ECE25100 Object Oriented Programming

Lab 4: Constructors, Get/Set Methods and Behaviors

Description:
The purpose of this lab is to help you understand more about some of the standard "things" that we do when we create objects. We will discuss how to initialize our objects as well as how to protect their state through get/set methods.

To get credit for the lab, you need to demonstrate to the instructor/lab assistant that you have completed all the requirements, and sign your name on the grade sheet.

STEP 1. Recall from the previous lab that we created and defined a **`Customer`** object with state and behaviors as follows:

```
public class Customer {
    String     name;
    int        age;
    char       sex;
    float      money;
    boolean    admitted;

    public boolean spend(float amount) {
        if (money < amount)
            return false;
        money -= amount;
        return true;
    }
    public void give(Customer c, float amount) {
        if (money >= amount) {
            money -= amount;
            c.money += amount;
        }
    }
    public float computeFee() {
        if (age <= 3)
            return 0;
        if (age < 18)
            return 8.50f;
        if (age >= 65)
            return 12.75f * 0.50f;

        return 12.75f;
    }
    public void payAdmission() {
        if (spend(computeFee()))
            admitted = true;
    }
}
```

Copy the above code into a new JAVA file called **`Customer.java`** and compile it. Also copy the code below into a **`CustomerTestProgram.java`** file and compile/run it to test the **`Customer`** object:

```
public class CustomerTestProgram {
    public static void main(String args[]) {
        Customer c1 = new Customer();
        c1.name = "Bob";
        c1.age = 17;
        c1.sex = 'M';
        c1.money = 10;
        c1.admitted = false;

        Customer c2 = new Customer();
        c2.name = "Dottie";
        c2.age = 3;
        c2.sex = 'F';
        c2.money = 0;
        c2.admitted = false;

        Customer c3 = new Customer();

        System.out.println("Bob looks like this:        " + c1);
        System.out.println("Dottie looks like this:     " + c2);
        System.out.println("Customer 3 looks like this: " + c3);
    }
}
```

Notice that we explicitly specified the **name**, **age**, **sex**, **money** and **admitted** for both customers. This is called *initializing* the object and it is usually done at the point that we create the object. Unfortunately, it took 5 lines of JAVA code to initialize each customer. This is a little annoying and tedious, but recall that if we did not set the values, they would be set to zeros.

A better way to initialize an object with less code is to create our own constructor. A constructor is a special method that takes 0 or more parameters and it is called automatically when we create a new object. Let's make a constructor that takes a single **name** parameter as follows:

```
public Customer(String name) {
}
```

Notice that it MUST be called **`Customer`** and that it does not have a return type, not even **void**. Write the above constructor in the **Customer** class then compile it. Now go to the **`CustomerTestProgram`** and re-compile it. You should get 3 errors saying "cannot find symbol constructor Customer()". What happened ? Well, because you created your own constructor, JAVA decided that you were not longer allowed to use the default constructor (i.e., the one that it provided to you before you wrote your own). Notice that your test code calls **new Customer()** which takes no parameters. However, the constructor that you just made requires a **String** parameter, so JAVA does not recognize it.

You can call your new constructor by supplying a String, which should be the name of the Customer. So, change the line `Customer c1 = new Customer();` to `Customer c1 = new Customer("Bob");` and re-compile. You will now see that only 2 errors remain, because JAVA now understands that you are calling your newly created constructor in the first line. Fix the remaining two errors by supplying a Strings "Dottie" and "Jane" to the constructor calls. The code should now compile.

Of course though, your constructor does not really do anything yet. Assume that we moved the line c1.name = "Bob"; into the constructor. Can you explain why the code below would not compile ?

```
public Customer(String name) {
    c1.name = "Bob";
}
```

Right! **c1** is not defined here in the **`Customer`** class because **c1** is just a variable we are using in our test program. If we then change the code as shown below the code will compile:

```
public Customer(String name) {
    name = "Bob";
}
```

Now the code assigns the name variable the value of "Bob". Assume then that we adjusted our test code as follows:

```
public class CustomerTestProgram {
    public static void main(String args[]) {
        Customer c1 = new Customer("Bob");
        c1.age = 17;
        c1.sex = 'M';
        c1.money = 10;
        c1.admitted = false;

        Customer c2 = new Customer("Dottie");
        c2.age = 3;
        c2.sex = 'F';
        c2.money = 0;
        c2.admitted = false;

        Customer c3 = new Customer("Jane");

        System.out.println("Bob looks like this:        " + c1.name);
        System.out.println("Dottie looks like this:     " + c2.name);
        System.out.println("Customer 3 looks like this: " + c3.name);
    }
}
```

