Wrapper classes

Wrapper Class

- Wrapper classes are classes that allow primitive types to be accessed as objects.
- Wrapper class in java provides the mechanism to convert *primitive into object* and *object into primitive*.
- Wrapper class is wrapper around a primitive data type because they "wrap" the primitive data type into an object of that class.

Wrapper Classes

- Each of Java's eight primitive data types has a class dedicated to it.
- They are one per primitive type: Boolean, Byte, Character, Double, Float, Integer, Long and Short.
- Wrapper classes make the primitive type data to act as objects.

CODERINDEED

Primitive Data Types and Wrapper Classes

Data Type	Wrapper Class
byte	Byte
short	Short
int	***Integer***
long	Long
char	***Character***
float	Float
double	Double
boolean	Boolean

CODERINDEED

Why Wrapper Class?

- Most of the objects collection store objects and not primitive types.
- Primitive types can be used as object when required.
- As they are objects, they can be stored in any of the collection and pass this collection as parameters to the methods.
- Wrapper classes are used to be able to use the primitive data-types as objects.
- Many utility methods are provided by wrapper classes.

  To get these advantages we need to use wrapper classes.

Why Wrapper classes??

- They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).
- The classes in java.util package handles only objects and hence wrapper classes help in this case also.
- Data structures in the Collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.

CODERINDEED

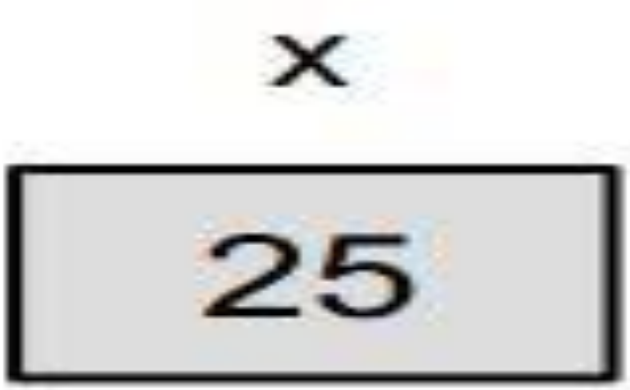
Difference b/w Primitive Data Type and Object of a Wrapper Class

- The following two statements illustrate the difference between a primitive data type and an object of a wrapper class:

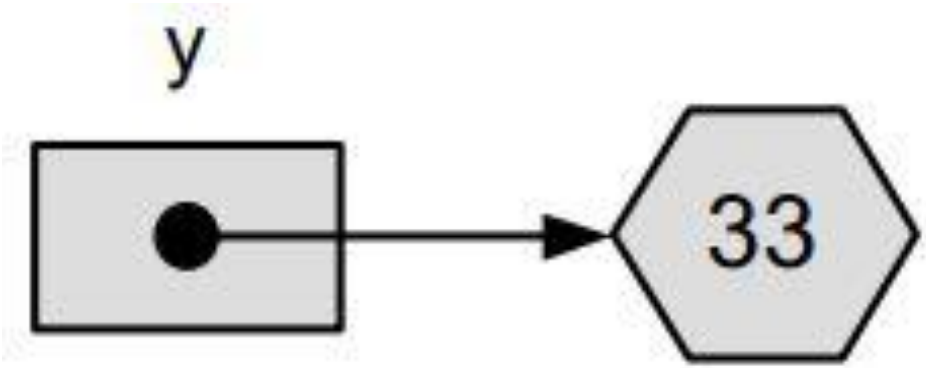
```
int x = 25;

Integer y = new Integer(33);
```

CODERINDEED

- The first statement declares an int variable named x and initializes it with the value 25.



- The second statement instantiates an Integer object. The object is initialized with the value 33 and a reference to the object is assigned to the object variable y.



- Clearly x and y differ by more than their values:

  x is a variable that holds a value;

  y is an object variable that holds a reference to an object.

CODERINDEED

Boxing and Unboxing

- The wrapping is done by the compiler.
- if we use a primitive where an object is expected, the compiler boxes the primitive in its wrapper class.
- Similarly, if we use a number object when a primitive is expected, the compiler un-boxes the object.

