COMP 1406
Winter 2020 - Tutorial #3

Objectives

- Learn how to create multiple objects that interact together
- Practice using arrays of object
- Understand the private and public access modifiers
- Design classes that make good use of encapsulation

Tutorial Problems:

1) Create the following *simplified* **Customer** class:

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

     public Customer(String n, int a, float m) {
          name = n;
          age = a;
          money = m;
     }

     public String toString() {
          return "Customer " + name + ": a " + age + " year old
with $" + money;
     }
}
```

Create the following simple **Store** class:

```
public class Store {
    public static final int MAX_CUSTOMERS = 500;

    String       name;
    Customer[]   customers;
    int          customerCount;

    public Store(String n) {
        name = n;
        customers = new Customer[MAX_CUSTOMERS];
```

```
            customerCount = 0;
        }

        public void addCustomer(Customer c) {
            if (customerCount < MAX_CUSTOMERS)
                customers[customerCount++] = c;
        }

        public void listCustomers() {
            for (int i=0; i<customerCount; i++)
                System.out.println(customers[i]);

        }
    }
```

Create the following simple **StoreTestProgram** class or download the StoreTestProgram.java file from cuLearn:

```
    public class StoreTestProgram {
        public static void main(String args[]) {
            Customer[]    result;
            Store         walmart;

            walmart = new Store("Walmart off Innes");
            walmart.addCustomer(new Customer("Amie", 14, 100));
            walmart.addCustomer(new Customer("Brad", 15, 0));
            walmart.addCustomer(new Customer("Cory", 10, 100));
            walmart.addCustomer(new Customer("Dave",  5, 48));
            walmart.addCustomer(new Customer("Earl", 21, 500));
            walmart.addCustomer(new Customer("Flem", 18, 1));
            walmart.addCustomer(new Customer("Gary",  8, 20));
            walmart.addCustomer(new Customer("Hugh", 65, 30));
            walmart.addCustomer(new Customer("Iggy", 43, 74));
            walmart.addCustomer(new Customer("Joan", 55, 32));
            walmart.addCustomer(new Customer("Kyle", 16, 88));
            walmart.addCustomer(new Customer("Smaug", 12, 1000));
            walmart.addCustomer(new Customer("Mary", 17, 6));
            walmart.addCustomer(new Customer("Nick", 13, 2));
            walmart.addCustomer(new Customer("Omar", 18, 24));
            walmart.addCustomer(new Customer("Patt", 24, 45));
            walmart.addCustomer(new Customer("Quin", 42, 355));
            walmart.addCustomer(new Customer("Ruth", 45, 119));
            walmart.addCustomer(new Customer("Snow", 74, 20));
            walmart.addCustomer(new Customer("Tamy", 88, 25));
            walmart.addCustomer(new Customer("Ulsa",  2, 75));
            walmart.addCustomer(new Customer("Vern",  9, 90));
            walmart.addCustomer(new Customer("Will", 11, 220));
            walmart.addCustomer(new Customer("Xeon", 17, 453));
            walmart.addCustomer(new Customer("Ying", 19, 76));
            walmart.addCustomer(new Customer("Zack", 22, 35));

            System.out.println("Here are the customers:\n");
            walmart.listCustomers();
        }
    }
```

2) Write a method in the **Store** class called **averageCustomerAge()** which should return an **int** representing the average age of all **Customer** objects in the store. Test your code by appending the following line to the end of the **StoreTestProgram**. You should get a result of **26**.

```
// Find average Customer age
System.out.println("\nAverage age of customers: " +
walmart.averageCustomerAge());
```

---

3) Write a method in the **Store** class called **richestCustomer()** which should return a **Customer** object representing the **Customer** in the store who has the most money. Make sure to return the whole **Customer** object, not just the **name**. Test your code by appending the following lines to the end of the **StoreTestProgram**. The result should be **Smaug** who has $1000:

```
// Find richest customer
System.out.println("\nRichest customer is: " +
walmart.richestCustomer());
```

---

4) Write a method called **rob(Customer c)** in the **Store** class. This method should reduce customer c's money to 0. Test your code by appending the following lines to the end of the **StoreTestProgram**:

```
//Make Smaug's walmart experience quite miserable
walmart.rob(walmart.customers[11]);
System.out.println("\n Smaug is now: " + walmart.customers[11]);
```

Clearly, Smaug is having a bad day. In the real world, it would not be acceptable to allow a Walmart store to rob a customer!

We have written our code in such a way that made all of our **Customer** attributes visible and modifiable. That is, we are allowing any class object (e.g., a Store, a Lawyer's office, a Movie Theatre, etc..) to mess around with the internal data stored in our customers.

This approach is not safe, nor advisable and represents poor coding style.
To prevent this, we need to use ***encapsulation***. That is, we need to hide information that others do not need to know and to also prevent others from modifying some internal attributes (i.e., the private parts) of our objects.

Fortunately, this is easily done in JAVA by using the **private** access modifier.

A. Go to your **Customer** class and write the word **private** in front of all attributes:

