

Unit-2

Index

- 2.1 Strings: Exploring the String class, String buffer class,
- 2.2 Command-line arguments.
- 2.3 Library: StringTokenizer,
- 2.4. Random class,
- 2.5. Wrapper classes.
- 2.6. Encapsulation: Abstraction.
- 2.7. Creating User defined Data Structures: Array of Objects,
- 2.8. User defined Linked List.

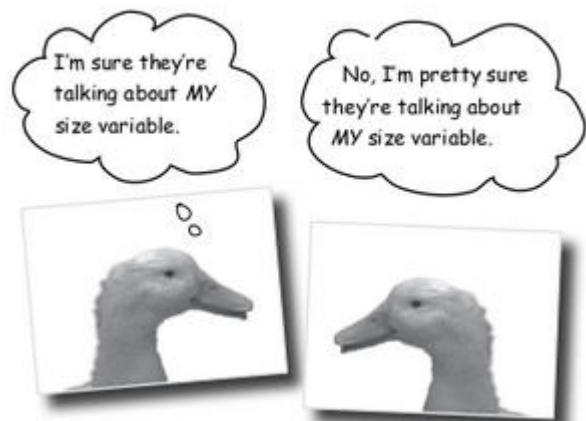


Static variables are shared.

All instances of the same class share a single copy of the static variables.

instance variables: 1 per **instance**
static variables: 1 per **class**

If you try to use an instance variable from inside a static method, the compiler thinks, "I don't know *which* object's instance variable you're talking about!" If you have ten Duck objects on the heap, a static method doesn't know about *any* of them.



String manipulations

Java.lang.String:-

String is used to represent group of characters or character array enclosed with in the double quotes.

Case 1:-String vs StringBuffer

String & StringBuffer both classes are final classes present in `java.lang` package.

Case 2:-String vs StringBuffer

We are able to create String object in two ways.

1) Without using new operator

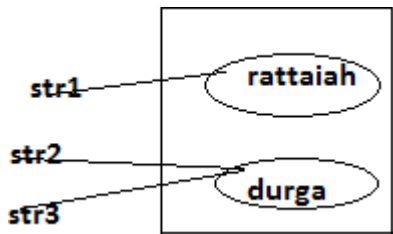
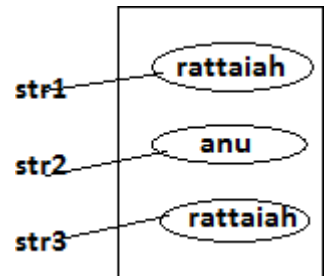
```
String str="krishna";
```

2) By using new operator

```
String str = new String("krishna");
```

We are able to create StringBuffer object only one approach by using new operator.

```
StringBuffer sb = new StringBuffer("krify technologies");
```

Creating a string object without using new operator :-	Creating a string object by using new operator
<ul style="list-style-type: none">➤ When we create String object without using new operator the objects are created in SCP (String constant pool) area.➤ <pre>String str1="rattaiah"; String str2="durga"; String str3="durga";</pre><p style="text-align: center;">SCP area</p>➤ When we create object in SCP area then just before object creation it is always checking previous objects.➤ If the previous object is available with the same content then it won't create new object that reference variable pointing to existing object.➤ If the previous objects are not available then JVM will create new object.➤ SCP area does not allow duplicate objects	<ul style="list-style-type: none">➤ Whenever we are creating String object by using new operator the object created in heap area. <pre>String str1=new String("rattaiah"); String str2 = new String("anu"); String str3 = new String("rattaiah");</pre>  <p style="text-align: center;">Heap area</p> <ul style="list-style-type: none">➤ When we create object in Heap area instead of checking previous objects it directly creates objects.➤ Heap memory allows duplicate objects

Example:-

class Test

```
{    public static void main(String[] args)
    {        //two approaches to create a String object
        String str1 = "krish";
        System.out.println(str1);
        String str2 = new String("anu");
        System.out.println(str2);

        //one approach to create StringBuffer Object (by using new operator)
        StringBuffer sb = new StringBuffer("krifysoft");
        System.out.println(sb);
    }
}
```

Case 3:- String

java.lang.String vs java.lang.StringBuffer:-

- String is **immutability class** it means once we are creating String objects it is not possible to perform modifications on existing object. (String object is fixed object)
- StringBuffer is a **mutability class** it means once we are creating StringBuffer objects on that existing object it is possible to perform modification.

Example :-

class Test

```
{    public static void main(String[] args)
    {        //immutability class (modifications on existing content not allowed)
        String str="ratan";
        str.concat("soft");
        System.out.println(str);

        //mutability class (modifications on existing content possible)
        StringBuffer sb = new StringBuffer("anu");
        sb.append("soft");
        System.out.println(sb);
    }
}
```

Java.lang.String class methods:-

Table 1: Some Most Commonly Used String Methods

Method Call	Task performed
<code>s2 = s1.toLowerCase;</code>	Converts the string s1 to all lowercase
<code>s2 = s1.toUpperCase;</code>	Converts the string s1 to all Uppercase
<code>s2 = s1.replace('x', 'y');</code>	Replace all appearances of x with y
<code>s2 = s1.trim();</code>	Remove white spaces at the beginning and end of the string s1
<code>s1.equals(s2)</code>	Returns 'true' if s1 is equal to s2
<code>s1.equalsIgnoreCase(s2)</code>	Returns 'true' if s1 = s2, ignoring the case of characters
<code>s1.length()</code>	Gives the length of s1
<code>s1.charAt(n)</code>	Gives nth character of s1
<code>s1.compareTo(s2)</code>	Returns negative if s1 < s2, positive if s1 > s2, and zero if s1 is equal s2
<code>s1.concat(s2)</code>	Concatenates s1 and s2
<code>s1.substring(n)</code>	Gives substring starting from n th character
<code>s1.substring(n, m)</code>	Gives substring starting from n th character up to m th (not including m th)
<code>String.valueOf(p)</code>	Creates a string object of the parameter p (simple type or object)
<code>p.toString()</code>	Creates a string representation of the object p
<code>s1.indexOf('x')</code>	Gives the position of the first occurrence of 'x' in the string s1
<code>s1.indexOf('x', n)</code>	Gives the position of 'x' that occurs after nth position in the string s1
<code>String.valueOf(Variable)</code>	Converts the parameter value to string representation

