CMPS 1600 Introduction to Computer Science II – Spring 2018
Lab 2

Objectives

- To practice implementing inheritance in Java.
- To learn about the `Object` superclass and the methods it contains.

Code quality

The code you submit should be easily readable, clean and well-commented.
Commented-out sections of non-working code should be removed. Indentations and
white spaces should be used to separate
logical blocks of the code. Variables, methods and classes should have descriptive names.
Every class and method should have a comment according to Javadoc format (see Zybook).
When overriding methods, use `@Override` compiler annotation.

Square and Object

1. Download your code from last week's `Rectangle` class and use it in this exercise. Refer to Zybook for examples to how to implement inheritance in Java.
2. Add `toString()` method of the `Rectangle` class so that it displays the information about the Rectangle object in a user-friendly way (i.e. it informs the user about the fact that it is a rectangle, and about its dimensions).
3. Make another class, `Square`, that is a subclass of `Rectangle`. The `Square` constructor should only take *one* size argument. Think about what can be directly inherited from the `Rectangle` class and what needs to be overwritten.
4. Override the `toString` method of the `Rectangle` class so that it displays the information about the Square object in a user-friendly way and informs the user about the fact that the object is a square, and about its dimensions.
5. Make a separate class, `SquareTester` to test the functionality of the above class. Create a square, print out the information about it using `toString` method, calculate and display its area and perimeter.
6. Look online at `http://docs.oracle.com/javase/8/docs/api/java/lang/Object.html` to read about Java's `Object` class. Every class in Java is a subclass of `Object`. Browse the various methods that the `Object` class defines. Do not worry about understanding what every method does.
7. In the `Square` class, override the `equals` method. The default `equals` method only returns `true` if the two objects are the same. For the `Square` class, two squares should be equal if they have the same size. The `equals` method takes an

`Object` as its argument. If your `Object` parameter is `Object obj`, you can cast it to a `Square` by using code such as: `Square s = (Square) obj;`. Before doing this, you should make sure that your `Object` is indeed a `Square`. To do this, you can use the `instanceof` method. Finally, you can now test the equality of the sizes of the squares.

8. Now in the `SquareTester` class from above, make two rectangles of the same size and two squares of the same size. Test the rectangles for equality using `==` and `equals`. Now do the same for the squares. Print out the results.

Bank Account

1. Create a class `BankAccount`. An object of this class should have a person's `name` and a `balance`, stored as a `double`. Create methods `getName` and `getBalance`, and create `deposit` and `withdraw` methods to add and subtract a specified amount from the `balance`. Furthermore, create a `transfer` method, that takes another `BankAccount` and a specified amount, and transfers that amount from the original `BankAccount` to the one that is given. Finally, create a `toString` method that gives the name and balance of the account.

2. Now create another class named `SavingsAccount`. This should be a subclass of `BankAccount`. A `SavingsAccount` should have an interest rate that is specified in its constructor. Furthermore, there should be a method, `addInterest`, that deposits interest into account, based on the account balance and interest rate.

3. Create another subclass of `BankAccount` named `CheckingAccount`. A `CheckingAccount` should keep track of the number of transactions made (deposits and withdrawals). Name this field `transactionCount`. Also, the set fee for transactions is three dollars, so create a `static final` field, `TRANSACTION_FEE`, that is set to 3.0. Finally, create a method, `deductFees`, that deducts fees from the account balance based on the number of transaction. This method should reset the `transactionCount` to zero.

4. Now make a class `BankAccountTest` with a `main` method to test the above classes. Create a `SavingsAccount` with zero balance, a 1% interest rate, and a name of your choice. Also create a `CheckingAccount` with a $500 initial balance and a name of your choice. Deposit $1000 into the `SavingsAccount`. Withdraw $100 from the `CheckingAccount`. Now transfer $200 from the `SavingsAccount` to the `CheckingAccount`. Print out both accounts. Now add interest to the `SavingsAccount` and deduct fees from the `CheckingAccount`. Print out both accounts and make sure the balances are as expected.