

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 its 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

Develop a program that captures a number and prints its value in binary, octal, decimal and hexadecimal.

➤ Program Code

Main.java

```
import java.util.*;

public class Main{

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Integer i1, i2;

        System.out.println("1st Number: ");
        i1 = new Integer(sc.nextLine());
        System.out.println("Binary: " + i1.toBinaryString(i1.intValue()));
        System.out.println("Octal: " + i1.toOctalString(i1.intValue()));
        System.out.println("Decimal: " + i1.toString(i1.intValue()));
        System.out.println("Hexadecimal: " + i1.toHexString(i1.intValue()));
    }
}
```

➤ Program execution

```
1st Number:
124
Binary: 1111100
Octal: 174
Decimal: 124
Hexadecimal: 7c
```

The use of Wrapper classes instead of primitive data types allowed to use the tools the Integer Class provides. In this case, being the methods toBinaryString, toOctalString and ToHexString for easy conversion between the bases.

➤ **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.