ECE25100 Object Oriented Programming

Lab 3: Defining Objects, Making State and Behaviors

Description:

The purpose of this lab is to help you understand how to create objects and how to use them in a simple way. You will make an object, give it some state and then create some very simple behaviors for it.

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. Copy and paste the following code into a file called **`CustomerTestProgram.java`**.

```
public class CustomerTestProgram {
    public static void main(String args[]) {
        Customer c;

        System.out.println(c);
    }
}
```

Compile the code and notice that there is an error. JAVA has no idea what a **Customer** object actually is and you are trying to use such an object. Now make a new JAVA class called **`Customer`** and save it into a file called **`Customer.java`** in the same directory as you made the **`CustomerTestProgram.java`**. The code for the **`Customer`** class should simply be as follows:

```
public class Customer {
}
```

Compile the **`Customer`** class.

Now let's see if you can use it. Go back and compile the **`CustomerTestProgram`**. Now JAVA will understand that you want to use the **`Customer`** object that you just defined (it knows this because a **`Customer`** class has been compiled in the same directory). However, the code will still not compile because it generates the error: "variable c might not have been initialized" This is JAVA's "annoying" way of telling you that you did not give a value to variable **c**. Change **Customer c;** to **Customer c =** null**;** and then recompile.

It will compile OK. If you run the code, you will see **null** displayed. **null** is JAVA's way of representing an undefined object. You see, even though we created a new **`Customer`** variable **c**, we did not make a new **`Customer`** object. We only made the variable to hold one.

Change **System.out.println(c);** into **System.out.println(c.toString());** and then recompile and run the code. JAVA will generate an error like this: "Exception in thread "main" java.lang.NullPointerException". This is JAVA's way of telling you that you are trying to use an object, but it has not been defined. We are trying to send the **toString()** message to **c**, but **c** is an empty variable because no **`Customer`** object exists yet ! You will see many NullPointerExceptions this term while you are writing your programs. Remember, it usually just means that you forgot to create your object somewhere.

Change **Customer c =** null**;** to **Customer c =** new **Customer();** and then recompile and run your code. You should now see something like this displayed **Customer@82ba41** (although the exact numbers and letters will vary each time). Congratulations! Now you have just created your first very own object.

Your **`Customer`** object is very plain. You cannot do much with it at the moment. We can only make a bunch of **`Customer`** objects and display them. Try the following code out:

```
public class CustomerTestProgram {
    public static void main(String args[]) {
        Customer c1 = new Customer();
        Customer c2 = new Customer();
        Customer c3 = new Customer();

        System.out.println(c1);
        System.out.println(c2);
        System.out.println(c3);
    }
}
```

Three **`Customer`** objects are created, but all look different when displayed because the weird numbers/letters represent the location of the object in the computer's memory and since each customer is a unique one, it is in a different location. Notice as well that you can leave off the call to the **`toString()`** method on the **`Customer`** objects when displaying them because **`System.out.println()`** will automatically call the **`toString()`** method whenever you try to display an object.

STEP 2. Assume that you want to specify the name of the **`Customer`**. Change your code to look as follows:

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

The dot **.** after the variable **c** means that we are trying to accessing a piece (or attribute) of the object ... in this case the customer's **name**. The code generates the compile error: "cannot find symbol variable name". This is because you have not told JAVA that your **`Customer`** objects are supposed to have names. You need to go back to your **`Customer`** class definition and add an attribute called **name**. Open your **`Customer`** class (i.e., click on the tab in JCreator if it is already opened). Add an instance variable (or attribute) called name which is of type **String** as follows:

```
public class Customer {
    String   name;
}
```

Recompile the **`Customer`** class and then go back and recompile and run the **`CustomerTestProgram`** class. It should work now, displaying "Bob". Add three more instance variables to keep track of the Customer's **age** (an **int**), **sex** (a **char** 'M' or 'F') and **money** (a **float**). Now test the following code:

```
public class CustomerTestProgram {
    public static void main(String args[]) {
        Customer c = new Customer();
        c.name = "Bob";
        System.out.println(c.name);
        System.out.println(c.age);
        System.out.println(c.money);
        System.out.println(c.sex);
    }
}
```

The code should compile and run. Notice the values of the **age**, **sex** and **money** variables. In JAVA, all unassigned variables are set to 0. Character 0 is the **null** character and it does not display on the screen. Try assigning values to the variables so that Bob is a 27 year old male with $50. Then re-run to see the results.