1) CompareTo(): -

- By using compareTo() we are comparing two strings character by character, such type of checking is called lexicographically checking or dictionary checking.
- compareTo() is return type is integer and it returns three values
 - a. zero ----- > if both String are equal
 - b. positive----- > if first string first character Unicode value is bigger than second String first character Unicode value then it returns positive.
 - c. Negative --- > if first string first character Unicode value is smaller than second string first character Unicode value then it returns negative.
- compareTo() method comparing two string with case sensitive.

class Test

```
{    public static void main(String... ratan)
    {        String str1="ratan";
              String str2="Sravya";
              String str3="ratan";
              System.out.println(str1.compareTo(str2)); //14
```

```

        System.out.println(str1.compareTo(str3)); //0
        System.out.println(str2.compareTo(str1)); // -13
        System.out.println("ratan".compareTo("RATAN")); // +ve
    }
};

```

Difference between **length()** method and length variable:-

- **length** variable used to find length of the Array.
- **length()** is method used to find length of the String.

Example :-

```

int [] a={10,20,30};
System.out.println(a.length);    //3

String str="rattaiah";
System.out.println(str.length()); //8

```

charAt(int):- By using above method we are able to extract the character from particular index position.

```
s1.charAt(n)
```

trim():- trim() is used to remove the trail and leading spaces. This method always used for memory saver.

```
s1.trim();
```

```

class Test
{
    public static void main(String[ ] args)
    {
        //charAt() method
        String str="ratan";
        System.out.println(str.charAt(1));
        //System.out.println(str.charAt(10)); StringIndexOutOfBoundsException
        char ch="ratan".charAt(2);
        System.out.println(ch);
        //trim()
        String ss="ratan";
        System.out.println(ss.length()); //7
        System.out.println(ss.trim()); //ratan
        System.out.println(ss.trim().length()); //5
    }
}

```

replace():-

replace() method used to replace the String or character.

toUpperCase() & toLowerCase():-

The above methods are used to convert lower case to upper case & upper case to lower case.

Example:-

```

class Test
{
    public static void main(String[] args)
    {
        String str="rattaiah how r u";
        System.out.println(str.replace('a','A'));//rAttAiAh
        System.out.println(str.replace("how","who"));//rattaiah who r u
        String str1="Sravya software solutions";
        System.out.println(str1);
        System.out.println(str1.replace("software","hardware"));// Sravya hardware solutions
        String str="ratan HOW R U";
        System.out.println(str.toUpperCase());
        System.out.println(str.toLowerCase());
        System.out.println("RATAN".toLowerCase());
        System.out.println("soft".toUpperCase());
    }
}

```

StringBuffer class methods:-

Table 2: Commonly Used StringBuffer Methods

Method	Task
s1.setCharAt(n, 'x')	Modifies the nth character to x
s1.append(s2)	Appends the string s2 to s1 at the end
s1.insert(n, s2)	Inserts the string s2 at the position n of the string s1
s1.setLength(n)	Sets the length of the string s1 to n. If n<s1.length() s1 is truncated. If n>s1.length() zeros are added to s1

reverse():-

```

class Test
{
    public static void main(String[] args)
    {
        StringBuffer sb=new StringBuffer("rattaiah");
        System.out.println(sb);
        System.out.println(sb.delete(1,3));
        System.out.println(sb);
        System.out.println(sb.deleteCharAt(1));
        System.out.println(sb.reverse());
    }
}

```

append():-

By using this method we can append the any values at the end of the string

Ex:-

```

class Test
{
    public static void main(String[] args)
    {
        StringBuffer sb=new StringBuffer("rattaiah");
        String str=" salary ";
        int a=60000;
        sb.append(str);
        sb.append(a);
    }
}

```

```

        System.out.println(sb);
    }
}

```

Insert():-

By using above method we are able to insert the string any location of the existing string.

```

class Test
{
    public static void main(String[] args)
    {
        StringBuffer sb=new StringBuffer("ratan");
        sb.insert(0,"hi ");
        System.out.println(sb);
    }
}

```

indexOf() and lastIndexOf():-

Ex:-

```

class Test
{
    public static void main(String[] args)
    {
        StringBuffer sb=new StringBuffer("hi ratan hi");
        int i;
        i=sb.indexOf("hi");
        System.out.println(i);
        i=sb.lastIndexOf("hi");
        System.out.println(i);
    }
}

```

replace():-

```

class Test
{
    public static void main(String[] args)
    {
        StringBuffer sb=new StringBuffer("hi ratan hi");
        sb.replace(0,2,"oy");
        System.out.println("after replaceing the string:-"+sb);
    }
}

```

Command Line Arguments

Main Method:-

public static void **main**(String[] args)

String[] args --> used to take command line arguments(the arguments passed from command prompt)

The arguments which are passed from command prompt is called **command line arguments**. We are passing command line arguments at the time program execution.

Example-1 :-

```
class Test
{
    public static void main(String[ ] leela)
    {
        System.out.println(leela[0]+" "+leela[1]);//printing command line arguments
        System.out.println(leela[0]+leela[1]);
        int a = Integer.parseInt(leela[0]); //conversion of String-int
        double d = Double.parseDouble(leela[1]); //conversion of String-double
        System.out.println(a+d);
    }
}
```

D:\>java Test 100 200
100 200
100200
300.0

Example 2: To provide the command line arguments with spaces , then take that command line argument with in **double quotes**.

```
class Test
{
    public static void main(String[ ] leela)
    {
        //printing command line arguments
        System.out.println(leela[0]);
        System.out.println(leela[1]);
    }
}
```

