UNIT- 2

Classes, Objects and Inheritance

Syllabus:

Classes and Objects- classes, Creating Objects, Methods, constructors, overloading methods and constructors, garbage collection, static keyword, this keyword, parameter passing, recursion, Arrays, Strings, Command line arguments.
Inheritance: Types of Inheritance, Deriving classes using extends keyword, concept of overriding, super keyword, final keyword.

Class:

- A class is a group of objects that has common properties. It is a template or blueprint from which objects are created.
- Simple classes may contain only code or only data, most real-world classes contain both.
- A class is declared by use of the **class** keyword.
- The general form of a class is

```
class  classname
{
        type  instance-variable1;
        type  instance-variable2;
        // …….
        type  instance-variableN;
        type  methodname1(parameter-list)
        {
              // body of method
        }
        type  methodname2(parameter-list)
         {
              // body of method
        }
        // ...
        type  methodnameN(parameter-list)
        {
              // body of method
        }
}
```

- The data, or variables, defined within a **class** are called ***instance variables****.*
- The code is contained within *methods.* Collectively, the methods and variables defined within a class are called ***members*** of the class.
- Simple Class like as bellow

```
class Box
{
        double width;
        double height;
        double depth;
}
```

- In the above, a class defines a new type of data. In this case, the new data type is called **Box**. You will use this name to declare objects of type **Box**.
- a **class** declaration only creates a template; it does not create an actual object.
- For creating a **Box** object we use,

  Box mybox = new Box(); // create a Box object called mybox
- Here **mybox** will be an instance of **Box**.
- To acces variables, methods inside a class we have to use .(dot) operator.
- The dot operator links the name of the object with the name of an instance variable.
- to assign the **width** variable of **mybox** the value 100, you would use the following statement:

  mybox.width=100;
- **Example:**

```
class Box
 {
        double width;
        double height;
        double depth;
}
class BoxDemo
{
        public static void main(String args[])
        {
                Box mybox = new Box();
                double vol;
                mybox.width = 10;
                mybox.height = 20;
                mybox.depth = 15;
                vol = mybox.width * mybox.height * mybox.depth;
                System.out.println("Volume is " + vol);
        }
}
```

- The above program must save with **BoxDemo.java** (because BoxDemo contains main())
- After compiling the program two class files are created one is **Box.class** and other one is **BoxDemo.class**
- At the time of running BoxDemo.class file will be loaded by JVM
- **For compilation:** javac BoxDemo.java
- **For Running:** java BoxDemo
- **Output:**

  Volume is 3000.0

- **We can create multiple objects for the same class**
- Example

```
class Box
 {
        double width;
        double height;
        double depth;
}
class BoxDemo
{
        public static void main(String args[])
        {
                Box  mybox1 = new Box();
                Box  mybox1 = new Box();
                double vol1,vol2;
                mybox1.width = 10;
                mybox1.height = 20;
                mybox1.depth = 15;
                vol1 = mybox1.width * mybox1.height * mybox1.depth;
                mybox1.width = 10;
                mybox1.height = 15;
                mybox1.depth = 5;
                vol2= mybox2.width * mybox2.height * mybox2.depth;
                System.out.println("Volume of mybox1  is " + vol1);
                System.out.println("Volume of mybox2  is " + vol2);

        }
}
```

- Output

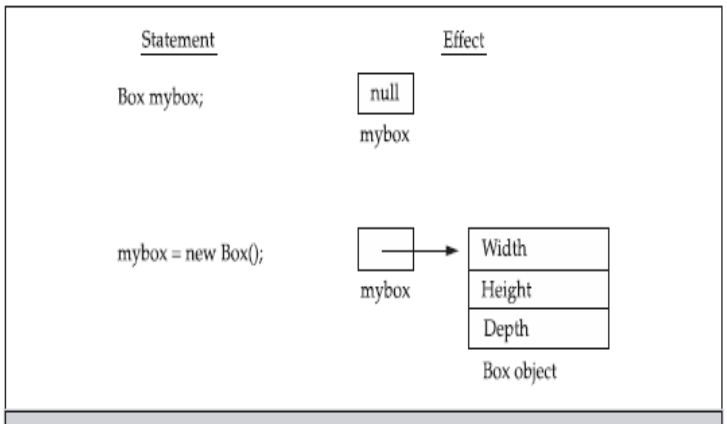
```
Volume of mybox1  is 3000.0
Volume of mybox2  is 750.0
```

Objects:

- Class is a template or blueprint from which objects are created. So object is the instance(result) of a class.
- for creating objects we have to use following code.