STEP 3. Assume now that you want your **`Customers`** to be able to spend some of their money. Perhaps we would want to write code like this:

```
public class CustomerTestProgram {
    public static void main(String args[]) {
        Customer c = new Customer();
        c.money = 50;

        System.out.println(c.money);
        c.spend(10);
        c.spend(4.75f);
        c.spend(15.25f);
        System.out.println(c.money);
    }
}
```

The code above simulates the situation in which our customer spends money 3 times. Before spending, he has $50 and afterwards, he has $20 remaining. As a result, the state of the **`Customer`** object will have been changed. Of course, the code does not compile. It generates 3 compile errors because we are trying to tell JAVA to send the "spend" message to the customer

c. However, JAVA goes to the **Customer** class and notices that we have not yet defined "how" to perform the spend operation on the customer. That is, there is no **spend()** method written yet, so it complains. We need to write such a method.

In the **Customer** class, create a **public** method called **spend** that takes a single parameter of type **float** called **amount**. We use a parameter here because we want to be able to specify the "amount" of money that the **Customer** is spending. If you will notice in the above code, the spend method is like a command, and we don't expect back any particular value from the method call, we just want the method to change the **Customer** object. So, the return type for the method is **void**:

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

    public void spend(float amount) {

    }
}
```

Now our earlier code will compile and run, but the customer's money will not decrease. That is because we have not "what" JAVA is supposed to do when **spend** is called. Add the proper line of code to the spend method so that it works properly. Recompile the **Customer** class, then run the **CustomerTestProgram** to verify if your code is correct.

Try adding **c.spend(100);** to your test code just above the last print statement. You will likely see **-80.0** as a result. The customer has over-spent. This is probably a bad thing. Go back and modify your **spend** method so that customers can never overspend. It should simply ignore any spend requests over the amount that the **Customer** actually has.

STEP 4. Here is a more interesting situation that uses two **Customer** objects where one **Customer** gives money to the other:

```
public class CustomerTestProgram {
    public static void main(String args[]) {
        Customer c1 = new Customer();
        c1.money = 50;
        Customer c2 = new Customer();

        System.out.println("Before giving ...");
        System.out.println("c1 has $" + c1.money);
        System.out.println("c2 has $" + c2.money);
        c1.give(c2, 23.75f);
        System.out.println("After giving ...");
        System.out.println("c1 has $" + c1.money);
        System.out.println("c2 has $" + c2.money);
    }
}
```

Notice that **Customer c1** gives **$23.75** to **Customer c2**. Of course, for this code to work, we would have to write the **give** method. Where do we write that method ? Well, just like the **spend** method, the **give** method is supposed to work on **Customer** objects (you can see this by looking at the object in front of **.give** in the code above ... which is **c1** ... which is a **Customer** object). So, we write the method in the **Customer** class. See if you can write the method. Notice that the method takes two parameters. The first being a **Customer** object and the second being the **amount** to be given to that **Customer**. Think about how each of the objects change as a result of this method call. That is, think about how the instance variables change for each **Customer** (i.e., **name**, **age**, **sex**, **money**). I am sure that you will see that only one of the attributes change in this particular case. Try out the code above to test it. For fun, try changing **c1.give(c2, 23.75f);** to **c1.give(c1, 23.75f);** (i.e., the customer gives money to himself) and see if it still works. Do you understand what is happening ?

STEP 5. Assume now that we want to compute an admission fee for customers that wish to enter an event (e.g., such as a circus). We will write a function to compute an appropriate fee for a Customer object. We will use the following test code:

```
public class CustomerTestProgram {
    public static void main(String args[]) {
        Customer c1 = new Customer();
        c1.name = "Bob";
        c1.age = 18;
        Customer c2 = new Customer();
        c2.name = "Dottie";
        c2.age = 3;
        Customer c3 = new Customer();
        c3.name = "Steve";
        c3.age = 68;
        Customer c4 = new Customer();
        c4.name = "Jane";
        c4.age = 66;