```
private String      name;
private int         age;
private float       money;
```

Now you will notice that the **Store** and **StoreTestProgram** will no longer compile. That is because they are directly accessing the internal attributes of the **Customer** object.

We will need to go back into that code and fix it. But first**, we need to decide which attributes should be visible to the world.** That is, which attributes are ok to be seen and accessed by other objects.

B. Well, for the sake of our tutorial, let's assume that the **name** and **age** should be publicly visible (i.e., we are willing to share that information with everybody) but that nobody should know how much money we have.

Currently, we are already preventing everyone outside the **Customer** class from seeing how much **money** a **Customer** has. So we just need to allow them to see the other attributes.

C. To do this, we make what are called ***get methods***. Get methods are public methods that will get the values for us. Write the following in the Customer class:

```
public String getName() { return name; }
```

D. Now, write a similar method called **getAge()** that returns the values of the **age** attribute.

Go into the **Store** class and **StoreTestProgram** class and everywhere that you notice that a **Customer**'s age attribute is being accessed directly.

E. You should adjust the code to use your new ***get method***.
For example, `customers[i].age` will become `customers[i].getAge()`.

Note that you will still have issues where the **money** attribute is accessed, because we did not make a ***get method*** for the **money** attribute. We will need to make a few changes to fix this. For starters, we can decide that we don’t want a store to be able to rob any customers:

F. Delete the **rob()** method.

Now, all that remains is to fix the **richestCustomer()** method, which still accesses the **money** attribute.
Since we did not create a ***get method*** for the **money**, we cannot even access it from the **Store** class and **money** is only visible within the **Customer** class.
Depending on how you wrote your code, notice what the code in the **if** statement is doing.
It compares the **money** attributes of two **Customer** objects. We can actually write a method that does this within the **Customer** class.

G. Create a method in the **Customer** class called **hasMoreMoneyThan(Customer c)** which returns **true** if the customer calling the method has more **money** than the customer **c**, otherwise it should return **false**.

H. Now adjust your code in the **Store** class's **richestCustomer()** method to make use of this newly created **hasMoreMoneyThan()** method. Now the code no longer accesses the money attribute of any **Customer** object and has no idea how much **money** each **Customer** has.

We have successfully encapsulated our data and protected it from being altered by anyone outside of the class.

---

5) We still have a slight problem from our encapsulation in the previous step. We have added a restriction that NONE of the attributes can ever be altered, even if we wanted them to. For example, assume that we want the **Customers** to each have a unique ID.

A. Add an **int** attribute called **id** to the **Customer** class and create an appropriate ***get method*** within the class.

B. Set the **id** to -1 in the constructor.

   Now ... who should be responsible to setting the **Customer** **id**?
   Likely the **Store** itself. This could be done, perhaps when a Customer makes a purchase for the first time.

C. Add a **private static** class variable called **LATEST_ID** to the **Store** class and set it to **100000** by default. Make sure that you did not make it **final**, because it will change.

D. Adjust the method called **addCustomer(Customer c)** in the **Store** class so that each **Customer** added gets a unique id (i.e., just increment the **LATEST_ID** as a counter). You will have trouble getting the code to compile because we made **id** to be **private** in the **Customer** class, so we can not access nor modify it.

E. To fix this problem, add a ***set method*** to the **Customer** class. The method should take an integer ID value as an argument and update the customer's id attribute.

F. Now go back to the **addCustomer()** method in the **Store** class and make use of this ***set method*** to fix the problem. We have successfully accomplished what we needed to do. We are allowing the **Store** class to modify a **private** attribute in the **Customer** class through use of a ***set method***. Just remember ... only create ***set methods*** for attributes that you are willing to be modifiable.

G. Go back to the **Store** class and make all the attributes **private**.

H. Add the following ***get methods*** in preparation for the next question:

```
public Customer[] getCustomers() {
    return customers;
}

public int getCustomerCount() {
    return customerCount;
}
```

MORE CHALLENGING QUESTIONS:

6) Create the following **Mall** class:

```
public class Mall {
    public static final int MAX_STORES = 100;

    private String   name;
    private Store[]  stores;
    private int      storeCount;

    public Mall(String n) {
        name = n;
        stores = new Store[MAX_STORES];
        storeCount = 0;
    }

    public void addStore(Store s) {
        if (storeCount < MAX_STORES) {
            stores[storeCount++] = s;
        }
    }
}
```

A. Write a method in the **Mall** class called **shoppedAtSameStore(Customer c1, Customer c2)** which returns **true** if the two given customers have shopped at the same store and **false** otherwise.

B. Use the following class to test your method; you can download a copy from cuLearn. The result should be **true**, **false**.