Box mybox;
// declare reference to object

mybox = new Box();
// allocate a Box object
Here new keyword is allocating memory of Box object.

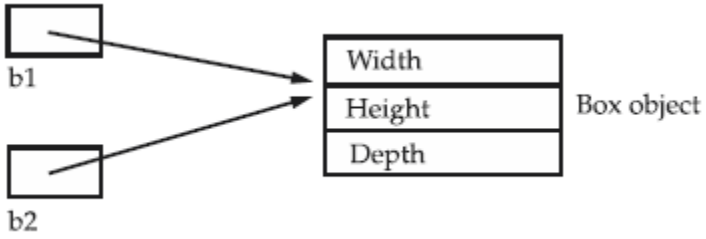


Assigning Object Reference Variables:

- For example see the following code

```
Box b1 = new Box();
Box b2 = b1;
```

- **In the above code b1** and **b2** will both refer to the *same* object.
- The assignment of **b1** to **b2** did not allocate any memory or copy any part of the original object.
- It simply makes **b2** refer to the same object as does **b1**.
- Thus, any changes made to the object through **b2** will affect the object to which **b1** is referring, since they are the same object.



******----------*******

Methods:

- In java, a method is like function i.e. used to expose behavior of an object.
- The advantages of methods are code reusability and code optimization
- This is the general form of a method:

```
type name(parameter-list)
{
        // body of method
}
```

- Here, *type* specifies the type of data returned by the method. This can be any valid type, including class types that you create.
- If the method does not return a value, its return type must be **void**.
- The name of the method is specified by *name.* This can be any legal identifier other than those already used by other items within the current scope.
- The *parameter-list* is a sequence of type and identifier pairs separated by commas.
- Parameters are essentially variables that receive the value of the *arguments* passed to the method when it is called.
- If the method has no parameters, then the parameter list will be empty.
- Methods that have a return type other than **void** return a value to the calling routine using the following form of the **return** statement:

```
return value;
```

  Here, *value* is the value returned.

- **Example program for usage of method:**

```
class Box
 {
        double width;
        double height;
        double depth;
        void volume() // method
        {
                double vol=width*height*depth;
                System.out.println("Volume is " + vol);
        }
}
class BoxDemo
{
        public static void main(String args[])
        {
                Box mybox = new Box();
                mybox.width = 10;
                mybox.height = 20;
                mybox.depth = 15;
                mybox.volume();
        }
}
```

Output: Volume is 3000.0

- **Example program for return a value from method :**

```
class Box
 {
        double width;
        double height;
        double depth;
        double volume() // method return type is double
        {
                return width*height*depth;
        }
}
class BoxDemo
{
        public static void main(String args[])
        {
                Box mybox = new Box();
                mybox.width = 10;
                mybox.height = 20;
                mybox.depth = 15;
                double vol= mybox.volume();
                System.out.printlln("volume is" + vol);
        }
}
```

Output: Volume is 3000.0

- **Example program for method with parameters:**

```
class Values
 {
        void add(int a,int b)
        {
                int c=a+b;
                System.out.println("addition of a, b  is "+c);
        }
 }
class Addition
{
        public static void main(String args[])
        {
                Addition ad=new Addition();
                ad.add(10,20);
        }
}
```

Output: addition of a, b is 30

- **<u>Returning values from a method:</u>**

```
class Values
 {
        void add(int a,int b)
        {
                c=a+b;
                return c;
        }
 }
class Addition
{
        public static void main(String args[])
        {
                int res;
                Addition ad=new Addition();
                res=ad.add(10,20);
                System.out.println("addition of a, b  is "+res);
        }
}
```

Output: addition of a, b is 30

Constructors:

- A *constructor* initializes an object immediately upon creation. It has the same name as the class in which it resides and is syntactically similar to a method.
- Once defined, the constructor is automatically called immediately after the object is created, before the **new** operator completes.
- Constructors do not any return type, not even **void.**
- This is because the implicit return type of a class' constructor is the class type itself.
- Example:

```
class Addition
{
        Addition()
        {
                System.out.println("This is Default constructor");
        }
}
class ConstructorDemo
{
        public static void main(String args[])
        {
                Addition ad=new Addition();
        }
}
```

Output: This is Default constructor

- Now we can understand why the parentheses are needed after the class name. What is actually happening is that the constructor for the class is being called. Thus, in the line

```
Addition ad=new Addition();
```

new Addition( ) is calling the **Addition( )** constructor.

