Lab 8

Name:_________________________  Checked:_____

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

Preparation, Part A: Exercise using the Die class
For this exercise we will be modifying an example from the text involving the Die class (for 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.

4. Create a new Die method called nudge() that increments the die’s value (if the value is six, it should get circle back to one.

   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.

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

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:
   | 5   |  | *   * |
   +-----+  +--------+
Preparation, Part B: Exercise using the Account class
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.
Run the Transactions class (NOT the Account class) and observe what happens. 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:

___________________________________________________________


d) What happens when you try to run the Account class?

________________________________________

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 "Warren Buffet" 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.
Examine the getBalance() method in the Account class. Note that it returns the balance in the account. Add some more code in OnePercent.java to use the getBalance() method to get the balances of the four accounts and 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: You will need to use NumberFormat to ensure that the numbers are displayed as currency.
   
   Examine the code for the Account class. Which of the Account method(s) use NumberFormat?

6. Write some additional code in OnePercent.java to "tax" the accounts.
Using the getBalance(), withdraw(), and deposit() methods, withdraw 15% from each of the first three accounts and deposit it in the "Uncle Sam" account.
   
   Note: Be sure to calculate the 15% tax by multiplying the current balance by 0.15 (i.e., do not calculate it yourself, let the program do it!).

Add some code following this to print all of the account information again. Add a couple of extra statements to label the output "before taxes" and "after taxes"

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

7. Change the toString() method in the Account class.
Change the code in 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 has NOT changed).

SUBMIT files for preparation through blackboard under Lab 8 prep:

- Die.java
- RollingDice.java
- Account.java
- OnePercent.java
Part A: Modifying the Account class

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. In this exercise you will define alternative withdraw() and constructor methods. Then, you will also define a method addInterest() to compute and add interest to an account.

1) Add another 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 the taxes from the accounts.
   Reminder: The name of this method should still be withdraw(). See note above.

2) 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.

3) Create a new method that adds interest to the account.
The amount added should be computed according to the rate given by its parameter. For example, if the acct1 balance is $100.00 and the method is invoked as follows:

   acct1.addInterest(0.015);

the balance of acct1 should increase by 1.5% (thus $100 + $1.50 = $101.50 ). Test your method by invoking it four times to add interest to all the accounts (including Uncle Sam’s!).
Part B: Implementing your own class and a client to test it

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. Use the comments below as guidelines (copy and paste them into a new Java file and fill in the required Java code).

   Be sure to set up the class 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. (optional) You can also add more instance variables to the `Person` class.

For example, a happiness attribute:

```java
boolean happiness
```

Modify your constructor so that it sets happiness to true (i.e., makes it the default value for happiness ☺). Add methods `setHappiness()` and `getHappiness()` similar to `setName()` and `getName()`, above.
Lab 8 Comments

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?