Example of boxing and unboxing:

- Integer x, y; x = 12; y = 15; System.out.println(x+y);
- When x and y are assigned integer values, the compiler boxes the integers because x and y are integer objects.
- In the println() statement, x and y are unboxed so that they can be added as integers.
- Boxing and unboxing can happen automatically, hence they are also known as AutoBoxing and Auto-UnBoxing

- **Autoboxing:** Converting a primitive value into an object of the corresponding wrapper class is called autoboxing. For example, converting int to Integer class.

The Java compiler applies autoboxing when a primitive value is:

- Passed as a parameter to a method that **expects an object** of the corresponding wrapper class.
- Assigned to a variable of the corresponding **wrapper class**.

- **Unboxing:** Converting an object of a wrapper type to its corresponding primitive value is called unboxing. For example conversion of Integer to int.

The Java compiler applies unboxing when an object of a wrapper class is:

- Passed as a parameter to a method that **expects a value** of the corresponding primitive type.
- Assigned to a variable of the corresponding **primitive type**.

Example

CODERINDEED

```
// Java program to illustrate the concept  of Autoboxing and Unboxing
class Example
{
        public static void main (String[] args)
        {
                // creating an Integer Object
                // with value 10.
                Integer i = new Integer(10);
                // unboxing the Object
                int i1 = i;
                System.out.println("Value of i: " + i);
                System.out.println("Value of i1: " + i1);
                //Autoboxing of char
                Character ch1 = 'a';
                // Auto-unboxing of Character
                char ch2 = ch1;
                System.out.println("Value of ch1: " + ch1);
                System.out.println("Value of ch2: " + ch2);

        }
}
```

Advantages of Autoboxing / Unboxing:

CODERINDEED

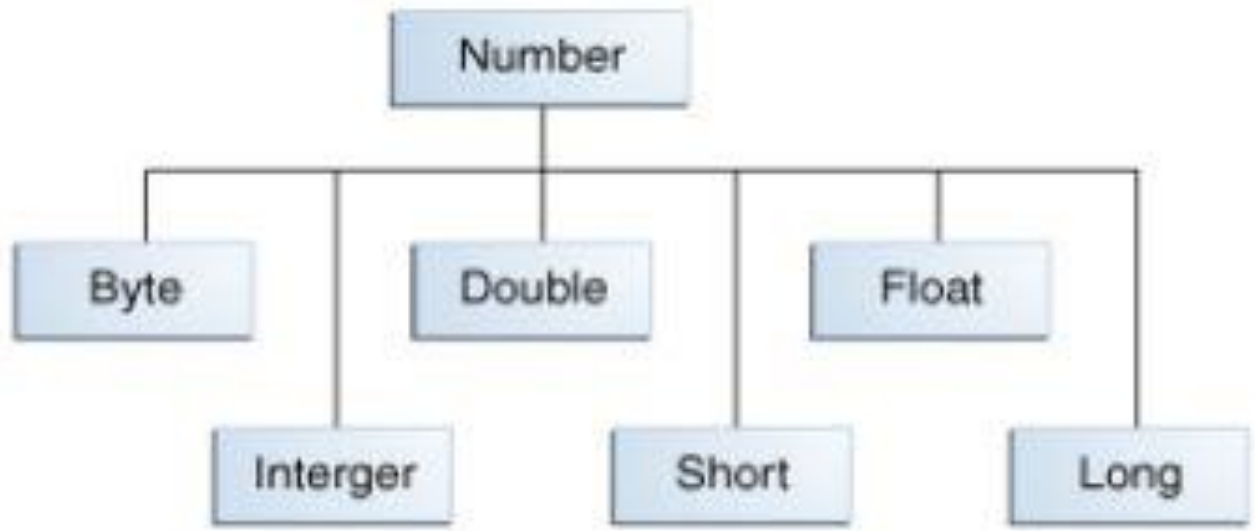
- Autoboxing and unboxing lets developers write cleaner code, making it easier to read.
- The technique let us use primitive types and Wrapper class objects interchangeably and we do not need to perform any typecasting explicitly.

CODERINDEED

Numeric Wrapper Classes

- All of the numeric wrapper classes are subclasses of the abstract class Number .

