CS 1302A – HW 3 (Spring 2016)
Total: 100 pts
Due: Wednesday, Feb. 10, 2016 11:59 pm

This homework has 4 problems which deal with inheritance (Ch 11)

Eclipse Reminder

- You will create a Java Project in Eclipse with the name: *hw3_FLastname*
- You will have three packages: *prob1*, *prob2*, *prob3*. The Problem 4 code will go in the *prob1* package.

Problem 1

Do problem 11.3 from the text (p.445). Notes:

1. You will need to copy your Account class from HW 2, Problem 1.
2. Name the two subclasses: CheckingAccount and SavingsAccount.
3. The CheckingAccount class should have at least 3 constructors to be consistent with the superclass. However, you will only provide one. This one constructor should accept: id, balance, and overdraft limit. It should utilize a superclass constructor. Similar for SavingsAccount, supply one constructor that accepts id and balance and utilizes a superclass constructor.
4. Make sure the checking account subclass has methods to get and set the overdraft limit.
5. The problem says, "a checking account has an overdraft limit." This is the way this should be handled:

    a. Override the *withdraw* method. Only allow a withdrawal if the balance does not go below the negative of the overdraft limit. Example:
        Balance = $1000, OverdraftLimit = $5000, Cannot withdraw $7000 (Balance would be -6000 which is less than -5000). Can withdraw $5900 (Balance would be -4900 which is greater than -5000)
    b. Technically, I guess you should also override the *setBalance* method to work in the same way to be consistent, **but you do not need to do that**. I don't think the class should even have this method. The balance should be read-only, and modified only through the methods *withdraw* and *deposit*.

6. The problem says, "a savings account cannot be overdrawn." This simply means that the balance can never be negative. Similar to the checking account except that we don't need the overdraft limit.
7. Write a test class AccountTester in package *prob1,* which is exactly as shown below (copy/paste). Your code should work perfectly with this test class:

```
package prob1;

public class Prob1_Tester {

	public static void main (String[] args) {
		CheckingAccount account = new CheckingAccount(1122, 1000.0, 5000.0);
		printCheckingInfo(account);
		account.withdraw(2000.0);
		printCheckingInfo(account);
		account.withdraw(5000.0);
		printCheckingInfo(account);
		account.setOverdraftLimit(2000.0);
```

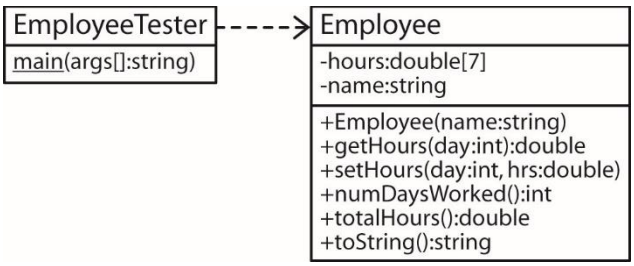
```
            printCheckingInfo(account);
            account.withdraw(1001.0);
            printCheckingInfo(account);
            account.withdraw(999.0);
            printCheckingInfo(account);

            SavingsAccount sAccount = new SavingsAccount(1122, 1000.0);
            printSavingsInfo(sAccount);
            sAccount.withdraw(1001.0);
            printSavingsInfo(sAccount);
            sAccount.withdraw(999.0);
            printSavingsInfo(sAccount);
      }

      public static void printCheckingInfo( CheckingAccount acnt) {
            System.out.printf("Bal: $%4.2f, overdraft limit: $%4.2f\n", acnt.getBalance(),
acnt.getOverdraftLimit());
      }

      public static void printSavingsInfo( SavingsAccount acnt) {
            System.out.printf("Bal: $%4.2f\n", acnt.getBalance());
      }
   }
```

