Topic: Wrapper Classes.

OOP concepts involved: Class Instantiation and Declaration, Class Members, Polymorphism, Constructor Methods, Class Modifiers, Interfaces, Objects, Getters and Setters.

Programming generic concepts involved: Functions, Data Types, Variables, Access Modifiers.

➢ Theoric introduction

WRAPPER CLASSES

A **Wrapper class** is a class whose object contains a primitive data type. This wrapper class wraps around the data type and gives it an object appearance, hence the name. In other words we can convert a primitive data type into an object.

Each primitive data type has a wrapper class attached to it, most of the times its named the same as the primitive data type but with the a capital first letter.

Primitive	Wrapper Class	Constructor Arguments
boolean	Boolean	boolean or String
byte	Byte	byte or String
char	Character	char
double	Double	double or String
float	Float	float, double, or String
int	Integer	int or String
long	Long	long or String
short	Short	short or String

WHY SHOULD I USE WRAPPER CLASSES?

Wrapper classes have multiple utilities, most of which might not seem obvious at first:

- ❏ They convert primitive data types into Objects, this is especially useful if we need to modify the arguments passed into a method (Since java passes primitive types by value)
- ❏ Allows access to object related packages like java.util
- ❏ Data structures in the Collection Framework, such as Arraylist and Vector, only store objects and not primitive types
- ❏ An object is needed to support synchronization in multithreading

AUTOBOXING AND UNBOXING

Now that we have talked about the uses and importance of Wrapper classes, how do we use them? Java introduced two automatic conversions called Autoboxing and Unboxing

Autoboxing is the automatic conversion of primitive types into their corresponding wrapper class. For example, conversion of int to Integer, long to Long, etc. This is done using the form below.

```
[primitive data type] pdt_identifier = value;
[Wrapper class] wc_identifier = pdt_identifier;
```

```
char ch = ‘a’;
Character wc = ch;
```

Unboxing is the reverse process of autoboxing. Automatically converting a wrapper class back into is corresponding primitive data type. For Example, Integer to int, Long to long, etc.

```
[Wrapper class] wc_identifier = value;
[primitive data type] pdt_identifier = wc_identifier;
```

```
Character wc = ‘a’;
char ch = wc;
```

➢ Statement

Utilize the different Wrapper classes available to test some of the functions they provide: Conversions (hexadecimal to integer, binary to decimal, etc), and methods they get access for being an Object and not a primitive type.

➢ Program Code

Main.java

```
public class WrapperTest{
        public static void main (String args[])
        {
                Integer Obj1 = new Integer (44);
                Integer Obj2 = new Integer ("44");
                Integer Obj3 = new Integer (55);
                Float f1 = new Float("10.25f");
                Float f2 = new Float("12.63f");
                Float f3 = new Float(10.25f);
                Float f = Obj1.floatValue() + f1;


                //compareTo() method
                System.out.println("Using compareTo() Obj1 and Obj2: " +
Obj1.compareTo(Obj2));
                System.out.println("Using compareTo() Obj1 and Obj3: " +
Obj1.compareTo(Obj3));

                //Equals() method
                System.out.println("Using equals() Obj1 and Obj2: " + Obj1.equals(Obj2));
                System.out.println("Using equals() Obj1 and Obj3: " + Obj1.equals(Obj3));

                System.out.println("Using compare() f1 and f2: " +Float.compare(f1,f2));
                System.out.println("Using compare() f1 and f3: " +Float.compare(f1,f3));

                //Addition of Integer with Float
                System.out.println("Addition of intObj1 and f1: "+ Obj1 +"+" +f1+"=" +f );
                Integer int1 = Integer.valueOf("12345");

                //Converting from binary to decimal
                Integer int2 = Integer.valueOf("101011", 2);

                //Converting from hexadecimal to decimal
                Integer int3 = Integer.valueOf("D", 16);
```

```
                System.out.println("Value of int1 Object: "+ int1);
                System.out.println("Value of int2 Object: "+ int2);
                System.out.println("Value of int3 Object: "+ int3);
                System.out.println("Hex value of int1: " + Integer.toHexString(int1));
                System.out.println("Binary Value of int2: "+ Integer.toBinaryString(int2));
    }
}
```

➢ Program execution

By making them wrapper classes instead of primitive data types, we can utilize a lot of object related methods. During the program execution we can see the results of some of those methods, saving time not writing code.

```
Using compareTo() Obj1 and Obj2: 0
Using compareTo() Obj1 and Obj3: -1
Using equals() Obj1 and Obj2: true
Using equals() Obj1 and Obj3: false
Using compare() f1 and f2: -1
Using compare() f1 and f3: 0
Addition of intObj1 and f1: 44+10.25=54.25
Value of int1 Object: 12345
Value of int2 Object: 43
Value of int3 Object: 13
Hex value of int1: 3039
Binary Value of int2: 101011
```

➢ Conclusions

Wrapper classes are a great tool in java. Since the language is Object oriented, most of the classes and methods require an object to work with. Wrapper classes serve the function of converting primitive data (the only data that's not object oriented), into objects.