- All of them implements Comparable



Example

```
//Java program to demonstrate typeValue() method
public class Test
{
        public static void main(String[] args)
        {
        // Creating a Double Class object with value "6.9685"
                Double d = new Double("6.9685");
        // Converting this Double(Number) object to
                // different primitive data types
                byte b = d.byteValue();
                short s = d.shortValue();
                int i = d.intValue();
                long l = d.longValue();
                float f = d.floatValue();
                double d1 = d.doubleValue();
System.out.println("value of d after converting it to byte : " +b);
System.out.println("value of d after converting it to short : " + s
        System.out.println("value of d after converting it to int : " + i);
        System.out.println("value of d after converting it to long : " + l);
        System.out.println("value of d after converting it to float : " + f);
        System.out.println("value of d after converting it to double : " + d1);
        }
}
```

Output values:

6

6

6

6

6.9685

6.9685

CODERINDEED

Features of Numeric Wrapper Classes

- All the numeric wrapper classes provide a method to convert a numeric *string into a primitive value*.

public static type parseType (String Number)

- parseInt()
- parseFloat()
- parseDouble()
- parseLong()

…

Example 1

CODERINDEED

```
//Java program to demonstrate Integer.parseInt() method
public class Test
{
        public static void main(String[] args)
        {
                // parsing different strings
                int z = Integer.parseInt("654",8);
                int a = Integer.parseInt("-FF", 16);
        long l = Long.parseLong("2158611234",10);
                System.out.println(z);
                System.out.println(a);
                System.out.println(l);

// run-time NumberFormatException will occur here
                // “Hello" is not a parsable string
                int x = Integer.parseInt(“Hello",8);
// run-time NumberFormatException will occur here
        // (for octal(8),allowed digits are [0-7])
                int y = Integer.parseInt("99",8);

        }
}
```

```
428
-255
2158611234
Exception in thread "main"
java.lang.NumberFormatException:
For input string: Hello"
```

Example 2

CODERINDEED

```
//Java program to demonstrate Integer.parseInt() method
public class Test
{
        public static void main(String[] args)
        {
                // parsing different strings
                int z = Integer.parseInt("654");
        long l = Long.parseLong("2158611234");

                System.out.println(z);
                System.out.println(l);

// run-time NumberFormatException will occur here
                // “Hello" is not a parsable string
                int x = Integer.parseInt(“Hello");

// run-time NumberFormatException will occur here
        // (for decimal(10),allowed digits are [0-9])
                int a = Integer.parseInt("-FF");

        }
}
```

Output:

654

2158611234

Exception in thread "main" java.lang.NumberFormatException : For input string: “Hello"

CODERINDEED

Features of Numeric Wrapper Classes

- All the wrapper classes provide a static method toString to provide the *string representation of the primitive values.*

public static String toString (type value)

Example:

public static String toString (int a)

```
// Java program to illustrate toString()

class Test {
public static void main(String[] args)
        {
Integer I = new Integer(10);
        String s = I.toString();
        System.out.println(s);
        }
}
```


```
// Java program to illustrate toString()
```


```
class Test {
        public static void main(String[] args)
        {
                String s = Integer.toString(10);
                System.out.println(s);
                String s1 = Character.toString('a');
                System.out.println(s1);
        }
}
```

CODERINDEED

Features of Numeric Wrapper Classes

- All numeric wrapper classes have a static method valueOf, which is used to *create a new object initialized to the value* represented by the specified string.

public static DataType valueOf (String s)

Example:

```
Integer i = Integer.valueOf (“135”);
Double d = Double.valueOf (“13.5”);
```

CODERINDEED

Example 1

```
// Java program to illustrate valueof()

class Test {
        public static void main(String[] args)
        {
                Integer I = Integer.valueOf("10");
                System.out.println(I);
                Double D = Double.valueOf("10.0");
                System.out.println(D);
                Boolean B = Boolean.valueOf("true");
                System.out.println(B);

                // Here we will get RuntimeException
                Integer I1 = Integer.valueOf("ten");
        }
}
```