Notice that we removed the code that set the customer's name and also displayed the name at the end. Compile and run this changed code. You should see that all of the names are displayed as

null. Why ? Notice that the constructor has a *parameter* called **name** and that the **`Customer`** class also has an *attribute* called **name**:

```
public class Customer {
    String     name;

    ...
    public Customer(String name) {
        name = "Bob";
    }
    ...
}
```

When JAVA sees that you are trying to set the **name** variable to "Bob", it actually thinks that you are talking about the parameter, not the customer's attribute. To avoid this problem, you should always choose parameter names (for all constructors and methods) that are different from your attribute (i.e., instance variable) names. This is better code:

```
public Customer(String aName) {
    name = "Bob";
}
```

However, if you ran the above code, everybody would be named **Bob**!! Instead, change your code to use **aName** instead of "Bob":

```
public Customer(String aName) {
    name = aName;
}
```

Now run your code and make sure that the customer names are correct. Notice that the test code is a little simpler now.

STEP 2. In a similar way, we can set the initial values for all of the other attributes as well. You can create as many constructors as you want, as long as their list of parameters is different. It is always a good idea to set ALL of the attribute values in each of your constructors. If the attribute value is not supplied as a parameter, set the attribute to some default value. For example, we can properly complete our constructor as follows:

```
public Customer(String aName) {
    name = aName;
    age = 0;
    sex = 'M';
    money = 0.0f;
    admitted = false;
}
```

Create three **more** constructors with parameters as follows:

- One that takes the **name**, **age** and **sex** of the Customer

- One that takes the initial **name**, **age**, **sex**, **money** and **admitted** values of the Customer
- One that takes *no parameters* (i.e., uses default values that are reasonable)

Change your test code to look as follows (copy/paste it):

```
public class CustomerTestProgram {
    public static void main(String args[]) {
        Customer c1 = new Customer("Bob", 17, 'M');
        Customer c2 = new Customer("Dottie", 3, 'F', 10, true);
        Customer c3 = new Customer("Jane");
        Customer c4 = new Customer();

        System.out.println("Bob looks like this:        " + c1);
        System.out.println("Dottie looks like this:     " + c2);
        System.out.println("Jane looks like this:       " + c3);
        System.out.println("Customer 4 looks like this: " + c4);
    }
}
```

The code should compile.   Notice how "clean" it looks when compared to our initial testing code that set each value explicitly.

STEP 3. If you run the code above, you will see that the customers do not display nicely.   That is because we did not write a method called **toString**.   The **toString** method is the method that JAVA calls in order to determine a **String** representation of the object.   You will now write your own **toString** method in the **`Customer`** class that will return a better representation of the customer.   All **toString** methods must have the following format:

```
public String toString() {
    ...
}
```

Notice that it returns a **String** object and takes no parameters.   Fill in the **toString** method using a single line:

return "Customer named " + name;

Re-compile your **`Customer`** class and then re-run your test code. Do you understand the output ?   Modify the **toString** method so that it displays all **Customer** objects in a format like this:

```
Customer Bob: a 17 year old male with $0.0 who has not been admitted
```

or

```
Customer Dottie: a 3 year old female with $10.00 who has been admitted
```

Hint: You will have to build up a **String** by joining together fixed strings with attribute values (i.e., use the + operator in JAVA). You will also need to use an **IF** statement to decide whether to write **male** or **female** as well as another IF statement to decide whether or not to write "**not**" before "**been admitted**". You may want to start off with a simple String variable and continue appending to it:

```
public String toString() {
    String result = "Customer ";
    ...
    return result;
}
```

You should run your test code and examine the output for correctness. You should be able to explain why it indicates **Jane** as being a **male**.