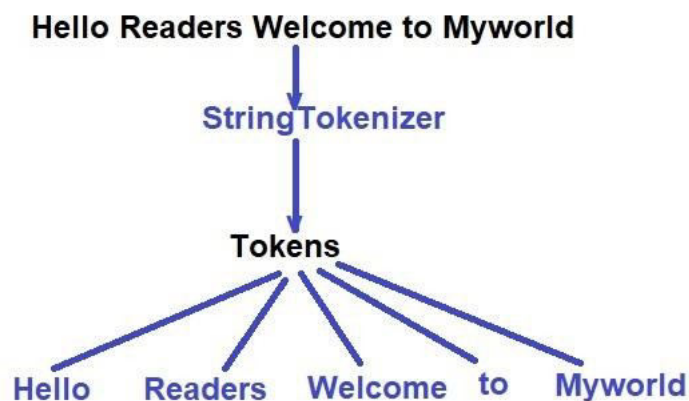
D:\>java Test corejava leela
corejava
leela
D:\>java Test core java leela
core
java
D:\>java Test "core java" leela
core java
leela

Java StringTokenizer

- ◆ In Java, StringTokenizer is used to break a string into tokens based on provided delimiter. Delimiter can be specified either at the time of object creation or on a per-token basis.
- ◆ Its object internally maintains a current position within the string to be tokenized. It is located into **java.util** package.
- ◆ In string, tokenizer objects are maintained internally and returns a token of a substring from the given string.

Note: StringTokenizer is a deprecated class and available only for compatibility reasons.

We can get idea from the below image, how tokenizer breaks the string into tokens.



Following are the constructors in stringTokenizer

1. StringTokenizer(String str)
2. StringTokenizer(String str, String delim)
3. StringTokenizer(String str, String delim, boolean returnValue)

Following are the methods in stringTokenizer

1. boolean hasMoreTokens()
2. String nextToken()
3. String nextToken(String delim)
4. boolean hasMoreElements()
5. Object nextElement()
6. int countTokens()

Example:

In this example, we are using StringTokenizer to break string into tokens based on space.

```
import java.util.StringTokenizer;
public class TokenDemo1
{
    public static void main(String args[])
    {
        StringTokenizer obj = new StringTokenizer("Welcome to my world"+" ");
        while (obj.hasMoreTokens())
        {
            System.out.println(obj.nextToken());
        }
    }
}
```

Output:

```
student@student-HP-245-G6-Notebook-PC:~$ javac TokenDemo1.java
student@student-HP-245-G6-Notebook-PC:~$ java TokenDemo1
Welcome
to
my
world
```

Example

Lets take another example to understand tokenizer, here we are breaking string into tokens based on the colon (:) delimiter.

```
import java.util.*;
public class TokenDemo2
{
    public static void main(String args[])
    {
        String a= " : ";
        String b= "Welcome : to : my : world : . : How : are : You : ?";
        StringTokenizer c = new StringTokenizer(b, a);
        int count1 = c.countTokens();
        for (int i = 0; i<count1; i++)
            System.out.println("token [" + i + "] : "+ c.nextToken());
        StringTokenizer d= null;
        while (c.hasMoreTokens())
            System.out.println(d.nextToken());
    }
}
```

Output:

```
student@student-HP-245-G6-Notebook-PC:~$ javac TokenDemo2.java
student@student-HP-245-G6-Notebook-PC:~$ java TokenDemo2
token [0] : Welcome
token [1] : to
token [2] : my
token [3] : world
token [4] : .
token [5] : How
token [6] : are
token [7] : You
token [8] : ?
```

Java Random Class

- Random class is part of java.util package.
- An instance of java Random class is used to generate random numbers.
- This class provides several methods to generate random numbers of type integer, double, long, float etc.
- Random number generation algorithm works on the seed value. If not provided, seed value is created from system nano time.
- If two Random instances have same seed value, then they will generate same sequence of random numbers.
- Java Random class is thread-safe, however in multithreaded environment it's advised to use `java.util.concurrent.ThreadLocalRandom` class.
- Random class instances are not suitable for security sensitive applications, better to use `java.security.SecureRandom` in those cases.

Java Random Constructors

Java Random class has two constructors which are given below:

1. `Random()`: creates new random generator
2. `Random(long seed)`: creates new random generator using specified seed

Java Random Class Methods

Let's have a look at some of the methods of java Random class.

1. `nextBoolean()`: This method returns next pseudorandom which is a boolean value from random number generator sequence.
2. `nextDouble()`: This method returns next pseudorandom which is double value between 0.0 and 1.0.
3. `nextFloat()`: This method returns next pseudorandom which is float value between 0.0 and 1.0.
4. `nextInt()`: This method returns next int value from random number generator sequence.
5. `nextInt(int n)`: This method return a pseudorandom which is int value between 0 and specified value from random number generator sequence

Java Random Example

Let's have a look at the below java Random example program.

```
import java.util.Random;
//Java Random Number Example Program

public class RandomNumberExample {

    public static void main(String[] args) {

        //initialize random number generator
        Random random = new Random();
```

```
        //generates boolean value
        System.out.println(random.nextBoolean());

        //generates double value
        System.out.println(random.nextDouble());

        //generates float value
        System.out.println(random.nextFloat());

        //generates int value
        System.out.println(random.nextInt());

        //generates int value within specific limit
        System.out.println(random.nextInt(20));

    }
}
```

Output of the above program is:

false

0.30986869120562854

0.6210066

-1348425743

18

Important points:

Random class:

==>The Random class implements a *random number generator*, which produces sequences of numbers that appear to be random.

==>To generate random integers , we construct an object of Random class , and then apply to the nextInt() method.

