Classes and Constructors

Rules:

- You ARE allowed to use char, String and any array
- Utilize “loops” and “if” conditions as necessary
- Use Scanner class when taking input

Task 1

Design a “Vehicle” class. A vehicle assumes that the whole world is a 2 dimensional graph paper. It maintains its x and y coordinates (both are integers). The vehicle gets manufactured (constructed?) at (0,0) coordinate.

Write an user class called “Vehicle”. It must have methods to move up, down, left, right and a toString method for printing current coordinate.

Note: All moves are 1 step. That means a single call to any move method changes value of either x or y or both by 1.

Take help from:
http://www.javabeginner.com/learn-java/java-tostring-method
http://cscie160-distance.com/toString.html

```
public class VehicleUser{
    public static void main(String[] args){
        Vehicle car = new Vehicle();
        System.out.println(car.toString());
        car.moveUp();
        System.out.println(car.toString());
        car.moveLeft();
        System.out.println(car.toString());
        car.moveDown();
        System.out.println(car.toString());
        car.moveRight();

// see, output for following two lines are same because toString() is
automatically called. So, you can omit toString when printing.
        System.out.println(car.toString());    }
}
```

Expected Output:

```
(0, 0)
(0, 1)
(-1, 1)
(-1, 0)
(0, 0)
```

Task 2

Design a Vehicle2010 class which inherits movement methods from Task1 and adds new methods called move UpperRight, UpperLeft, LowerRight, LowerLeft. Each of these diagonal move methods must re-use two inherited and appropriate move methods. Write user class as well which will show that all of your methods are working. A small user class is shown below as an example.

Note: All moves are 1 step. That means a single call to any move method changes value of either x or y or both by 1.

```
public class Vehicle2010User{
    public static void main(String[] args){
        Vehicle2010 car = new Vehicle2010();
        System.out.println(car.position());
        car.moveLowerLeft();
        System.out.println(car.position());

        Vehicle2010 car2 = new Vehicle2010();
        car2.moveLeft();
        System.out.println(car.position());
        car2.moveDown();
        System.out.println(car.position());
    }
}
```

Expected Output:

```
(0, 0)
(-1, -1)
(-1,0)
(-1,-1)
```

Task 3

Create a class called `Student` as described below:

- **Fields:**
  `name, id, address, cgpa`
- **Methods:**

```
public String getName()
public void setName(String n)
public String getID()
public void setID(String i)
public String getAddress()
public void setAddress(String a)
public double getCGPA()
public void setCGPA(double c)
```

Write a class called `Main` to write a `main()` method:

-
```
public static void main(String[] args){

}
```

- Inside the `main()` method
  - Create 3 objects/instances of `Student` called `john`, `mike` and `carol`
  - Set their fields to some value using the public methods.
  - Print the information of each `Student` using `System.out.println()`

Task 4

Create a class called `BankAccount` as described below:

- **Fields:**
  `name, address, accountID, balance`
- **Methods:**

```
public String getName()
public void setName(String n)
public String getAccountID()
public void setAccountID(String i)
public String getAddress()
public void setAddress(String a)
public double getBalance()
public void setBalance(double c)
public void addInterest() //adds 7% of the balance
```

1. Write a class called `Main` to write a `main()` method:
   -
```
public static void main(String[] args){

}
```
   - Inside the `main()` method
     - Create 3 objects/instances of `BankAccount` called `acc1`, `acc2` and `acc3`
     - Set their fields to some value using the public methods.
     - Call `addInterest()` on `acc1` and `acc3`
     - Print the information of each `BankAccount` using `System.out.println()`

2. Add constructors to `Student` and `BankAccount` and use the constructor to set the field values.

Task 5

Create a class **SavingsAccount,** which will use a **static** class variable to store the **annualInterestRate** for all account holders.

- Each object of the class contains a **private** instance variable **savingsBalance** indicating the amount the saver currently has on deposit.
- Provide method **calculateMonthlyInterest()** to calculate the monthly interest [by multiplying the **savingsBalance** by **annualInterestRate** divided by 12], this interest should be added to **savingsBalance**.
- Provide a **static** method **modifyInterestRate()** that sets the **annualInterestRate** to a new value.

Write a driver program to test class **SavingsAccount**.

- Instantiate two **SavingsAccount** objects, **saver1** and **saver2**, with balances $20000.00 and $30000.00, respectively using constructor.

- Set **annualInterestRate** to 4.2%, then calculate the monthly interest and print the new balances for each of the savers using **printSavingsBalance( )** method.

Then set the annualInterestRate **to 5.5% and calculate the next month's interest and print the new balances for each of the savers.**

Task 6

Assume that a bank maintains two kinds of accounts for its customers, one called savings account and the other current account. The savings account provides compound interest (5%) and withdrawal facilities. The current account provides no interest. Current account holders should also maintain a minimum balance and if the balance falls below this level, a service charge in imposed.

Create a class **Account** that stores customer name, account number, type of account and balance. From this **Account** class derive the classes **CurrentAccount** and **SavingsAccount** to make them more specific to their requirements. Include the necessary methods in order to achieve the following tasks:

- Initialize all instance variables through constructors. [use super()]
- Accept deposit from a customer & update the balance [**depAmount** (…)]
- Display the balance [**showBalance** ( )]
- Compute and deposit interest [**computeInterest** ( )]
- Permit withdrawal and update the balance [**withdraw** (…)]
- Check for the minimum balance (assume $500), impose penalty (print a message, if necessary) and restrict the withdrawal of balance.

[Use only methods to initialize the class members and other tasks.]