STEP 4. Recall from our discussion in class that it is not always a good idea to allow an object's attributes (i.e., instance variables) to be manipulated directly in our code:

```
Customer c1 = new Customer();
c1.money = 5000000;
```

To prevent this, it is common practice (that means that you should do this) to declare our instance variables **private**. Do to your **Customer** class and change all of the instance variables to have **private** access:

```
public class Customer {
    private String     name;
    private int        age;
    private char       sex;
    private float      money;
    private boolean    admitted;
    ...
}
```

Now re-compile the **Customer** class. Our test code (from part 2 of this lab) should still run. Why ? Are we not still manipulating the Customer objects from outside the Customer class (i.e., from our test code) ? Yes. We are changing the objects. However, **private** access must means that we cannot access the private parts (i.e., attributes) of the object ***directly***. Notice that all of the code that accesses/modifies the object attributes is in the **Customer** class's methods (i.e., **spend**, **give**, **computeFee**, **payAdmission**, **toString**). These are the only places that we are accessing and modifying the attributes of the object.

Copy/paste the following code as your **CustomerTestProgram** code and the compile it:

```
public class CustomerTestProgram {
    public static void main(String args[]) {
        Customer c = new Customer();
        c.name = "Steve";
        c.money = 20;
        c.admitted = true;
        System.out.println(c.name);
        System.out.println(c.money);
        System.out.println(c.admitted);
    }
}
```

You should see six errors indicating that each variable "has private access in Customer". So we can no longer write code like this that attempts to directly access the object's parts. But, it is often the case when programming that we need to ask an object for some information about it (e.g., given a **Customer** object, we want to know his/her name) or we want to change something (e.g., age, or money amount).

In these cases, we usually create get/set methods for the object. A ***get*** method is a public method that "gets" the value of an object's attributes. Here is the general format:

```
public <theAttribute'sType> get<theAttribute'sName>() { return
<theAttribute'sName>; }
```

Create a **get** method in the **Customer** class for each of the 5 attributes (i.e., instance variables). Compile your code.

A ***set*** method is a **public** method that "sets" the value of an object's attribute to whatever value you supply as a parameter. Here is the general format:

```
public void
set<theAttribute'sName>(<theAttribute'sType>  <aParameterName>)
{  <theAttribute'sName> = <aParameterName>; }
```

Create a **set** method in the **Customer** class for each of the 5 attributes (i.e., instance variables). Compile your code. Go back to your test method and make the appropriate changes so that it uses your new set methods and your new get methods.

Just as a side note, get methods that return booleans are usually start with **is** or **hasBeen** instead of **get**. For example, you should change the name of **getAdmitted()** to either **isAdmitted()** or **hasBeenAdmitted()**. Sometimes, **boolean** attributes have multiple set methods to set specific values. For example, instead of doing **c.setAdmitted(true)** or **c.setAdmitted(false)**, you can make separate set methods like these:

```
public void admit() { admitted = true; }
public void deny() {admitted = false; }
public void toggleAdmittance() { admitted = !admitted; }
```

Then you can say **c.admit()** or **c.deny()** or **c.toggleAdmittance()**. Do you understand all of these and when you might want to use them ?

Lastly, there are some situations in which we may not even want any **set** methods (i.e., never allow anyone outside to change the attributes). Also, in cases where there are attributes that nobody needs to know about (e.g., parts of our attributes that are specific to the way we wrote our code and no outside class would even need to access them), then we may not even create a **get** method for them.

STEP 5. Backing up to question 2 on this lab, you will notice that your constructors probably have duplicate information. Here are my constructors, although yours may have different default values:

```
public Customer() {
    name = "";
    age = 0;
    sex = 'M';
    money = 0.0f;
    admitted = false;
}

public Customer(String aName) {
    name = aName;
    age = 0;
    sex = 'M';
    money = 0.0f;
    admitted = false;
}

public Customer(String aName, int anAge, char aSex) {
    name = aName;
    age = anAge;
    sex = aSex;
    money = 0.0f;
    admitted = false;
}

public Customer(String aName, int anAge, char aSex, float aMoney,
boolean anAdmitted) {
    name = aName;
    age = anAge;
    sex = aSex;
    money = aMoney;
    admitted = anAdmitted;
}
```

Notice that there is a LOT of duplication in regards to the code that sets the default values. The most flexible constructor is the last one which takes 5 parameters. Adjust the code in the other 3 constructors by reducing them to one line of code. Have them call the 5-parameter constructor by using the appropriate default values. Recall that to call one constructor from another one, we use the keyword **this**, followed by the parameter list within round brackets.