- **When we do not explicitly define a constructor for a class, then Java creates a default constructor for the class.**

Parameterized Constructors:

- We can pass parameters to the constructor

```
class Addition
{
        Addition(int a,int b)
        {
                int c;
                c=a+b;
                System.out.println("Tha addition of a, b is ");
                System.out.println(c);
        }
}
```

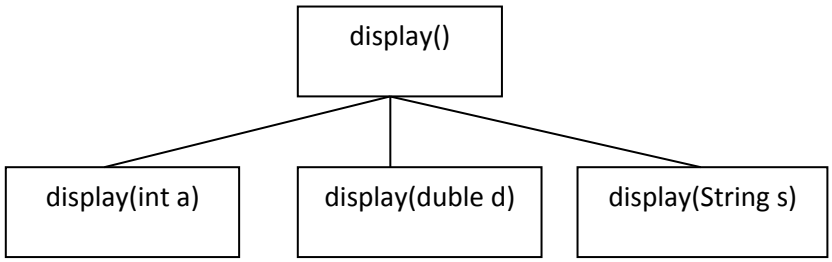
```
class ParameterConstructor
{
        public static void main(String args[])
        {
                Addition ad=new Addition(10,20);
        }
}
```

Output: Tha addition of a, b is 30

******--------*******

Method Overloading:

- In Java it is possible to define two or more methods within the same class that share the same name, as long as their parameter declarations are different.
- These methods are said to be *overloaded,* and the process is referred to as *method overloading.*
- Method overloading is one of the ways that Java implements polymorphism.
- **Method overloading** is also called as " **Static binding** or **Compile time polymorphism** or **Early binding**".



- **Example for method overloading**

```
class Overload
 {
        void dispaly()
        {
                System.out.println("No parameters");
        }
        void display(int a)
        {
                System.out.println("integer a: " + a);
        }
        void display(double d)
        {
                System.out.println("double d: " + d);
        }
        void display(String s)
        {
                System.out.println("String s:" + s);
        }
}
```

```
class OverloadDemo
{
        public static void main(String args[])
        {
                OverloadDemo ob = new OverloadDemo();
                ob.display();
                ob. display(10);
                ob. display(10.325);
                ob. display("hello");
        }
}
```

Output:

```
No parameters
integer a: 10
double d: 10.325
String s: hello
```

Constructor overloading:

- In addition to overloading normal methods, you can also overload constructor methods.
- In Java it is possible to define two or more constructors within the same class that share the same name, as long as their parameter declarations are different.
- Example

```
class Display
{
        Dispaly()
        {
                System.out.println("No parameters");
        }
        Dispaly(int a)
        {
                System.out.println("integer a: " + a);
        }
        Dispaly(double d)
        {
                System.out.println("double d: " + d);
        }
        Display(String s)
        {
                System.out.println("String s:" + s);
        }
}
class OverloadDemo
{
        public static void main(String args[])
        {
                Display ds=new Display();
                Display ds1=new Display(10);
                Display ds2=new Display(10.325);
                Display ds3=new Display("hello");
        }
}
```

Output:

```
No parameters
integer a: 10
double d: 10.325
String s: hello
```

Java Garbage Collection

- In java, garbage means unreferenced objects.
- Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.
- To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

How can an object be unreferenced?

There are many ways:

- By nulling the reference
- By assigning a reference to another
- By annonymous object etc.