For example, the call generator.nextInt(6) gives us a random number between 0 and 5

Example 1: Testing a Random Integer for Negativity

```
//java program to test a Random integer for negativity
import java.util.Random;
class Testnegativity
{
    public static void main(String[] args)
    {
        Random r=new Random();
        int n=r.nextInt();
        System.out.println("n="+n);
        if(n<0)
        {
            System.out.println("****n<0");
        }
        System.out.println("Goodbye");
    }
}
student@student-HP-245-G6-Notebook-PC:~$ javac Testnegativity.java
student@student-HP-245-G6-Notebook-PC:~$ java Testnegativity
n=1288870215
Goodbye
student@student-HP-245-G6-Notebook-PC:~$ javac Testnegativity.java
student@student-HP-245-G6-Notebook-PC:~$ java Testnegativity
n=-740627638
****n<0
Goodbye
```

This program uses a random number generator to generate a random integer. It then reports whether the integer is negative:

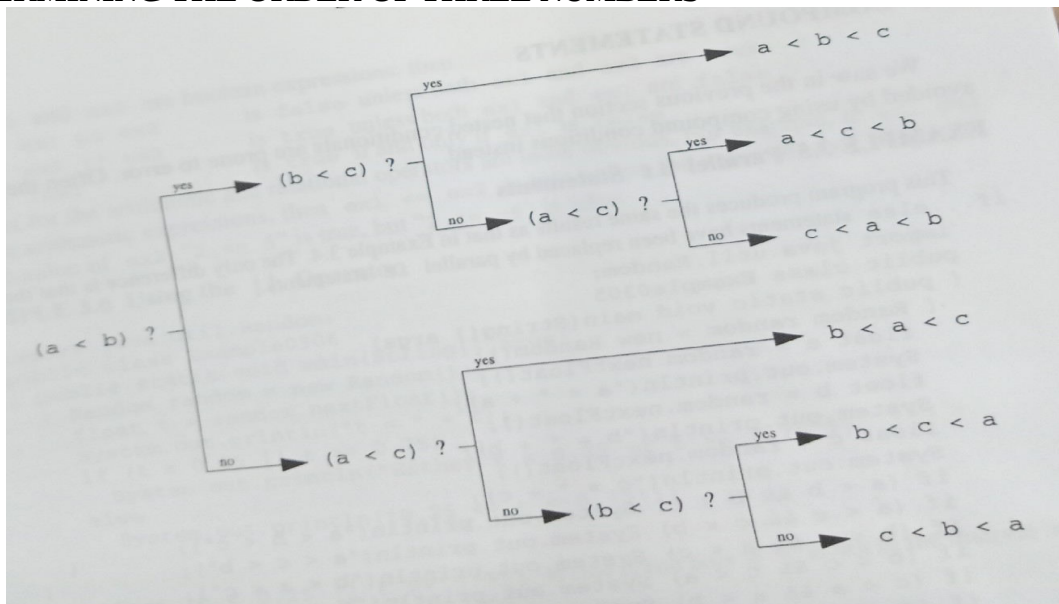
EXAMPLE 2: TESTING TWO RANDOM INTEGERS FOR THEIR MINIMUM USING if else STATEMENT:

```
//program to test 2 Random integers for their minimum
import java.util.Random;
class Testmin
{
    public static void main(String[] args)
    {
        Random r=new Random();
        int m=r.nextInt();
        System.out.println("m="+m);
        int n=r.nextInt();
        System.out.println("n="+n);
        if(m<n)
            System.out.println("The minimum is"+m);
        else
            System.out.println("The minimum is"+n);
    }
}
student@student-HP-245-G6-Notebook-PC:~$ javac Testmin.java
student@student-HP-245-G6-Notebook-PC:~$ java Testmin
m=593905876
n=20943430
The minimum is20943430
student@student-HP-245-G6-Notebook-PC:~$ javac Testmin.java
student@student-HP-245-G6-Notebook-PC:~$ java Testmin
m=-1626594304
n=431772941
The minimum is-1626594304
```

EXAMPLE 3: CHOOSING FROM FOUR ALTERNATIVES

```
//Java program for choosing among 4 alternatives
import java.util.Random;
class Testchoose
{
    public static void main(String[] args)
    {
        Random r=new Random();
        double t=r.nextDouble();
        System.out.println("t="+t);
        if(t<0.25)
            System.out.println("0<=t<1/4");
        else if(t<0.5)
            System.out.println("1/4<=t<1/2");
        else if(t<0.75)
            System.out.println("1/2<=t<3/4");
        else
            System.out.println("3/4<=t<1");
    }
}
student@student-HP-245-G6-Notebook-PC:~$ javac Testchoose.java
student@student-HP-245-G6-Notebook-PC:~$ java Testchoose
t=0.06985721594071936
0<=t<1/4
student@student-HP-245-G6-Notebook-PC:~$ javac Testchoose.java
student@student-HP-245-G6-Notebook-PC:~$ java Testchoose
t=0.7119579914650608
1/2<=t<3/4
```


EXAMPLE 4: DETERMINING THE ORDER OF THREE NUMBERS



/*java program that uses pairwise comparisons to determine the increasing order of three randomly generated real numbers using nested if*/

```

import java.util.Random;
class Nestedif
{
    public static void main(String[] args)
    {
        Random r=new Random();
        float a=r.nextFloat();
        System.out.println("a="+a);
        float b=r.nextFloat();
        System.out.println("b="+b);
        float c=r.nextFloat();
        System.out.println("c="+c);
        if(a<b)
            if(b<c)
                System.out.println("a<b<c");
            else if(a<c)
                System.out.println("a<c<b");
            else
                System.out.println("c<a<b");
        else
            if(a<c)
                System.out.println("b<a<c");
            else if(b<c)
                System.out.println("b<c<a");
            else
                System.out.println("c<b<a");
    }
}
student@student-HP-245-G6-Notebook-PC:~$ javac Order.java
student@student-HP-245-G6-Notebook-PC:~$ java Order
a=0.36980355
b=0.34350783
c=0.6177115
b<a<c
  
```

EXAMPLE 5 PARALLEL IF STATEMENTS:

/*java program that uses pairwise comparisons to determine the increasing order of three randomly generated real numbers using parallel if*/

```
import java.util.Random;
class Parallelif
{
    public static void main(String[] args)
    {
        Random r=new Random();
        float a=r.nextFloat();
        System.out.println("a="+a);
        float b=r.nextFloat();
        System.out.println("b="+b);
        float c=r.nextFloat();
        System.out.println("c="+c);
        if(a<b && b<c)
            System.out.println("a<b<c");
        if(a<c && c<b)
            System.out.println("a<c<b");
        if(b<a && a<c)
            System.out.println("b<a<c");
        if(b<c && c<a)
            System.out.println("b<c<a");
        if(c<a && a<b)
            System.out.println("c<a<b");
        if(c<b && b<a)
            System.out.println("c<b<a");
    }
}
student@student-HP-245-G6-Notebook-PC:~$ javac Parallelif.java
student@student-HP-245-G6-Notebook-PC:~$ java Parallelif
a=0.65420145
b=0.7130023
c=0.8444757
a<b<c
```

