WRAPPER CLASSES

Wrapper Classes

Collection is an object, it able to store a group of other objects.

In java applications, Collection objects are able to store only objects, they will not store primitive data.

JAVA has provided 8 no of wrapper classes w.r.t the 8 number of primitive data types.

Primitive DTs	Wrapper Classes
byte	java.lang.Byte
short	java.lang.Short
int	java.lang.Integer
long	java.lang.Long
float	java.lang.Float
double	java.lang.Double
char	java.lang.Character
boolean	java.lang.Boolean

Note: Java has provided all the wrapper classes in the form of immutable classes.

Conversions from Primitive Type To Object Type:

A) By Using Parameterized Constructors From Wrapper Classes

public XXX(xxx value)
XXX ----> Wrapper classes
xxx ----> Primitive types.

```
EX:  int i = 10;
     Integer in = new Integer(i);
     System.out.println(i+"   "+in);
     OUTPUT: 10 10
```

B) By Using Valueof(-) Method Provided By Wrapper Classes:

public static XXX valueOf(xxx value)
XXX ----> Wrapper classes
xxx ----> primitive data types

```
EX:  int i = 10;
     Integer in = Intreger.valueOf(i);
     System.out.println(i+"    "+in);
     OUTPUT: 10   10
```

C) By Using Auto-Boxing Approach:

Auto-Boxing approach was provided by JDK5.0 version, in this approach no need to use pre defined methods and constructor, simply, we have to assign primitive variable to wrapper class reference variable.

EX: int i = 10;
Integer in = i;
System.out.println(i+" "+in);
OUTPUT: 10 10

2) Conversions From Object Type To Primitive Types:

a) By Using xxxValue() Method From Wrapper Classes:

public xxx xxxValue()
xxx ----> primitive data types
EX: Integer in = new Integer(10);
int i = in.intValue();
System.out.println(in+" "+i);
OUTPUT: 10 10

b) By using Auto-Unboxing:

Auto-Unboxing was provided by JDK 5.0 version, in this approach no need to use any predefined methods, simply assign wrapper class reference variables to the respective primitive variables.

EX: Integer in = new Integer(10);
int i = in;
System.out.println(in+" "+i);
OUTPUT: 10 10

3) Conversions from String Type to Object Type:

- **By Using String Parameterized Constructors From Wrapper Classes:**

public XXX(String value)
XXX----> Wrapper classes
EX: String data = "10";
Integer in = new Integer(data);
System.out.println(data+" "+in);
OUTPUT: 10 10

b) By Using Static valueOf(-) Method From Wrapper Classes:

public static XXX valueOf(String data)

EX: String data = "10";
Integer in = Integer.valueOf(data);
System.out.println(data+" "+in);
OUTPUT: 10 10

4) Conversions From Object Type To String Type:

a) By using toString() Method from Wrapper Classes:

public String toString()

EX: Integer in = new Integer(10);
String data = in.toString();
System.out.println(in+" "+data);
OUTPUT: 10 10

b) By Using '+' Concatenation Operator:

If we concatenate any reference variable with "" by using '+' operator then JVM will access toString() method over the provided reference variable.

EX: Integer in = new Integer(10);
String data = ""+in;
System.out.println(in+" "+data);
OUTPUT: 10 10

5. Conversions From Primitive Data Types To String Data Types:

- **By using Static toString() Method from Wrapper Classes:**

public static String toString(xxx value)

xxx----> Primitive types

EX: int i = 10;
String data = Integer.toString(i);
System.out.println(i+" "+data);
OUTPUT: 10 10

- **By using '+' Concatenation Operator:**

If we concatenate any primitive variable with "" by using '+' operator then the resultant value will be generated in String data type.

EX: int i = 10;
String data = ""+i;
System.out.println(i+" "+data);
OUTPUT: 10 10

6) Conversions From String Type To Primitive Type:

a) By using parseXXX() Method from Wrapper Classes:

public static xxx parseXxx(String data)

EX: **String data = "10";**
int i = Integer.parseInt(data);
System.out.println(data+" "+i);
OUTPUT: 10 10