Lab 7 Name:___________________________ Checked:_______

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

1. Download and compile `Account.java` and `Transactions.java`. Run the Transactions class (NOT the Account class).

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) 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. Create a new client for the `Account` class (similar to `Transactions.java`). This application should be named `OnePercent.java` and should do the following:

- create an account for someone named "Donald Trump" with $400 as initial balance and account number: 20230715
- create an account for someone named "Bill Gates" with $500 as initial balance and account number 31558040
- create an account for someone named "Warren Buffet" with $600 as initial balance and account number 44003050
- print the information for these three accounts

Test your code before proceeding

3. 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

4. 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.

5. Now 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 of each account by 0.15 (i.e., do not calculate it yourself, use getBalance() to obtain it and let the program do the calculation). When withdrawing the tax, use

- 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.

6. Next, we will make some changes to the `Account` class.

a) Change 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 this one since it should NOT be changed).

* Write out how Donald Trump's account info is displayed here:

____

b) 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 without also charging a fee.

> ***Note:*** Recall that Java allows you to define alternative versions of methods using the same method name as long as the different versions also have different a different number or types of parameters. In this exercise you will define alternative `withdraw()` and constructor methods.

c) Add another version of the constructor that takes only 2 parameters: name and account number (ie, 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.

d) Create a new method that adds interest to the account, 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% (so $100 + $1.50 = $101.50). Test your method by invoking it four times to add interest to all the accounts (including Uncle Sam's!).

Extra Exercise 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 `toString()` method.

c) Compile your class and fix any errors before proceeding.

```
//********************************************************************
//   Person.java       Author: YOUR NAME HERE
//      Represents a person, with attributes: name, age.
//********************************************************************

   // instance variables: name, age

   //-----------------------------------------------------------------
   //   Constructor: Sets up the person by defining the name, and age
   //-----------------------------------------------------------------

   //-----------------------------------------------------------------
   // toString():returns a String describing this person, eg:
   //        "Jasmine, 19"
   //-----------------------------------------------------------------
```

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 `main()` method appropriately).

```
//********************************************************************
//   PeopleBeingPeople.java       Author: YOUR NAME HERE
//
//      Driver class to test Person class.
//********************************************************************

   //   main(): creates some Person objects, prints their info.


      // Instantiate three objects of the Person class, assign them
      //    to variables named friend1, friend2, friend3.
      //    (Use names and ages of your choice.)


      // Print out info about friend1, friend2, friend3.
```

3. In the `Person` class, add another constructor that has only one parameter, the name. Modify your driver class 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.

```
//--------------------------------------------------------------
// 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 and to then 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).

Extra Exercise 2: Modify the `Die` class.

1. Download and compile `Die.java` and `RollingDice.java`. Run the RollingDice class (NOT the Die class).

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

- Modify RollingDice.java so that it creates a third die and rolls it also

2. 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 |
+---+
```

- Run RollingDice.java to observe the effect of this change.

3. 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. Test your method by adding some code in RollingDice.java to "nudge" up the values of the three dice and print them again.