Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value. To declare an array, define the variable type with **square brackets**.

Example

```
int[] a;          //recommended to use because name is clearly separated from the type
int []b;
int c[];
```

At the time of declaration, we can't specify the size otherwise we will get compile time error.

Example

```
int[] a;          //valid
int[5] a;         //invalid
```

Two-dimensional array declaration

Example

```
int[][] a;
int [][]a;
int a[][];
int[] []a;
int[] a[];
int []a[];
```

Three-dimensional array declaration

Example

```
int[][][] a;
int [][][]a;
int a[][][];
int[] [][]a;
int[] a[][];
int[] []a[];
int[][] []a;
int[][] a[];
int []a[][];
int [][]a[];
```

Which of the following declarations are valid?

```
1) int[] al, bl;          a1 and b1 one dimension
2) int[] a2[],b2;         a2 is two-dimensional, b2 one dimensional
3) int[] []a3,b3;         both two dimensional
4) int[] a, []b;          //C.E:<identifier> expected(invalid)
```

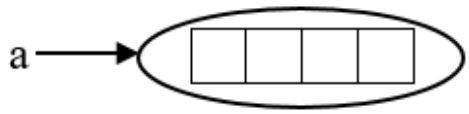
If we want to specify dimension before the variable that facility is applicable only for first variable in a declaration. If we try to apply for remaining variable, we will get compile time error.

Array construction

Every array in java is an object and new operator is used for creating it.

Example

```
int[] a=new int[3];
```



For every array type there is corresponding class. These classes are part of java language but not available at programmer level.

```
System.out.println(a.getClass().getName());   //it will print corresponding class of array.
```

Array	Class
int[]	[I
int[][]	[[I
double[]	[D
boolean[]	[Z
byte[]	[B
short[]	[S

Rules of array creation

- It is compulsory to specify the size of array at the time of creation otherwise we will get compile time error.
- Array with size zero is valid in java.

```
Class size
{
        public static void main (String [] args)
        {
                System.out.println(args.length)                 //output is Zero
        }
}
```

- Array size with -ve int value is invalid. If we take negative value then we will get runtime exception saying

NegativeArraySizeException.

- Only byte, short, char, int data types are allowed to specify array size else we will get compile time error.
- The maximum allowed array size in java is maximum value of int size

Example

```
int[] a=new int[3];
int[] a=new int[];              //C.E:array dimension missing
int[] a=new int[0];             //valid
int[] a=new int[-3];            //R.E:NegativeArraySizeException
int[] a=new int['a'];           //(valid)
byte b=10;
int[] a=new int[b];             //(valid)
short s=20;
int[] a=new int[s];             //(valid)
int[] a=new int[10L];           //C.E:possible loss of precision (invalid)
int[] a=new int[l0.5];          //C.E:possible loss of precision (invalid)
int[] al=new int[2147483647];         (valid)
```

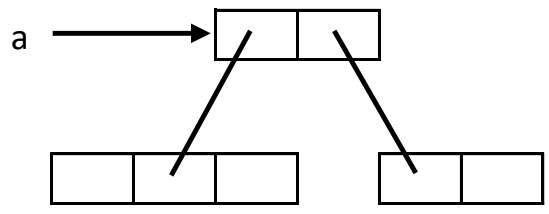
Two-dimensional array creation

In java multidimensional arrays are implemented as array of arrays approach but not matrix form.
The main advantage of this approach is to improve memory utilization.