Your code should print out:

```
Bal: $-1000.00, overdraft limit: $5000.00
Bal: $-1000.00, overdraft limit: $5000.00
Bal: $-1000.00, overdraft limit: $2000.00
Bal: $-1000.00, overdraft limit: $2000.00
Bal: $-1999.00, overdraft limit: $2000.00
Bal: $1000.00
Bal: $1000.00
Bal: $1.00
```

Problem 2

The class diagram for HW 2, Problem 2 is shown below. :



1. Copy the *Employee* and *EmployeeTester* classes from HW 2, Problem 2 into your project in the *Prob2* folder/package.
2. Add a *wages* method to the *Employee* class. This method accepts a pay rate ($/hr) that the employee earns. The method computes the pay by multiplying the total hours by the pay rate (e.g. no overtime). Hint: all you have to do is multiply the result of *totalHours()* by the pay rate. Example (assume pay rate of $10/hr)

3. Add a subclass, *HourlyEmployee.* Override the *wages* method so that it computes and returns the pay using the pay rate for the first 40 hours and time-and-a-half for any hours over 40.0. Four examples (assume pay rate of $10/hr):

hours						
M	Tu	W	Th	F	Sa	Su
10	10	10	10	10	6	0

Reg Wages	OT Wages	Total Wages
$400.00	$240.00	$640.00

hours						
M	Tu	W	Th	F	Sa	Su
10	0	10	10	0	4	0

Reg Wages	OT Wages	Total Wages
$340.00	$0.00	$340.00

hours						
M	Tu	W	Th	F	Sa	Su
10	10	10	10	10	0	0

Reg Wages	OT Wages	Total Wages
$400.00	$150.00	$550.00

hours						
M	Tu	W	Th	F	Sa	Su
8	8	8	8	8	8	0

Reg Wages	OT Wages	Total Wages
$400.00	$120.00	$520.00

4. Add a subclass, *SalariedEmployee.* Override the *wages* method so that the employee gets the normal pay rate during the week (Monday-Friday), no matter how many hours they work. Any hours worked on Saturday or Sunday always get time-and-a-half. Four examples (assume pay rate of $10/hr):

hours						
M	Tu	W	Th	F	Sa	Su
10	10	10	10	10	6	0

Reg Wages	OT Wages	Total Wages
$500.00	$90.00	$590.00

hours						
M	Tu	W	Th	F	Sa	Su
10	0	10	10	0	4	0

Reg Wages	OT Wages	Total Wages
$300.00	$60.00	$360.00

hours						
M	Tu	W	Th	F	Sa	Su
10	10	10	10	10	0	0

Reg Wages	OT Wages	Total Wages
$500.00	$0.00	$500.00

hours						
M	Tu	W	Th	F	Sa	Su
8	8	8	8	8	8	0

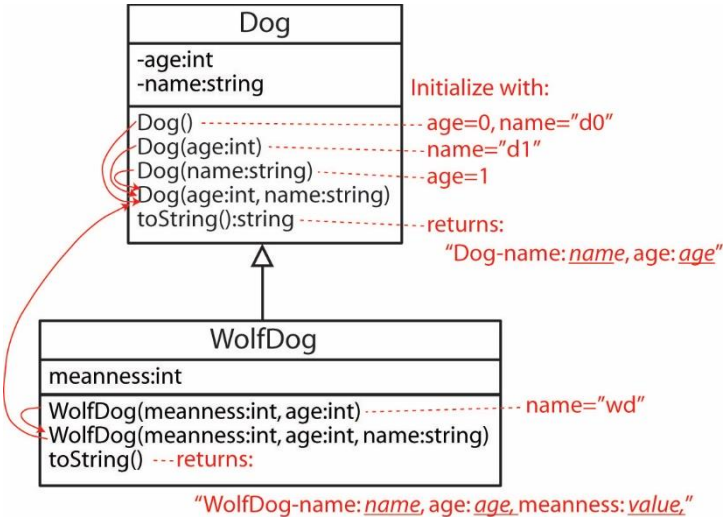
Reg Wages	OT Wages	Total Wages
$400.00	$120.00	$520.00

