The contents of this document represent and go over the topics covered in lecture on 4/11 and 4/13 Use this to review the content as you please, also be sure not to use these in place of lecture as these documents will only cover lecture highlights and important details and do not represent the full scope of what you may be tested on. As always if you have any questions ask on ED!

Lecture 23 - 4/11/2023

Last week we learned about inheritance and how to extend other classes and the relationship between super and subclasses. Today we are going to talk about interfaces, which have inheritance like properties but those will be discussed later.

Interfaces are blueprints of classes in Java they contain method declarations and potentially static variables but nothing else, another way to think about Interfaces is that they are **fully abstract classes** In order to write interfaces in Java you would do it as follows:

```
public interface MyInterface {
     //method declarations
     public void myMethod();
}
```

In Lecture we specifically saw the Comparable Interface, which is defined as follows:

```
public interface Comparable<T>{
     public int compareTo(Comparable<T> other);
}
```

This is another example of a parameterized structure. ArrayLists were the first example that we saw, all that means is we must declare the type when we use it. For the Card class if we wished to implement the Comparable interface we would do so as follows:

```
public class Card implements Comparable<Card>{
     public int compareTo(Card other){
          //whatever you implement
     }
     //other Card class methods
}
```

When it comes to the compareTo() method convention is when you have a method call a.compareTo(b); If a is to come before b then a negative quantity is returned, if they are equal 0 and otherwise a positive quantity. To link this

back to inheritance we can make the declared type of objects, the interface they implement so for the card example above:

```
Comparable c = new Card('h', 12);
```

However like with inheritance we are only able to call any method from Comparable at compile time otherwise we will receive an error meaning the only valid method call would be:

```
c.compareTo(someOtherCardObject);
```

If you are unfamiliar with Polymorphism please refer to last week's lecture review! There was more stuff done on codio in Lecture 23 examples but in terms of content this is where I will end it for today.

Lecture 24 - 4/13/2023

Last week you were introduced to Exceptions and Exception Handling in Java. Wouldn't it be absolutely amazing if we could just prevent potential exceptions from occurring in the first place? Well by the tone of which you probably read that in your head you can probably guess that we very much can do this. In fact you are expected to do so on your next homework assignment (Homework 10). We can accomplish this using conditionals and we can avoid exceptions in one of two ways:

1. Check if the conditions for the exception are met, if they are ,throw an exception. You can throw an exception in the following way:

   ```
   throw new Exception();
   ```

2. Check if the conditions for the exception are not met, if this is true run the program as normal

You can see both of these in the works, in the BankAccount.java file in Lecture 24 on codio!

Something you'll notice is that there is a custom exception! Well just like we can make custom data types through class and instantiation of objects of the respective type we can do the same for Exceptions, all we have to do is extend some other related exception in order to inherit the necessary properties of Exceptions that will allow us to throw them in the first place!

Here is the code from the respective file defining a custom exception from the codio examples:

```
import java.io.*;

public class OverdrawException extends IllegalArgumentException{

    public OverdrawException(String m){
         super(m);
    }

    public OverdrawException()
    {
       super("Dude, you need money");
    }
}
```

Notice we extend a predefined exception, this is necessary as it allows us to inherit the properties of exceptions like I discussed earlier.

This lecture was quite light on content, be sure to ask on ED if you have any questions about anything!