Example

```
int[][] a=new int[2] [];
a[0]=new int[3];
a[l]=new int[2];
```

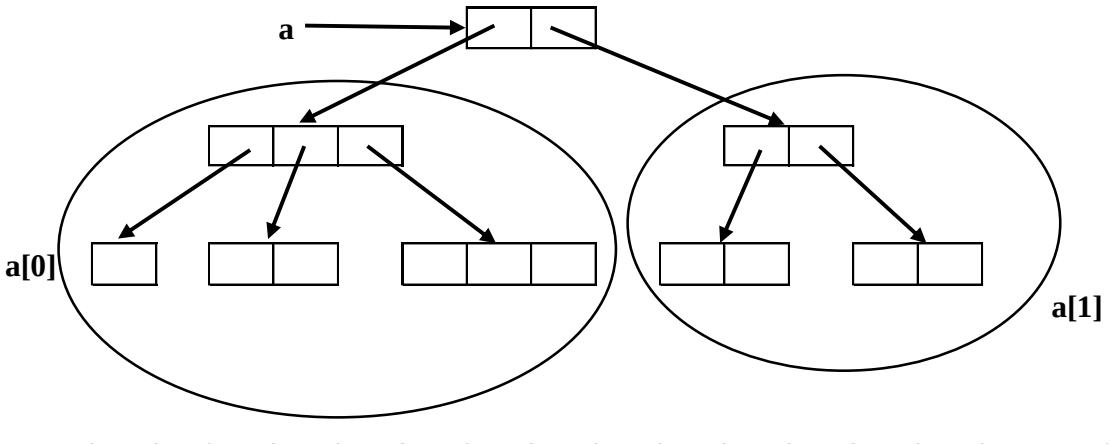
Memory Representation



Example

```
int[][][] a=new int[2][][];
a[0]=new int[3][];
a[0][0]=new int[l];
a[0][l]=new int[2];
a[0][2]=new int[3];
a[l]=new int[2][2];
```

Memory Representation



Array initialization

Every element is initialized with default value automatically.

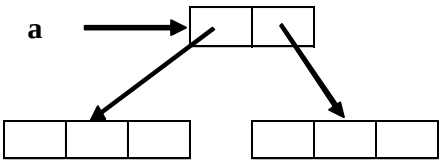
Example

```
int[] a=new int[3];
System.out.println(a);          //[1@3e25a5
System.out.println(a[0]);       //0
```

Whenever we are trying to print any object reference internally toString() method will be executed which is implemented by default to return the following.

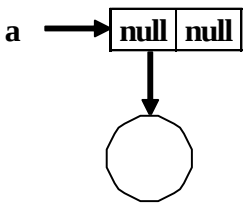
classname@hexadecimalstringrepresentationofhashcode.

Example:




```
int [][] a=new int[2][3]
System.out.println (a);         //[[1@3e25a5
System.out.println(a [0]);      //[1@19821f
System.out.println(a [0] [0]);  //0
```

Example




```
int[][] a=new int[2][];
System.out.println(a);          //[[I@3e25a5
System.out.println(a[0]);       //null
System.out.println(a[0][0]);    //R.E:NullPointerException
```

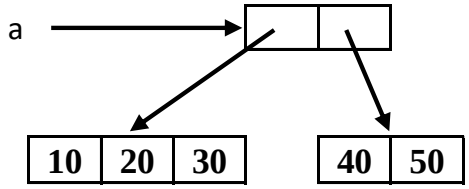
Declaration construction and initialization of an array in a single line

Arrays can also be initialized with customized values like:

Example

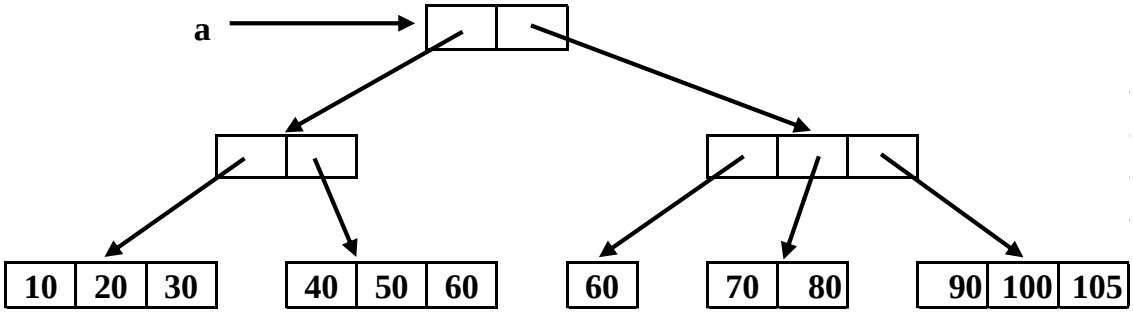
```
int[] a=new int[4];
a[0]=l0;
a[1]=20;
a[2]=30;
a[3]=40;  or int[] a={l0,20,30};

int[][] a={{l0,20,30},{40,50}};
```



Int [][][]={{{10,20,30},{40,50,60}},{{60},{70,80},{90,100,105}}}

Memory Representation




```
System.out.println(a[0][l][l]);          //50(valid)
System.out.println( a[l] [ 0] [2]) ;     //R. E:ArraylndexOutOfBoundsException: 2(invalid)
System.out.println(a[1][2][1]);          //100(valid)
System.out.println{ a[ 1] [2] [2]);      //l05(valid)
System.out.println{ a[2] [ 1] [ 0]) ;    //R. E:ArraylndexOutOfBoundsException: 2(invalid)
System.out.println{a[l][l][l]);          //80(valid)
```

length Vs length()

length

It is the final variable applicable only for arrays.
It represents the size of the array.