CODERINDEED

Example 2

```
// Java program to illustrate valueof()

class Test {
        public static void main(String[] args)
        {
                Integer I = Integer.valueOf(10);
                Double D = Double.valueOf(10.5);
                Character C = Character.valueOf('a');
                System.out.println(I);
                System.out.println(D);
                System.out.println(C);
        }
}
```

CODERINDEED

Methods implemented by subclasses of Number

- Compares this Number object to the argument.

  int compareTo(Byte anotherByte)
  int compareTo(Double anotherDouble)
  int compareTo(Float anotherFloat)
  int compareTo(Integer anotherInteger)
  int compareTo(Long anotherLong)
  int compareTo(Short anotherShort)

- returns int

CODERINDEED

Methods implemented by subclasses of Number

boolean equals(Object obj)

- Determines whether this number object is equal to the argument.
- The methods return true if the argument is not null and is an object of the same type and with the same numeric value.

Example

CODERINDEED

```
//Java program to demonstrate compareTo() method
public class Test
{
        public static void main(String[] args)
        {
// creating an Integer Class object with value "10"
        Integer i = new Integer("10");

        // comparing value of i
                System.out.println(i.compareTo(7));
                System.out.println(i.compareTo(11));
                System.out.println(i.compareTo(10));
        }
}
```

Output:

1

-1

0

Example

CODERINDEED

```
//Java program to demonstrate equals() method
public class Test
{
        public static void main(String[] args)
        {
// creating a Short Class object with value "15"
                Short s = new Short("15");
// creating a Short Class object with value "10"
                Short x = 10;
// creating an Integer Class object with value "15"
                Integer y = 15;
// creating another Short Class object with value "15"
                Short z = 15;
//comparing s with other objects
                System.out.println(s.equals(x));
                System.out.println(s.equals(y));
                System.out.println(s.equals(z));
        }
}
```

Output:

false

false

true

CODERINDEED

Character Class

- Character is a wrapper around a char.

- The constructor for Character is :

  Character(char ch)

  Here, ch specifies the character that will be wrapped by the Character object being created.

- To obtain the char value contained in a Character object, call charValue( ), shown here:

  char charValue( );

-

CODERINDEED

Boolean Class

- Boolean is a wrapper around boolean values.

- It defines these constructors:

  Boolean(boolean boolValue)

  Boolean(String boolString)

- In the first version, boolValue must be either true or false.

- In the second version, if boolString contains the string “true” (in uppercase or lowercase i.e TrUE, trUE), then the new Boolean object will be true. Otherwise, it will be false.

- To obtain a boolean value from a Boolean object, use booleanValue( ), shown here:

  boolean booleanValue( )

- It returns the boolean equivalent of the invoking object.

ARRAY LIST

CODERINDEED

ArrayList

ArrayList is a part of collection framework and is present in java.util package. It provides us with dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed. This class is found in java.util package.

Basic Example

CODERINDEED

```
import java.util.*;
class ArrayListExample {
        public static void main(String[] args)
        {
                int n = 5;
                ArrayList<Integer> arrli = new ArrayList<Integer>(n);
                for (int i = 1; i <= n; i++)
                arrli.add(i);
                System.out.println(arrli);
                arrli.remove(3);
                System.out.println(arrli);
                for (int i = 0; i < arrli.size(); i++)
                System.out.print(arrli.get(i) + " ");
        }
}
```

CODERINDEED

Various methods of ArrayList

Adding Elements: In order to add an element to an ArrayList, we can use the add() method. This method is overloaded to perform multiple operations based on different parameters. They are:

add(Object): This method is used to add an element at the end of the ArrayList.

add(int index, Object): This method is used to add an element at a specific index in the ArrayList.

CODERINDEED

Example of add Elements in ArrayList

```
// Java program to add elements
// to an ArrayList

import java.util.*;
public class ABC {
        public static void main(String args[])
        {
                ArrayList<String> al = new ArrayList<>();
                al.add("Welcome");
                al.add("Java");
                al.add(1, "to");
                System.out.println(al);
        }
}
```

Changing Elements: After adding the elements, if we wish to change the element, it can be done using the set() method. Since an ArrayList is indexed, the element which we wish to change is referenced by the index of the element. Therefore, this method takes an index and the updated element which needs to be inserted at that index.