EXAMPLE 6:

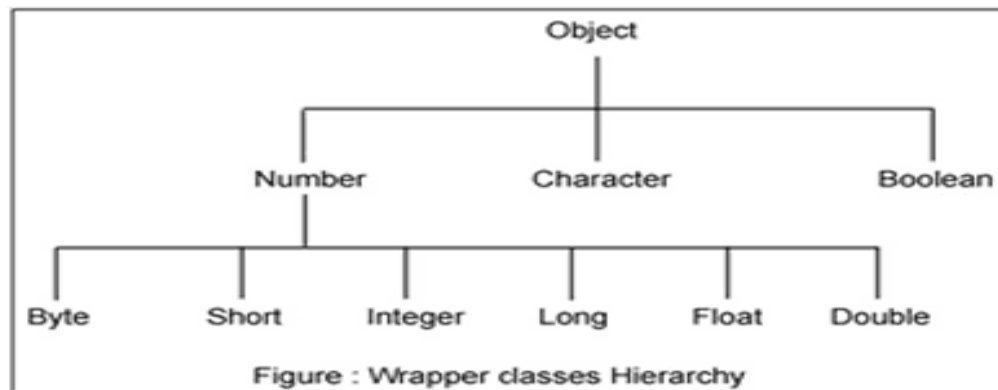
USING the || OPERATOR:

```
//java program using || operator
import java.util.Random;
class Logicalor
{
    public static void main(String[] args)
    {
        Random r= new Random();
        float t=r.nextFloat();
        System.out.println("t="+t);
        if(t<0.25 || t>=0.75)
            System.out.println("Either t<0.25 or t>=0.75");
        else
            System.out.println("0.25<=t<0.75");
    }
}
student@student-HP-245-G6-Notebook-PC:~$ javac Logicalor.java
student@student-HP-245-G6-Notebook-PC:~$ java Logicalor
t=0.90939236
Either t<0.25 or t>=0.75
student@student-HP-245-G6-Notebook-PC:~$ java Logicalor
t=0.12464261
Either t<0.25 or t>=0.75
```

Wrapper classes

- Java is an Object oriented programming language so represent everything in the form of the object, but java supports 8 primitive data types these all are not part of object.
- To represent 8 primitive data types in the form of object form we required 8 java classes these classes are called wrapper classes.
- All wrapper classes present in the **java.lang** package and these all classes are **immutable** classes.

Wrapper classes hierarchy:-



Wrapper classes constructors:-

```
Integer i = new Integer(10);  
Integer i1 = new Integer("100");  
Float f1= new Float(10.5);  
Float f1= new Float(10.5f);  
Float f1= new Float("10.5");  
Character ch = new Character('a');
```

datatypes	wrapper-class	Constructor argument
byte	Byte	byte, String
short	Short	short, String
int	Integer	int, String
long	Long	long, String
float	Float	double, float, String
double	Double	double, String
char	Character	char
boolean	Boolean	boolean, String

Note :- To create wrapper objects all most all wrapper classes contain two constructors but Float contains **three** constructors(float, double, String) & char contains **one** constructor(char).

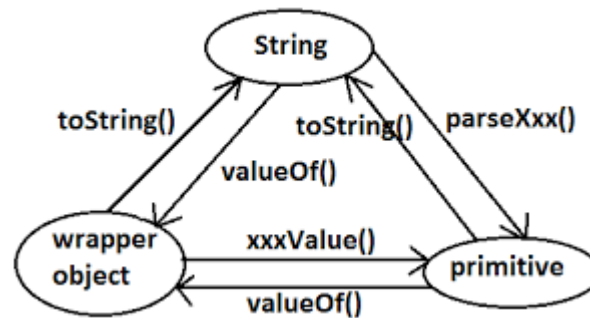
The most common methods of the wrapper class are

1. `valueOf()`
2. `xxxValue()`-----`intValue()`, `floatValue()`, `doubleValue()` etc
3. `toString()`
4. `parseXxx()`-----`parseInt()`, `parseFloat()` etc

1. `valueOf()`:-

in java we are able to create wrapper object in two ways.

- a) By using constructor approach
 - b) By using `valueOf()` method
- ✓ `valueOf()` method is used to create wrapper object just it is alternate to constructor approach and it a static method present in wrapper classes.



Example:-

class Test

```
{    public static void main(String[] args)
    {    //constructor approach to create wrapper object
        Integer i1 = new Integer(100);
        System.out.println(i1);

        Integer i2 = new Integer("100");
        System.out.println(i2);

        //valueOf() method to create Wrapper object
        Integer a1 = Integer.valueOf(10);
        System.out.println(a1);

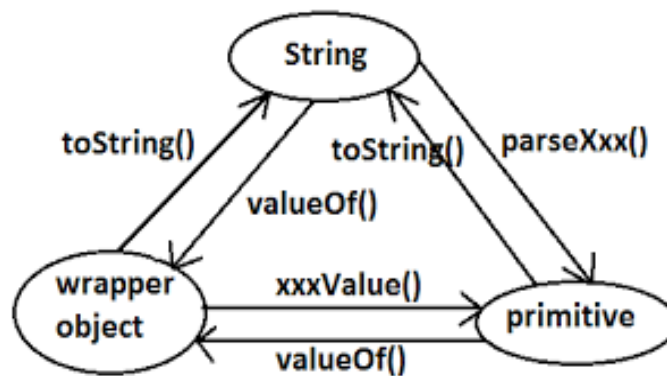
        Integer a2 = Integer.valueOf("1000");
        System.out.println(a2);
    }
}
```

Example :-conversion of primitive to String.

```
class Test
{
    public static void main(String[] args)
    {
        int a=100;
        int b=200;
        System.out.println(a+b);

        //primitive to String object
        String str1 = String.valueOf(a);
        String str2 = String.valueOf(b);
        System.out.println(str1+str2);
    }
}
```

2. xxxValue()::- It is used to convert wrapper object into corresponding primitive value.