Example:

```
int[] x=new int[3];
System.out.println(x.length());          //C.E: cannot find symbol
System.out.println(x.length);            //3
```

length() method

It is a final method applicable for String objects.
It returns the no of characters present in the String.

Example

```
String s="computer";
System.out.println(s.length);            //C.E:cannot find symbol
System.out.println(s.length());          //8
```

For multidimensional arrays length variable will return only base size but not total size.

```
Example: int[][] a=new int[6][3];
System.out.println(a.length);            //6
System.out.println(a[0].length);         //3
```

Anonymous Arrays

An array without name is called anonymous arrays. The main objective of anonymous arrays is "just for instant use". We can create anonymous array as follows.

```
new int[]{10,20,30,40};                  (valid)
new int[J[]{{10,20},{30,40}};            (valid)
```

At the time of anonymous array creation we can't specify the size otherwise we will get compile time error.

Example

```
class Anarray
{
public static void main(String [] args)
{
        System.out.println(sum(new int[]{1,2,3,4,5}));
 }
    public static int sum(int [] a)
    {
        int sum=0;
        for (int i :a)
        {
            sum=sum+i;
 }
        return sum;
 }
}
```

Array element assignment

Case 1

```
int [] x=new int[5]
x[0]=10;
x[1]='a';                    //97 will be stored
byte b=30
x[2]=b;
short s=30
x[3]=s;
x[5]=10L                     //CE: possible loss of precision found: long required: int.
```

In case of primitive type array awe can provide any type of element which can be promoted to declared type. For float type arrays the allowed element types are byte, short, char, int, long, float.

Case 2:

The case of Object type arrays as array elements we can provide either declared type objects or its child class objects.

```
Object[] a=new Object[10];
a[0]=new lnteger(10);        //(valid)
a[1]=new Object();           //(valid}
a[2]=new String("CSE"};      //(valid)

Number[] n=new Number[10];
n[0]=new lnteger(10);        //{valid)
n[l]=new Double(10.5);       //{valid)
n[2]=new String("CSE");      //C.E:incompatible types/ /(invalid)
```

Case 3

In the case of interface type arrays as array elements we can provide its implemented class objects.

```
Runnable[] r=new Runnable[10];
r[0]=new Thread();
r[l]=new String("CSE");      //C.E: incompatible types
```

Array Variable assignment

Case 1

Element level promotions are not applicable at array level.

A char value can be promoted to int type but char array cannot be promoted to int array.

Example

```
int[] a={l0,20,30};
char[] ch={'a', 'b', 'c'};
int[] b=a;                          //(valid)
int[] c=ch;                         //C.E: incompatible types{invalid)
```

Which of the following promotions are valid?

char	int	(valid)
char[]	int[]	(invalid)
int	long	(valid)
int[]	long[]	(invalid)
double	float	(invalid)
doublel]	float[]	(invalid)
String	Object	(valid)
String[]	Object[]	(valid)

In the case of object type arrays child type array can be assign to parent type array variable.

Example

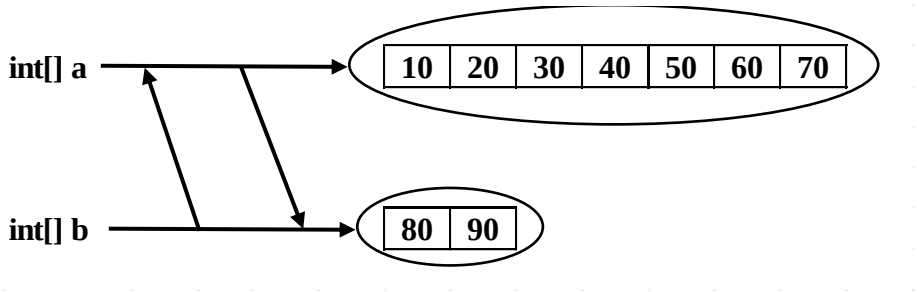
```
String[] s={"A", “B”};
Object[] o=s;
```

Case 2

Whenever we are assigning one array to another array internal elements won't be copy just reference variables will be reassigned hence sizes are not important but types must be matched.

