns the name of this person
   // ------------------------------------------
   ```

7. Test your methods by adding some code to your client (**PeopleBeingPeople** class).
   
   - Change someone’s name
   - Print a message that uses the names (just the names, not the other info) of the people.

8. *(optional)* You can also add more instance variables to the **Person** class. For example, try adding a happiness attribute: **boolean happiness**

   - Modify your constructor so that it sets happiness to **true** (that makes it the default value for happiness 😊).
   - Add methods `setHappiness()` and `getHappiness()` similar to `setName()` and `getName()`, above that allow the client code to modify the Person’s happiness state.
   - Adapt the `toString()` method to include the happiness attribute.
Comments on this lab, please:

What was the most valuable thing you learned in this lab?

What did you like best about this lab?

Was there any particular problem?

Do you have any suggestions for improving this lab as an effective learning experience?