The most common methods of the Integer wrapper class related to `xxxValue()` are summarized in below table.

METHOD	PURPOSE
byteValue()	returns the value of this Integer as a byte
doubleValue()	returns the value of this Integer as a double
floatValue()	returns the value of this Integer as a float
intValue()	returns the value of this Integer as an int
shortValue()	returns the value of this Integer as a short
longValue()	returns the value of this Integer as a long

Example:-

class Test

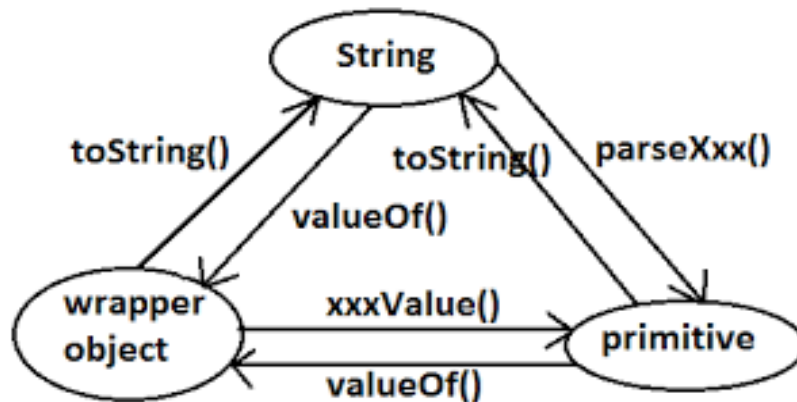
```
{    public static void main(String[] args)
    {        //valueOf() method to create Wrapper object
        Integer a1 = Integer.valueOf(10);
        System.out.println(a1);
        Integer a2 = Integer.valueOf("1000");
        System.out.println(a2);

        //xxxValue() [wrapper object into primitive value]
        int x1 = a1.intValue();
        byte x2 = a1.byteValue();
        double x3 = a1.doubleValue();
        System.out.println("int value="+x1);
        System.out.println("byte value="+x2);
        System.out.println("double value="+x3);
    }
}
```

3. toString():-

- toString() method present in Object class it returns class-name@hashcode.
- String,StringBuffer classes are overriding toString() method it returns content of the objects.
- All wrapper classes overriding toString() method to return content of the wrapper class objects.

METHOD	PURPOSE
toString(i)	<i>returns a new String object representing the integer i</i>



Example :-

class Test

```
{    public static void main(String[] args)
    {        Integer i1 = new Integer(100);
              System.out.println(i1);
              System.out.println(i1.toString());

              Integer i2 = new Integer("1000");
              System.out.println(i2);
              System.out.println(i2.toString());

              Integer i3 = new Integer("ten");//java.lang.NumberFormatException
              System.out.println(i3);
    }
```

- In above example for the integer constructor we are passing “1000” value in the form of String it is automatically converted into Integer format.
- In above example for the integer constructor we are passing “ten” in the form of String but this String is unable to convert into integer format it generate exception **java.lang.NumberFormatException**.

Example:-conversion of wrapper to String by using toString() method

```
class Test
{
    public static void main(String[] args)
    {
        Integer i1 = new Integer(100);
        Integer i2 = new Integer("1000");
        System.out.println(i1+i2); //1100
        //conversion [wrapper object - String]
        String str1 = i1.toString();
        String str2 = i2.toString();
        System.out.println(str1+str2); //1001000
    }
}
```

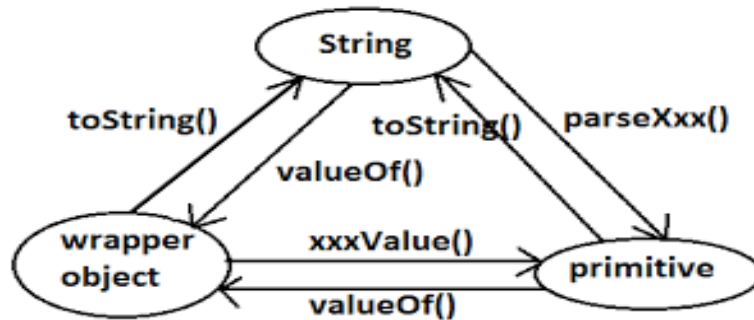
Example:-

- In java we are able to call **toString()** method only on reference type but not primitive type.
- If we are calling **toString()** method on primitive type then compiler generate error message.

```
class Test
{
    public static void main(String[] args)
    {
        Integer i1 = Integer.valueOf(100);
        System.out.println(i1);
        System.out.println(i1.toString());
        int a=100;
        System.out.println(a);
        //System.out.println(a.toString()); error:-int cannot be dereferenced
    }
}
```

4. parseXxx():- it is used to convert String into corresponding primitive value& it is a *static method* present in wrapper classes.

METHOD	PURPOSE
parseInt(s)	returns a signed decimal integer value equivalent to string s



Example :-

```

class Test
{
    public static void main(String[] args)
    {
        String str1="100";
        String str2="100";
        System.out.println(str1+str2);
        //parseXXX() conversion of String to primitive type
        int a1 = Integer.parseInt(str1);
        float a2 = Float.parseFloat(str2);
        System.out.println(a1+a2);
    }
}
  
```

EXAMPLES:

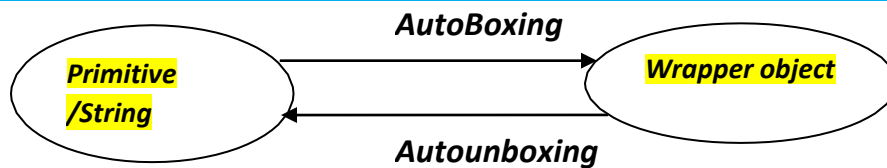
- 1) **primitive ----->Wrapper Object**
Integer i = Integer.valueOf(100);
- 2) **wrapper object ----> primitive**
byte b = i.byteValue();
- 3) **String value ----> primitive**
String str="100";
int a = Integer.parseInt(str);
- 4) **primitive value --> String Object**
int a=100;
int b=200;
String s1 = String.valueOf(a);
String s2 = String.valueOf(b);
System.out.println(s1+s2);//100200
- 5) **String value ---->Wrapper object**
Integer i =Integer.valueOf("1000");
- 6) **wrapper object --->String object**
Integer i = new Integer(1000);
String s = i.toString();

Autoboxing and Autounboxing:- (introduced in the 1.5 version)

- Up to 1.4 version to convert *primitive/String* into *Wrapper object* we are having two approaches
 - **Constructor approach**
 - **valueOf() method**

- Automatic conversion of primitive to wrapper object is called **autoboxing**.
- Automatic conversion of wrapper object to primitive is called **autounboxing**.