1) By nulling a reference:

```
Employee e=new Employee();
e=null;
```

2) By assigning a reference to another:

```
Employee e1=new Employee();
Employee e2=new Employee();
e1=e2;//now the first object referred by e1 is available for garbage collection
```

3) By annonymous object:

```
new Employee();
```

static keyword:

- It is possible to create a member that can be used by itself, without reference to a specific instance.
- To create such a member, precede its declaration with the keyword **static**.
- When a member is declared **static**, it can be accessed before any objects of its class are created, and without reference to any object.
- We can declare both methods and variables to be **static**.
- The most common example of a **static** member is **main( )**. **main( )** is declared as **static** because it must be called before any objects exist.
- Instance variables declared as **static** are, essentially, global variables.
- Methods declared as **static** have several restrictions:
    - They can only call other **static** methods.
    - They must only access **static** data.
    - They cannot refer to **this** or **super** in any way.
- We can declare a **static** block which gets executed exactly once, when the class is first loaded.
- Example

```
class UseStatic
{
        static int a = 3;
        static int b;
        static void display(String s)
        {
                System.out.println("Static method invoked String s= "+s);
        }
        static
        {
                System.out.println("Static block initialized.");
                b = a * 4;
        }
}
class  StaticDemo
{
        public static void main(String args[])
        {
                System.out.println("static variable a= "+ UseStatic.a);
                System.out.println("static variable b= "+UseStatic.b);
                UseStatic. Display("JAVA");
        }
}
```

Ouput:

```
Static block initialized.
static variable a= 3
static variable b= 12
static method invoked String s= JAVA
```

this keyword:

- In java, this is a **reference variable** that refers to the current object.

Usage of this keyword

- Here some usages of this keyword.
    1. To refer current class instance variable.
    2. To Invoke current class constructor.

To refer current class instance variable:

- If there is ambiguity between the instance variable and parameter, this keyword resolves the problem of ambiguity.

```
class Student
{
        int id;
        String name;
        Student10(int id,String name)
        {
                this.id = id;
                this.name = name;
        }
        void display()
        {
                System.out.println(id+" "+name);
        }
        public static void main(String args[])
        {
                Student  s1 = new Student10(111,"Karan");
                Student  s2 = new Student10(321,"Aryan");
                s1.display();
                s2.display();
        }
}
```

Output: 111 Karan
321 Aryan

To invoke current class constructor

- The this() constructor call can be used to invoke the current class constructor (constructor chaining).
- This approach is better if you have many constructors in the class and want to reuse that constructor.
- Example:

```
class Student1
{
        int id;
        String name;
        Student1()
        {
                System.out.println("default constructor is invoked");
        }

        Student1(int id,String name)
        {
                this ();//it is used to invoked current class constructor.
                this.id = id;
```

```
                this.name = name;
        }
        void display()
        {
                System.out.println(id+" "+name);
        }

        public static void main(String args[])
        {
                Student1 e1 = new Student13(111,"karan");
                Student1 e2 = new Student13(222,"Aryan");
                e1.display();
                e2.display();
        }
}
```

Output:

```
default constructor is invoked
default constructor is invoked
111 Karan
222 Aryan
          **********----------**********
```

parameter passing techniques:

- call by value (or) pass by value
- call by reference (or) pass by reference

Call by value:

- In call by value we are passing only value to the method.
- If we perform any changes inside method that changes will not reflected to main method.
- Example:

```
class CallByVal
{

        public static void main(String args[])
        {
                int x=20;
                System.out.println("Before method calling value= "+x);
                display(x);
                System.out.println("After method calling value= "+x);
        }

         public static void display(int y)
        {
                y=y+1;
                System.out.println("Inside method value= "+y);
        }

}
```

Output:

```
Before method calling value= 20
Inside method value= 21
After method calling value= 20
```

Call by reference:

- In call by reference we are passing reference i.e an object to the method.
- In this case, If we perform any changes inside method that changes will not reflected to main method.
- Example:

```
class CallByRef
{
	int x;
	public static void main(String args[])
	{
		CalByRef c=new CalByRef();
		c.x=20;
		System.out.println("Before method calling value= "+c.x);
		display(c);
		System.out.println("After method calling value= "+c.x);
	 }
	public static void display(CalByRef m)
	{
		m.x=m.x+1;
		System.out.println("Inside method value= "+m.x);
	}
}
```

Output:

```
Before method calling value= 20
Inside method value= 21
After method calling value= 21
```

******_____******

Recursion:

