COP 3330 – Object Oriented Programming in Java
More About Classes and Objects

GiftCard Class

Our next example will be the GiftCard class. For this implementation, a GiftCard will be assigned to a person (we'll store their first and last name), be good to use at a store, and have a balance. So the instance variables are:

```
private String firstName;
private String lastName;
private String storeName;
private double amount;
```

Here are the constructors we'll define for the GiftCard:

```
// Initializes a newly created GiftCard for the person with the name
// first last with no balance for the store store.
public GiftCard(String first, String last, String store)

// Initializes a newly created GiftCard for the person with the name
// first last with the balance equal to amt for the store store.
public GiftCard(String first, String last, String store, double amt)
```

Here are the methods we'll define for the GiftCard:

```
// If the balance on this GiftCard is amt or greater, amt is spent on
// this GiftCard and true is returned. Otherwise, false is returned and
// no change is made to the balance of this GiftCard.
public boolean spend(double amt)

// This transfer's the GiftCard to the person with the name first last.
public void transfer(String first, String last)

// Assuming that amt is positive, amt gets added to the balance of
// this GiftCard.
public void addAmount(double amt)
```

In addition, our GiftCard class will have these methods which are often part of classes defined in Java (more explanation later):

```
// Returns a String representation of this GiftCard.
public String toString();

// Returns true if and only if the Object g is equal to this GiftCard.
public boolean equals(Object g)

// Returns a negative integer if this GiftCard has less value than g,
// returns a positive integer if this GiftCard has more value than g, and
// returns 0 if this and g have the same value.
public int compareTo(GiftCard g)
```

Our constructors are quite straight-forward. Let's take a look at the code on the next page:

```
// Constructor for an "empty" gift card.
public GiftCard(String first, String last, String store) {
    firstName = first;
    lastName = last;
    storeName = store;
    amount = 0;
}

// This constructor initializes the card to amt dollars.
public GiftCard(String first, String last, double amt) {
    firstName = first;
    lastName = last;
    storeName = store;
    amount = amt;
}
```

As emphasized before, the most important thing is to make sure that the instance variables of the object, in this case, firstName, lastName and amount, get set to something (initialized). Thus, these three items should appear on the left hand side of assignment statements. A majority of constructors are fairly "boring" in this manner as they usually set each instance variable to the corresponding formal parameter. We did see some exceptions to this in the last lecture – for example, the Time_V1 constructor that took in a single integer (total number of seconds) and used it to set all three instance variables (hours, minutes, seconds).

When writing instance methods in a class, **you assume you have access to a specific object.** Within the method, the name of that object is this, but you are free to omit the keyword this. Typically, most programmers omit the keyword this unless it's necessary or they feel that adding it enhances the readability of the code. Here is an example of how the spend method would be written if you explicitly used the this keyword:

```
public boolean spend(double amt) {
    if (amt > this.amount)
        return false;
    this.amount -= amt;
    return true;
}
```

Here is how it appears in the posted code:

```
public boolean spend(double amt) {
    if (amt > amount)
        return false;
    amount -= amt;
    return true;
}
```

Here are methods for transfer and addAmount:

```
public void transfer(String first, String last) {
    firstName = first;
    lastName = last;
}


public void addAmount(double amt) {
    if (amt > 0)
        amount += amt;
}
```

For now, we'll not look at the equals and compareTo methods, but will analyze these when the corresponding topics are taught. Let's quickly look at the toString method:

```
public String toString() {
    DecimalFormat fmt = new DecimalFormat("0.##");
    return firstName+" "+lastName+" $"+fmt.format(amount);
}
```

We'll explain more on this later when we talk about inheritance and the Object class, but for now, just know that this returns a String representation of a GiftCard object and naturally gets instantiated anytime we print out a GiftCard object.

Now, let's look at two applications that use the GiftCard class, Shopping.java and Shopping2.java. In the first the user has a single GiftCard to Macy's and in the second the user has two GiftCard objects, one for Macy's and one for Starbucks.

Our Shopping example will only have one GiftCard and allow the user to use it to buy things, allow the user to add money to the GiftCard's balance and allow the user to transfer the GiftCard to someone else. Also, we'll give the user the option to print out information about the GiftCard.

In our Shopping class we'll ask their name and give them a $50 gift card. Once we ask their name, here's the line of code that creates the GiftCard object:

```
GiftCard macys = new GiftCard(first, last, "Macy's", 50.00);
```

(True story: Back in 2007 when I made this example, I used to teach at Winter Park High School and the parent group gave all of the IB teachers $50 gift cards to Macy's as a thank you Christmas present. It was pretty much the first time I visited the Mall of Millenia!)

Now, whenever we want to perform actions on this object, we can do things like:

```
if (macys.spend(cost))
    System.out.println("Great, your purchase has been made.");
```

Since there's just one GiftCard object, it's pretty clear that we'll call each method on macys and the code is reasonably straightforward.

Now, let's look at new example where the user has two GiftCards, one to Macy's and one to Starbucks.