Automatic conversion of the primitive to wrapper and wrapper to the primitive:-



Example:-

```
class Test
{
    public static void main(String[] args)
    {
        //autoboxing [primitive - wrapper object]
        Integer i = 100;
        System.out.println(i);
        System.out.println(i.toString());

        //autounboxing [wrapper object - primitive]
        int a = new Integer(100);
        System.out.println(a);
    }
}
```

Object Oriented Programming (OOPS)

Agenda:

1. Data Hiding
2. Abstraction
3. Encapsulation(Data Hiding+ Abstraction)
4. Tightly Encapsulated Class

1. Data Hiding:

- Our internal data should not go out directly i.e., outside person can't access our internal data directly.
- By using **private** modifier we can implement data hiding.

Example:

```
class Account
{
    private double balance;
    .....;
    .....;
}
```

- After providing proper username and password only, we can access our Account information.
- The main advantage of data hiding is **security**.

Note: recommended modifier for data members is **private**.

2. Abstraction:

- Hide internal implementation and just highlight the set of services, is called **abstraction**.
- By using **abstract classes** and **interfaces** we can implement abstraction.

Example:

By using ATM GUI screen bank, people are highlighting the set of services what they are offering without highlighting internal implementation.

The main advantages of Abstraction are:

1. We can achieve security as we are not highlighting our internal implementation.(i.e., outside person doesn't aware our internal implementation.)

- Enhancement will become very easy because without effecting end-user , we can able to perform any type of changes in our internal system.
- It provides more **flexibility** to the end user to use system very easily.
- It improves **maintainability** of the application.
- It improves **modularity** of the application.
- It improves **easyness** to use our system.

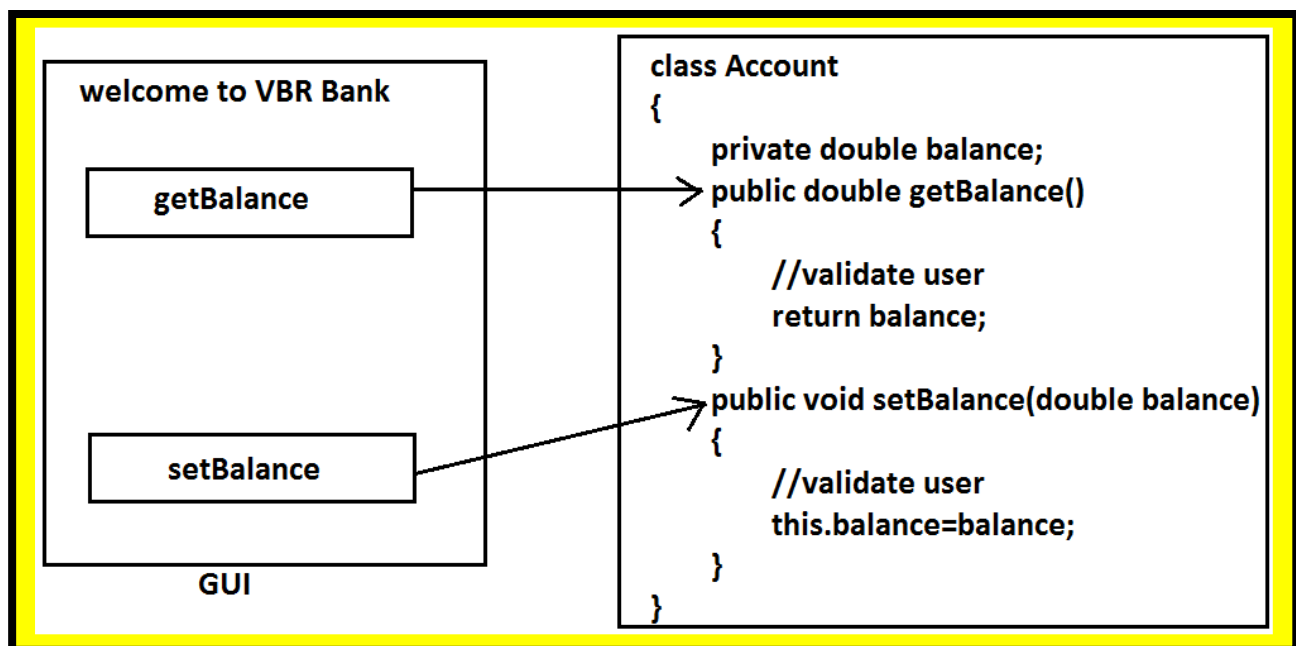
By using **interfaces (GUI screens)** we can implement abstraction.

3. Encapsulation:

- Binding of data and corresponding methods into a single unit is called Encapsulation .
- If any java class follows data hiding and abstraction such type of class is said to be encapsulated class.

Encapsulation=Datahiding+Abstraction

Example:



Every data member should be declared as private and for every member we have to maintain getter & Setter methods.

The main advantages of encapsulation are :

- We can achieve security.
- Enhancement will become very easy.
- It improves maintainability and modularity of the application.
- It provides flexibility to the user to use system very easily.

The main disadvantage of encapsulation is it increases length of the code and slows down execution.

4. Tightly encapsulated class:

A class is said to be tightly encapsulated if and only if every variable of that class declared as private whether the variable has getter and setter methods are not, and whether these methods declared as public or not, these checkings are not required to perform.