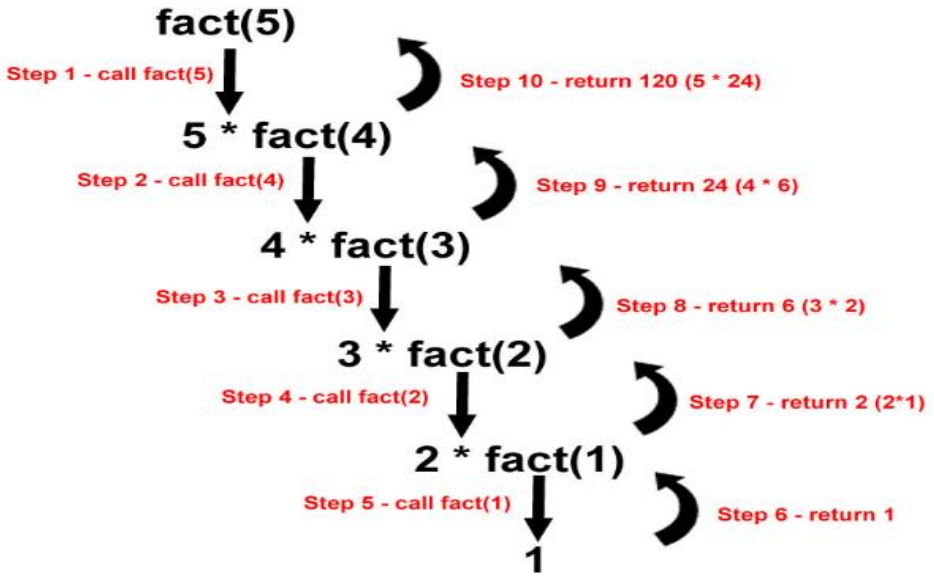
- Recursion is the process of method calling itself.
- A method calling itself is called as recursive method.
- The classic example of recursion is the computation of the factorial of a number.

```
class Factorial
{
        int fact(int n)
        {
                if(n==1)
                        return 1;
                else
                        n*fact(n-1) ;
        }
}
class Recursion
{
        public static void main(String args[])
        {
                Factorial f = new Factorial();
                System.out.println("Factorial of 3 is " + f.fact(3));
                System.out.println("Factorial of 4 is " + f.fact(4));
                System.out.println("Factorial of 5 is " + f.fact(5));
        }
}
```

Output:

```
Factorial of 3 is 6
Factorial of 4 is 24
Factorial of 5 is 120
```

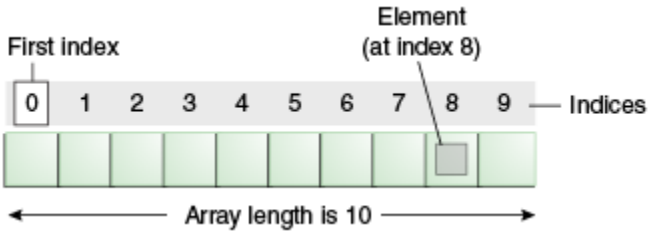
- Recursion process of fact(5) is given bellow.



******_____******

Arrays:

- An *array* is a group of similar data elements that are referred to by a common name.
- Arrays of any type can be created and may have one or more dimensions.
- A specific element in an array is accessed by its index.
- It is a data structure where we store similar elements. We can store only fixed set of elements in a java array.



- The advantage of an array is code optimization and random access.

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array

- Syntax for declare an array

```
dataType[] arr; (or)
dataType []arr; (or)
dataType arr[];
```

- Instantiation of an Array

```
arr=new datatype[size];
```

- Example

```
class Testarray
{
    public static void main(String args[])
    {

        int a[]=new int[3];
        a[0]=10;
        a[1]=20;
        a[2]=70;
        for(int i=0;i<a.length;i++)
            System.out.println(a[i]);
    }
}
```

Output: 10
20
70

- **Declaration, Instantiation and Initialization of an Array:**

datatype arr[]={val1,val2,val3,…};//declaration, instantiation and initialization

Example:

```
class Testarray1
{
    public static void main(String args[])
    {
        int a[]={33,3,4,5};
        for(int i=0;i<a.length;i++)
            System.out.print(a[i]+" ");
    }
}
```

Output:33 3 4 5

Multidimensional array

- data is stored in row and column based index (also known as matrix form).
- Syntax to Declare Multidimensional Array

```
dataType[][]  arrayRefVar; (or)
dataType  [][]arrayRefVar; (or)
dataType  arrayRefVar[][]; (or)
dataType  []arrayRefVar[];
```

- To instantiate Multidimensional Array

```
int[][] arr=new int[3][3];//3 row and 3 column
```

- To initialize Multidimensional Array

```
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
…….arr[2][2]=9;
```