5. Modify *main* in the *EmployeeTester* class so that it produces a dialog and results like this:

```
-->Enter a name : Chen
-->Enter an Employee type (e,h, or s) : s
-->Enter hours worked (7 values separated by spaces): 10 10 10 10 10 6 0
-->Enter pay rate ($/hr): 10.0

Output:
Chen worked for 6 day(s) for a total of 56.0 hours.
Total Wages = $590.00
```

Problem 3

This problem focuses on constructor chaining, *e.g.* using *this* and *super*. Consider the class diagram and notes below.



Notes (also indicated by the arrows):

a. The first three Dog constructors call the fourth one.
b. The first WolfDog constructor calls the second one.
c. The second WolfDog constructor calls the 2-arg Dog constructor.

1. Write the two classes exactly as shown in the class diagram.
2. Create another *DogTester* class in package *prob3* and copy this code into it:

```
package prob3;

public class DogTester {

	public static void main(String[] args) {
		Dog d1 = new Dog();
		Dog d2 = new Dog("Lucky");
		Dog d3 = new Dog(7, "GG");
		Dog d4 = new Dog(12);

		WolfDog w1 = new WolfDog(9001, 5, "Juno");
		WolfDog w2 = new WolfDog(10, 1);
		WolfDog w3 = new WolfDog(100, 5, "Venti");

		System.out.println(d1 + "\n" + d2 + "\n" + d3 + "\n" + d4 + "\n" +
				w1 + "\n" + w2 + "\n" + w3);
	}
}
```

Your code should print out the following:

```
Dog-name: d0, age: 0
Dog-name: Lucky, age: 1
Dog-name: GG, age: 7
Dog-name: d1, age: 12
WolfDog-name: Juno, age: 5, meanness: 9001
WolfDog-name: wd, age: 1, meanness: 10
WolfDog-name: Venti, age: 5, meanness: 100
```

Problem 4

1. This problem continues Problem 1. Create a class, *Prob4_Tester* **in the *prob1* package (you will not create a *prob4* package).**

2. Write a static method, *printOverdraftLimit(Account a)* that prints the overdraft limit for the account if it is a checking account and prints "not applicable" otherwise.

3. Use this *main* to test:

```
public static void main(String[] args) {
	CheckingAccount ca = new CheckingAccount(3344, 1000.0, 2000.0);
	ca.setOverdraftLimit(500.0);
	printOverdraftLimit( ca );

	Account a = new SavingsAccount(3344, 1000.0);
	printOverdraftLimit( a );
}
```

The code should print out:

```
Overdraft Limit: $500.00
Not applicable
```

Submission

Follow the directions in Lab 1 to zip the folder *hw3_FLastname*.

Make sure you zip the ENTIRE folder *hw3_FLastname*!! Submit your zip file to Blazeview by the due date. The name of your file should be: *hw3_FLastname.zip*.

Grading

I will RANDOMLY select one problem for grading. For example, if problem 3 is selected, I will only go to the package/folder *prob3* for grading. So make sure all your Java codes are in the correct packages/folders.

Additional Requirements:

1) No late submission will be accepted.
2) Please exactly follow the naming rules described above. You will be deducted 10 points for incorrect naming.
3) Write comments at the beginning of your Java source file(s) to indicate your name, student ID, "CS 1302-A Homework 3", and the due date.
4) Make sure that your programs are well-documented, readable, and user friendly. Make sure you document your programs by adding comments to your statements/blocks so that I can understand your code. You will

get 0 to 10 points based on the helpfulness of your comments. You will be deducted 10 points for no comments.

5) It is your responsibility to make sure your programs can compile and run correctly. Please be aware that programs that do not compile may receive ZERO credit.
6) When grades are returned to you via BlazeView, you have 7 days to meet with the instructor to discuss on grade changes. After 7 days, the grades are written in stone and can't be changed after that point.