Example:

```
class Account
{
    private double balance;
    public double getBalance()
    {
        return balance;
    }
}
```

Which of the following classes are tightly encapsulated?

```
class A
{
    private int x=10; (valid)
}
class B extends A
{
    int y=20;(invalid)
}
class C extends A
{
    private int z=30; (valid)
}
```

Which of the following classes are tightly encapsulated?

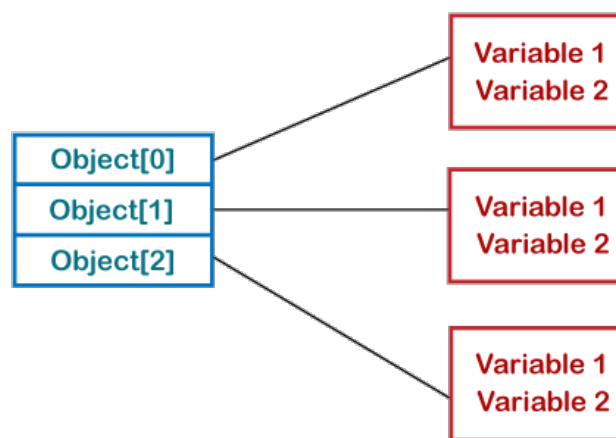
```
class A
{
    int x=10;    //not
}
class B extends A
{
    private int y=20; //not
}
class C extends B
{
    private int z=30; //not
}
```

Note: if the parent class is not tightly encapsulated, then no child class is tightly encapsulated.

Array of Objects in Java

- Java is an object-oriented programming language.
- Most of the work done with the help of **objects**.
- We know that an array is a collection of the same data type that dynamically creates objects and can have elements of primitive types.
- Java allows us to store objects in an array.
- In [Java](#), the class is also a user-defined data type. An array that contains **class type elements** are known as an **array of objects**.
- It stores the reference variable of the object.

Arrays of Objects



Creating an Array of Objects

Before creating an array of objects, we must create an instance of the class by using the **new** keyword. We can use any of the following statements to create an array of objects.

Syntax:

```
ClassName obj[] = new ClassName[array_length]; //declare and instantiate an array of objects
```

or

```
ClassName[] objArray;
```

or

```
ClassName objArray[];
```

Suppose, we have created a class named Employee. We want to keep records of 20 employees of a company having three departments. In this case, we will not create 20 separate variables. Instead of this, we will create an array of objects, as follows.

```
Employee department1[20];
```

```
Employee department2[20];
```

Employee department3[20];

The above statements create an array of objects with 20 elements.

Let's create an array of objects in a [Java program](#)

In the following program, we have created a class named Product and initialized an array of objects using the constructor. We have created a constructor of the class Product that contains product id and product name. In the main function, we have created individual objects of the class Product. After that, we have passed initial values to each of the objects using the constructor.

PROGRAM:

```
public class ArrayOfObjects
{
    public static void main(String args[])
    {
        //create an array of product object
        Product[] obj = new Product[5] ;
        //create & initialize actual product objects using constructor
        obj[0] = new Product(23907,"Dell Laptop");
        obj[1] = new Product(91240,"HP 630");
        obj[2] = new Product(29823,"LG OLED TV");
        obj[3] = new Product(11908,"MI Note Pro Max 9");
        obj[4] = new Product(43590,"Kingston USB");
        //display the product object data
        System.out.println("Product Object 1:");
        obj[0].display();
        System.out.println("Product Object 2:");
        obj[1].display();
        System.out.println("Product Object 3:");
        obj[2].display();
        System.out.println("Product Object 4:");
        obj[3].display();
        System.out.println("Product Object 5:");
        obj[4].display();
    }
}

//Product class with product Id and product name as attributes
class Product
{
    int pro_Id;
    String pro_name;
    //Product class constructor
    Product(int pid, String n)
    {
        pro_Id = pid;
        pro_name = n;
    }
    public void display()
    {
        System.out.print("Product Id = "+pro_Id + " " + " Product Name = "+pro_name);
        System.out.println();
    }
}
```


Output:

```
student@student-HP-245-G6-Notebook-PC:~$ javac ArrayOfObjects.java
student@student-HP-245-G6-Notebook-PC:~$ java ArrayOfObjects
Product Object 1:
Product Id = 23907    Product Name = Dell Laptop
Product Object 2:
Product Id = 91240    Product Name = HP 630
Product Object 3:
Product Id = 29823    Product Name = LG OLED TV
Product Object 4:
Product Id = 11908    Product Name = MI Note Pro Max 9
Product Object 5:
Product Id = 43590    Product Name = Kingston USB
```

//Java Program to create a user defined LinkedList class and store list of books and display

```
import java.util.*;
class Book
{
    String bname;
    String author;
    int price;
    Book(String bname, String author, int price)
    {
        this.bname = bname;
        this.author = author;
        this.price = price;
    }
    public String toString()
    {
        String s = "Name = " + this.bname + "\tAuthor = " + this.author + "\tPrice = " + this.price;
        return s;
    }
}
class BookListDemo
{
    public static void main(String[] args)
    {
        LinkedList ll = new LinkedList();
        ll.add(new Book("Java", "Herbert", 200));
        ll.add(new Book("C", "Dennis", 150));
        ll.add(new Book("C++", "Schildt", 350));
        ll.add(new Book("Operating Systems", "Galvin", 500));
        ListIterator litr;
        System.out.println("***** Book Details *****");
        for( litr = ll.listIterator(); litr.hasNext(); )
            System.out.println(litr.next() + "\t");
    }
}
```

Output:

```
student@student-HP-245-G6-Notebook-PC:~$ javac BookListDemo.java
Note: BookListDemo.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
student@student-HP-245-G6-Notebook-PC:~$ java BookListDemo
***** Book Details *****
Name = Java      Author = Herbert      Price = 200
Name = C         Author = Dennis Price = 150
Name = C++       Author = Schildt      Price = 350
Name = Operating Systems      Author = Galvin Price = 500
```