- Example:

```
class Testarray3
{
    public static void main(String args[])
    {
        int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
        for(int i=0;i<3;i++)
        {
            for(int j=0;j<3;j++)
            {
                System.out.print(arr[i][j]+" ");
            }
            System.out.println();
        }
    }
}
```

Output:1 2 3
2 4 5
4 4 5

******------******

Strings:

- Generally, string is a sequence of characters. But in java, string is an object that represents a sequence of characters.
- The java.lang.String class is used to create string object.
- Strings are immutable in nature, i.e once an object created that object doesn't allow any changes to it.
- String is basically an object that represents sequence of char values. An array of characters works same as java string. For example:

```
char[] ch={'j','a','v','a'};
String s=new String(ch);
```

  is same as:

```
String s="javA";
```

- There are two ways to create String object:
  1. By string literal
  2. By new keyword

String Literal

- Java String literal is created by using double quotes. For Example:

```
String s="welcome";
```

- The advantage of string literal usage is memory efficiency.

By new keyword

```
String s=new String("Welcome");
```

Example:

```
class StringExample
{
	public static void main(String args[])
	{
		String s1="java";//creating string by java string literal
		char ch[]={'s','t','r','i','n','g','s'};
		String s2=new String(ch);//converting char array to string
		String s3=new String("example");//creating java string by new keyword
		System.out.println(s1);
		System.out.println(s2);
		System.out.println(s3);
	}
}
```

Output

```
java
strings
example
```

String handling methods:

S.N	Method name	General form	Example
1	length()	int length( )	String s = new String("hello"); System.out.println(s.length());
2	charAt( )	char charAt(int *where*)	char ch; ch = "abc".charAt(1);

3	getChars( )	void getChars
		(int *sourceStart*, int *sourceEnd*,
		char *target*[ ], int *targetStart*)
			String s = "This is a demo of the
			getChars method.";
			int start = 10;
			int end = 14;
			char buf[] = new char[10];
			s.getChars(start, end, buf, 0);
			System.out.println(buf);

4	equals( )	boolean equals(Object *str*)	
5	equalsIgnoreCase( )	boolean equalsIgnoreCase (String *str*)	String s1 = "Hello"; String s2 = "hello"; System.out.println("s1 and s2 are equal " +s1.equals(s2)); System.out.println("s1 and s2 are equal" +s1.equalsIgnoreCase(s4));
6	substring( )	String substring(int *startIndex*) String substring(int *startIndex*, int *endIndex*)	String s1="hello java"; System.out.println(substring(1,4));
7	concat( )	String concat(String *str*)	String s1 = "one"; String s2 = s1.concat("two");
8	replace( )	String replace(char *original*, char *replacement*)	String s = "Hello".replace('l', 'w');
9	trim( )	String trim( )	String s = " Hello World "; System.out.println(s.trim());
10	toLowerCase( ) toUpperCase( )	String toLowerCase( ) String toUpperCase( )	String s1 = " HELLO"; String s2="hello"; System.out.println(s1.toLowerCase( )); System.out.println(s2.toUpperCase( ));

Example program String handling methods:

```
class StringDemo1
{
  public static void main(String args[])
  {
        String s1="hello qiscet";
        String s2="HELLO QISCET";
        System.out.println("length of string s1= "+s1.length());
        System.out.println("index of 'o' is: "+s1.indexOf('o'));
        System.out.println("String s1 in Uppercase: "+s1.toUpperCase());
        System.out.println("String s1 in Uppercase: "+s2.toLowerCase());
        System.out.println("s1 equals to s2?: "+s1.equals(s2));
        System.out.println("s1 is equal to s2 after ignoring case:"
                                                +s1.equalsIgnoreCase(s2));
        int result=s1.compareTo(s2);
        if(result==0)
```

```
            System.out.println("s1 is equals to s2");
        else if(result>0)
            System.out.println("s1 is greater than s2");
        else
            System.out.println("s1 is smaller than s2");
        System.out.println("character at index of 6 in s1: "+s1.charAt(6));
        String s3=s1.substring(2,8);
        System.out.println("substring s3 in s1 is: "+s3);
        System.out.println("repalcing 'e' with 'a' in s1: "+s1.replace('e','a'));
        String s4="   qiscet   ";
        System.out.println("string s4: "+s4);
        System.out.println("string s4 after trim: "+s4.trim());
    }
}
```