CODERINDEED

```
// Java program to change elements
// in an ArrayList
import java.util.*;
public class ABC {
		public static void main(String args[])
		{
			ArrayList<String> al = new ArrayList<>();

			al.add("welcome");
			al.add("java");
			al.add(1, "to");

			System.out.println("Initial ArrayList " + al);

			al.set(1, "the");

			System.out.println("Updated ArrayList " + al);
		}
}
```

Removing Elements: In order to remove an element from an ArrayList, we can use the remove() method. This method is overloaded to perform multiple operations based on different parameters. They are:

remove(Object): This method is used to simply remove an object from the ArrayList. If there are multiple such objects, then the first occurrence of the object is removed.

remove(int index): Since an ArrayList is indexed, this method takes an integer value which simply removes the element present at that specific index in the ArrayList. After removing the element, all the elements are moved to the left to fill the space and the indices of the objects are updated.

CODERINDEED

```
// Java program to remove elements
// in an ArrayList

import java.util.*;
public class ABC {

        public static void main(String args[])
        {
                ArrayList<String> al = new ArrayList<>();

                al.add("Welcome");
                al.add("java");
                al.add(1, "to");
                System.out.println("Initial ArrayList " + al); // Welcome to java
                al.remove(1);
                System.out.println("After the Index Removal " + al); //Welcome java
                al.remove("java");
                System.out.println("After the Object Removal " + al); //Welcome
        }
}
```

Iterating the ArrayList: There are multiple ways to iterate through the ArrayList. The most famous ways are by using the basic for loop in combination with a get() method to get the element at a specific index and the advanced for loop.

CODERINDEED

```
import java.util.*;
public class ABC {
	public static void main(String args[])
	{
		ArrayList<String> al = new ArrayList<>();
		al.add("Welcome");
		al.add("java");
		al.add(1, "to");

		for (int i = 0; i < al.size(); i++) {

			System.out.print(al.get(i) + " ");
		}

		System.out.println();
		for (String str : al)
			System.out.print(str + " ");
	}
}
```

CODERINDEED

Important Features

- ArrayList inherits AbstractList class and implements List interface.
- ArrayList is initialized by the size. However, the size is increased automatically if the collection grows or shrinks if the objects are removed from the collection.
- Java ArrayList allows us to randomly access the list.
- ArrayList can not be used for primitive types, like int, char, etc. We need a wrapper class for such cases.
- ArrayList in Java can be seen as a vector in C++.
- ArrayList is not Synchronized. Its equivalent synchronized class in Java is Vector.

clear method()

CODERINDEED

The clear() method of ArrayList in Java is used to remove all the elements from a list. The list will be empty after this call returns.

Example:

```
import java.util.ArrayList;
public class ABC {
        public static void main(String[] args)
        {
                ArrayList<Integer> arr = new ArrayList<Integer>(4);
                arr.add(1);
                arr.add(2);
                arr.add(3);
                arr.add(4);
                System.out.println("The list initially: " + arr);
                arr.clear();
                System.out.println("The list after using clear() method: " + arr); //[]
        }
}
```

contains() Method

CODERINDEED

ArrayList contains() method in Java is used for checking if the specified element exists in the given list or not.

Example:

```
import java.util.ArrayList;
class ABC {
        public static void main(String[] args)
        {
                ArrayList<Integer> arr = new ArrayList<Integer>(4);
                arr.add(1);
                arr.add(2);
                arr.add(3);
                arr.add(4);
                boolean ans = arr.contains(2);
                if (ans)
                        System.out.println("The list contains 2");
                else
                        System.out.println("The list does not contains 2");
        }
}
```

Some other important Methods

int indexOf(Object o)

Returns the index in this list of the first occurrence of the specified element, or -1 if the List does not contain the element.

int lastIndexOf(Object o)

Returns the index in this list of the last occurrence of the specified element, or -1.

void ensureCapacity(int minCapacity)

Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.

boolean isEmpty()

used to check whether the Arraylist is empty or not?

trimToSize() Method

The **trimToSize()** method of **ArrayList** in Java trims the capacity of an ArrayList instance to be the list's current size. This method is used to trim an ArrayList instance to the number of elements it contains.