        System.out.println("The fee for Bob is $" + c1.computeFee());
        System.out.println("The fee for Dottie is $" +
        c2.computeFee());
        System.out.println("The fee for Steve is $" +
        c3.computeFee());
        System.out.println("The fee for Jane is $" + c4.computeFee());
    }
}
```

For this code to work, you will need to write a computeFee method in the **Customer** class that computes and returns a float representing the fee. Here is how to compute the fee. The basic adult fee is **$12.75** and applies to anyone **18** years of age or older. Children **3** and under get in free and anyone over the age of **65** receives a **50%** discount. Children and youths (between the age of **4** to **17**) pay only **$8.50**. OK? Go ahead and write the method, then test it out.

STEP 6. Now we will modify our test code to make use of our **computeFee** method to enter the event (i.e., circus). Here is the test scenario:

```
public class CustomerTestProgram {
    public static void main(String args[]) {
        Customer c1 = new Customer();
        c1.name = "Bob";
        c1.age = 17;
        c1.money = 10;
        Customer c2 = new Customer();
        c2.name = "Dottie";
        c2.age = 3;
        c2.money = 0;
        Customer c3 = new Customer();
        c3.name = "Steve";
        c3.age = 67;
        c3.money = 30;
        Customer c4 = new Customer();
        c4.name = "Jane";
        c4.age = 64;
        c4.money = 0;

         System.out.println("Here is the situation before going into
         the circus:");
         System.out.println("  Bob has $" + c1.money);
         System.out.println("  Dottie has $" + c2.money);
         System.out.println("  Steve has $" + c3.money);
         System.out.println("  Jane has $" + c4.money);

         // Simulate people going into the circus
         c1.payAdmission();
         c2.payAdmission();
         c3.payAdmission();
         c4.payAdmission();
         c3.give(c4, 15);
         c4.payAdmission();

         System.out.println("Here is the situation after going into the
         circus:");
         System.out.println("  Bob has $" + c1.money);
         System.out.println("  Dottie has $" + c2.money);
         System.out.println("  Steve has $" + c3.money);
         System.out.println("  Jane has $" + c4.money);
     }
}
```

You need to write the **payAdmission** method. It should make use of the **computeFee** and **spend** methods that you wrote earlier. Write the method and then test the code. Examine the output to make sure that it is correct.

STEP 7. In part 3 of this lab, a problem that occurs from the test program's point of view is that when we call the **spend** method, we do not really know if the operation was ignored (due to insufficient funds) or if the operation was successful. We could examine the customer's **money** variable before the call and then after the call and compare them to see if there was a change. But this is cumbersome. A better way would be to provide feedback to the caller of the method (i.e., the test program) by returning a value from the method to indicate whether or not it worked. We can return a **boolean** indicating whether or not the spending was successful. Modify your **spend** method so that it returns **true** if the spending was possible and **false** otherwise.

You can then change your testing code to look as follows:

```
public class CustomerTestProgram {
    public static void main(String args[]) {
        Customer c = new Customer();
        c.money = 50;

        System.out.println(c.money);
        if (!c.spend(10))
          System.out.println("Unable to spend $10");
        if (!c.spend(4.75f))
          System.out.println("Unable to spend $4.75");
        if (!c.spend(15.25f))
          System.out.println("Unable to spend $15.25");
        if (!c.spend(100))
          System.out.println("Unable to spend $100");
        System.out.println(c.money);

        if (!c.spend(100))
          System.out.println("Unable to spend $100");
    }
}
```

Try your changed code to make sure that it works.

STEP 8. Try adding a **boolean** attribute (i.e., instance variable) to the **`Customer`** class called **admitted**. Then modify your **payAdmission** method so that this variable is set to **true** when the customer has successfully paid the proper fee amount. Adjust your testing code in part 6 of this lab to display this **boolean** value after each call to **payAdmission** so that you can see if it is set properly. Try playing with the **money** amounts and see what happens.