In this example we'll ask the user to input the values of the two GiftCards and create each with those values. In our main method, the references to the GiftCards will be called macys and starbucks.

Since we have two stores now, for options 1 and 2, we'll create new methods that handle executing these transactions since it will now involve asking the user which store and how much and then doing the appropriate action.

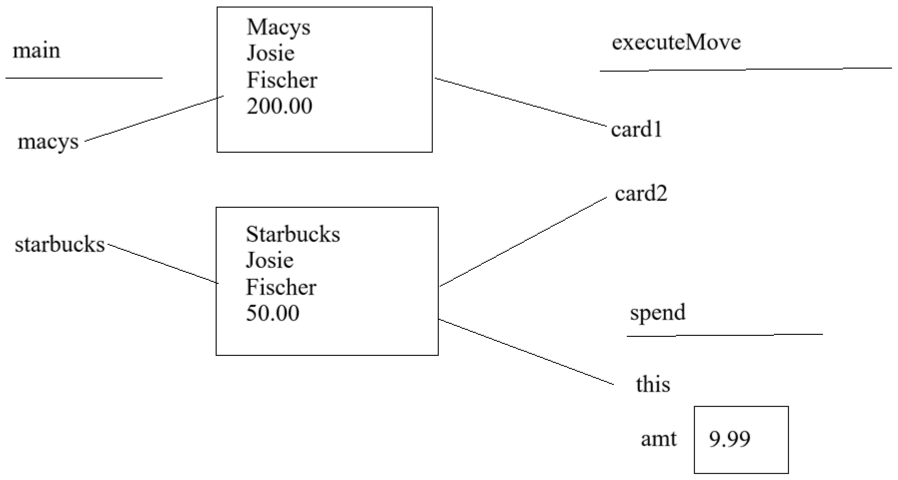
Here's the most complicated static method that handles the buy transaction:

```
public static void executePurchase(GiftCard card1, GiftCard card2) {

    System.out.println("Which store do you want to make a purchase from?");
    System.out.println("From "+card1.getStoreName()+" or from
                       "+card2.getStoreName());
    String store = stdin.next();

    if (store.equals(card1.getStoreName())) {

        System.out.println("Cost of your item from "+store+"?");
        double cost = stdin.nextDouble();

        if (card1.spend(cost))
            System.out.println("Your purchase was made from "+store+".");
        else
            System.out.println("You do not have enough for that purchase.");
    }

    else {

        store = card2.getStoreName();

        System.out.println("Cost of your item from "+store+"?");
        double cost = stdin.nextDouble();

        if (card2.spend(cost))
            System.out.println("Your purchase was made from "+store+".");
        else
            System.out.println("You do not have enough for that purchase.");

    }
}
```

We have a main if-else that handles the two different stores. Then, we execute the purchase from the appropriate store, which also involves its own if-else for when something is too expensive.

In this method, we are just going to make the appropriate method calls on either card1 or card2 as we need to.

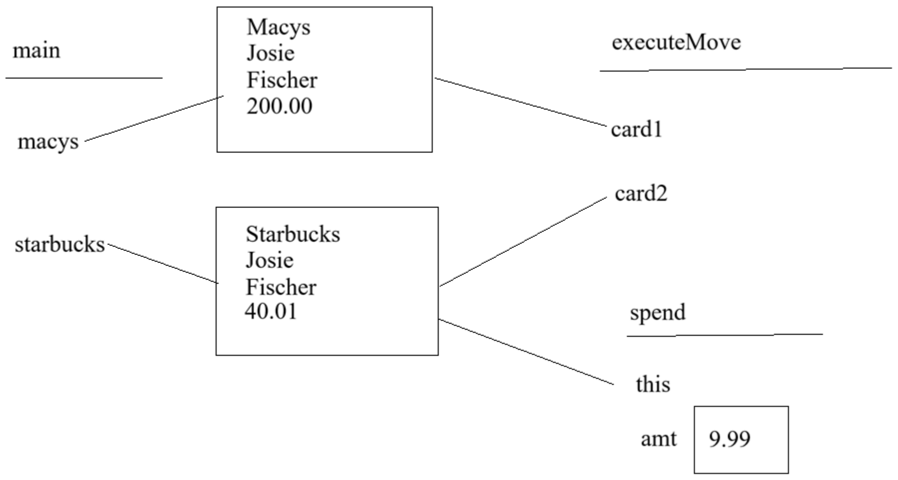
Let's just look at one picture of how everything is in memory when we're spending some money from our starbucks card. In this picture the user has entered 9.99 to spend from Starbucks:



When we execute the line of code:

```
amount -= amt;
```

in the spend method, the variable spend is automatically assumed to be part of this since it's not defined as a local variable in the spend method. Thus, the spend method will deduct 9.99 from the amount instance variable in the object referenced by this (which is the object referenced by starbucks in main):



Then, the method will return true to executeMove, which will then finish sending control back to main.