Output:

```
length of string s1= 12
index of 'o' is: 4
String s1 in Uppercase: HELLO QISCET
String s1 in Uppercase: hello qiscet
s1 equals to s2?: false
s1 is equal to s2 after ignoring case: true
s1 is greater than s2
character at index of 6 in s1: q
substring s3 in s1 is: llo qi
repalcing 'e' with 'a' in s1: hallo qiscat
string s4:    qiscet
string s4 after trim: qiscet
```

*******_______*******

StringBuffer:

- StringBuffer is a peer class of String that provides much of the functionality of strings.
- String represents fixed-length, immutable character sequences. In contrast, StringBuffer represents growable and writeable character sequences.
- General form of StringBuffer

StringBuffer sb=new StringBuffer(“Hello World”);

- StringBuffer class exists in “java.lang” package.
- StringBuffer is immutable that means, any changes performed to the StringBuffer object that changes will reflect to object.

- **StringBuffer Methods:**

S.N	Method name	General form	Example
1	length( )	int length( )	StringBuffer sb = newStringBuffer("Hello"); System.out.println("buffer = " + sb); System.out.println("length="+sb.length()); System.out.println("capacity="+ sb.capacity());
2	capacity( )	int capacity( )	
3	ensure Capacity( )	void ensureCapacity (int *capacity*)	sb.ensureCapacity(30));

4	setLength( )	void setLength(int *len*)	StringBuffer sb = new StringBuffer("Hello");
			System.out.println("buffer before = " + sb);
			System.out.println("charAt(1) before = " + sb.charAt(1));
			sb.setCharAt(1, 'i');
			sb.setLength(2);
			System.out.println("buffer after = " + sb);
			System.out.println("charAt(1) after = " + sb.charAt(1));

5	charAt( )	char charAt(int *where*)	
6	setCharAt( )	void setCharAt(int *where,* char *ch*)	

7	getChars( )	void getChars(int *sourceStart,* int *sourceEnd,* char *target*[ ], int *targetStart*)	StringBuffer sb=new StringBuffer("This is a demo of the getChars method.");
			int start = 10;
			int end = 14;
			char buf[] = new char[10];
			sb.getChars(start, end, buf, 0);
			System.out.println(buf);

8	append()	StringBuffer append(String *str*) StringBuffer append(int *num*)	StringBuffer sb = new StringBuffer(40); s = sb.append("is a number"); System.out.println(s);
9	insert( )	StringBuffer insert(int *index*, String *str*) StringBuffer insert(int *index*, char *ch*)	StringBuffer sb = new StringBuffer("I Java!"); sb.insert(2, "like "); System.out.println(sb);
10	reverse( )	StringBuffer reverse( )	StringBuffer s = new StringBuffer("abcdef"); System.out.println(s); s.reverse(); System.out.println(s);

11	delete( ) and deleteCharAt( )	StringBuffer delete(int *startIndex,* int *endIndex*)
		StringBuffer deleteCharAt(int *loc*)
			StringBuffer sb = new StringBuffer("This is a test.");
			sb.delete(4, 7);
			System.out.println("After delete: " + sb);
			sb.deleteCharAt(0);
			System.out.println("After deleteCharAt: " + sb);

12	replace( )	StringBuffer replace(int *startIndex,* int *endIndex,* String *str*)	StringBuffer sb = new StringBuffer("This is a test."); sb.replace(5, 7, "was"); System.out.println("After replace: " + sb);
13	substring( )	String substring(int *startIndex*) String substring(int *startIndex,* int *endIndex*)	StringBuffer sb = new StringBuffer(”hello java”); String s= substring(1,4); System.out.println(s);

******_____*******

Command line arguments:

- A command-line argument is the information that directly follows the program's name on the command line when it is executed.
- To access the command-line arguments inside a Java program is quite easy—they are stored as strings in the **String** array passed to **main( )**.
- Forexample,
-

```
class CommandLine
{
        public static void main(String args[])
         {
                for(int i=0; i<args.length; i++)
                        System.out.println("args[" + i + "]: " +args[i]);
        }
}
```

Compilation and Execution:

```
javac  CommandLine.java
java CommandLine this is a test 100 -1
```

Output:

```
args[0]: this
args[1]: is
args[2]: a
args[3]: test
args[4]: 100
args[5]: -1
```

*******----------********