```
public class MallTestProgram {
    public static void main(String args[]) {
        // Make the mall
        Mall    trainyards = new Mall("Trainyards");

        // Make some stores
        Store   walmart, dollarama, michaels, farmBoy;
        trainyards.addStore(walmart = new Store("Walmart"));
        trainyards.addStore(dollarama = new Store("Dollarama"));
        trainyards.addStore(michaels = new Store("Michaels"));
        trainyards.addStore(farmBoy = new Store("Farm Boy"));

        // Create the customers
        Customer    c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11,
c12, c13, c14;
        Customer    c15, c16, c17, c18, c19, c20, c21, c22, c23,
c24, c25, c26;
        c1 = new Customer("Amie", 14, 100);
        c2 = new Customer("Brad", 15, 0);
        c3 = new Customer("Cory", 10, 100);
        c4 = new Customer("Dave", 5, 48);
        c5 = new Customer("Earl", 21, 500);
        c6 = new Customer("Flem", 18, 1);
```

```java
c7 = new Customer("Gary", 8, 20);
c8 = new Customer("Hugh", 65, 30);
c9 = new Customer("Iggy", 43, 74);
c10 = new Customer("Joan", 55, 32);
c11 = new Customer("Kyle", 16, 88);
c12 = new Customer("Lore", 12, 1000);
c13 = new Customer("Mary", 17, 6);
c14 = new Customer("Nick", 13, 2);
c15 = new Customer("Omar", 18, 24);
c16 = new Customer("Patt", 24, 45);
c17 = new Customer("Quin", 42, 355);
c18 = new Customer("Ruth", 45, 119);
c19 = new Customer("Snow", 74, 20);
c20 = new Customer("Tamy", 88, 25);
c21 = new Customer("Ulsa", 2, 75);
c22 = new Customer("Vern", 9, 90);
c23 = new Customer("Will", 11, 220);
c24 = new Customer("Xeon", 17, 453);
c25 = new Customer("Ying", 19, 76);
c26 = new Customer("Zack", 22, 35);

// Add the customers to the stores
walmart.addCustomer(c1);
walmart.addCustomer(c3);
walmart.addCustomer(c4);
walmart.addCustomer(c5);
walmart.addCustomer(c8);
walmart.addCustomer(c12);
walmart.addCustomer(c13);
walmart.addCustomer(c14);
walmart.addCustomer(c17);
walmart.addCustomer(c19);
walmart.addCustomer(c25);
dollarama.addCustomer(c2);
dollarama.addCustomer(c3);
dollarama.addCustomer(c5);
dollarama.addCustomer(c6);
dollarama.addCustomer(c13);
dollarama.addCustomer(c16);
dollarama.addCustomer(c18);
dollarama.addCustomer(c19);
dollarama.addCustomer(c20);
michaels.addCustomer(c1);
michaels.addCustomer(c2);
michaels.addCustomer(c7);
michaels.addCustomer(c9);
michaels.addCustomer(c15);
michaels.addCustomer(c18);
michaels.addCustomer(c22);
michaels.addCustomer(c23);
michaels.addCustomer(c24);
michaels.addCustomer(c26);
farmBoy.addCustomer(c1);
farmBoy.addCustomer(c2);
farmBoy.addCustomer(c5);
farmBoy.addCustomer(c10);
farmBoy.addCustomer(c11);
```

```
                farmBoy.addCustomer(c19);
                farmBoy.addCustomer(c21);
                farmBoy.addCustomer(c24);
                farmBoy.addCustomer(c25);

                // Determine whether or not certain customers shopped at
the same store
                   System.out.println("Did Amie and Xeon shop at the same
store: " +
                                   trainyards.shoppedAtSameStore(c1,
c24));
                   System.out.println("Did Brad and Nick shop at the same
store: " +
                                   trainyards.shoppedAtSameStore(c2,
c14));
    }
}
```

---

7) Write a method in the **Mall** class called **getUniqueCustomerCount()** which returns an integer that represents the total number of unique **Customer** objects that were at the Mall that day.

Note that each **Store** potentially has some customers in it and that some customers visit multiple stores. So you will need to consider the total number of customers in the stores but you must not include duplicates (note: the ArrayList class may have useful functionality for this question).

A. Add the following line to the end of the **MallTestProgram** to test your method. The result should be **26**.

```
// Determine the number of unique Customers in the mall
System.out.println("The number of unique customers in the mall is: "
+ trainyards.getUniqueCustomerCount ());
```

---

8) Write a method in the **Store** class called **friendsFor(Customer lonelyCustomer)** which should return an array of all **Customer** objects in the store that are possible friends for the one specified in the parameter **lonelyCustomer**.

Customers will be considered "possible friends" if they are within 3 years of age of each other.

Test your code by appending the following lines to the end of the **StoreTestProgram**. You should see **7** friends for **Omar** and **6** friends for **Amie**.

```
// Find friends for Amie
System.out.println("\n\nFriends for 18 year old Omar:");
result = walmart.friendsFor(walmart.getCustomers()[14]);
for (Customer c:  result) System.out.println(c);
```

```
// Find friends for Brad
System.out.println("\n\nFriends for 14 year old Amie:");
result = walmart.friendsFor(walmart.getCustomers()[0]);
for (Customer c:  result) System.out.println(c);
```