Example

```
int[] a={l0,20,30,40,50,60, 70};
int[] b={80,90};
a=b;                                //(valid)
b=a;                                //(valid)
```



Case 3

Whenever we are assigning one array to another array dimensions must be matched that is in the place of one-dimensional array, we should provide the same type only otherwise we will get compile time error.

Example

```
int[][] a=new int[3][];
a[0]=new int[4][5];                 //C.E:incompatible types(invalid)
a[0]=10;                            //C.E:incompatible types(invalid)
a[0]=new int[4];                    //(valid)
```

Uninitialized arrays

Example

```
class Test
{
        int[] a;
        public static void main(String[] args)
        {
                Test t=new Test();
```

```
        System.out.println(t.a);                    //null
        System.out.println(t.a[0]};                 //R.E:NullPointerException
    }
}
```

Instance level
Example

```
int[] a;
System.out.println(obj.a);                  //null
System.out.println( obj.a[0]);              //R. E:NullPointerException
```

Example

```
int[] a=new int[3];
System.out.println(obj.a);                  //[1@3e25a5
System.out.println(obj.a[0]);               //0
```

Static level
Example

```
static int[] a;
System.out.println(a);                      //null
System.out.println(a[0]);                   //R.E:NullPointerException
```

Example

```
static int[] a=new int[3];
System.out.println(a);                      //[1@3e2Sa5
System.out.println(a[0]);                   //0
```

Local level
Example

```
int[] a;
System.out.println(a);                      //C.E: variable a might not have been initialized
System.out.println(a[0]);
```

Example

```
int[] a=new int[3];
System.out.println(a);                      //[I@3e25a5
System.out.println(a[0]);                   //0
```

Once we created an array every element is always initialized with default values irrespective of whether it is static or instance or local array.

Var-arg methods (variable no of argument methods)

- Until 1.4v we can't declare a method with variable no. of arguments. If there is a change in no of arguments compulsory, we have to define a new method.
- But from 1.5 version onwards we can declare a method with variable no. Of arguments such type of methods are called var-arg methods. We can declare a var-arg method as follows.

methodname (int ... x) ellipse

We can call or invoke this method by passing any no. of int values including zero number.

```
class Test
{
    public static void func(int ... x)
    {
        System.out.println("var-arg method");
```

```
	}
	public static void main(String(] args)
	{
		func();
		func(l0);
		func(l0,20,30);
	}
}
```

Output

```
var-arg method
var-arg method
var-arg method
```

- Internally var-arg parameter implemented by using single dimensional array therefore, within the var-arg method we can differentiate values by using index.

```
class Test
{
	public static void sum(int … x)
	{
		 int total=0;
		for(int i=0;i<x.length;i++)
		{
			total=total+x[i];
		}
		System.out.println("The sum :"+total);
	}
	public static void main(String[] args)
	{
		sum();
		sum(l0);
		sum(l0,20);
		sum(l0,20,30,40);
	}
}
```

Output:

```
The sum: 0
The sum: 10
The sum: 30
The sum: 100
```

- We can mix var-arg parameter with general parameters also.

```
method (int a,int ... b)
method (String s,int... x)
```

- If we mix var-arg parameter with general parameter then var-arg parameter should be the last parameter.
- We can take only one var-arg parameter inside var-arg method

```
methodOne(int ... a, int ... b)		(invalid)
```

```
class Test
{
	public static void sum(int i)
	{
		System.out.println("general method");
```

```
    }
    public static void sum(int ... i)
    {
        System.out.println("var-arg method");
    }
    public static void main(String[] args)
    {
        sum();          //var-arg method
        sum(l0,20);     //var-arg
        sum(l0);        //general method
    }
}
```

- Inside a class we can not declare var-arg method and corresponding one-dimensional array as argument in method. Otherwise we will get compile time error.

```
class A
 {
    public void func(int[] i)
    {
    }
    public void func(int ... i)
    {
    }
}
```

Output

Compile time error. Cannot declare both func(int ...) and func(int[] in A

```
class Test
{
    public static void methodOne(int i)
    {
        System.out.println("general method");
    }
    public static void methodOne(int ... i)
    {
        System.out.println("var-arg method");
    }
    public static void main(String[] args)
    {
        methodOne();//var-arg method
        methodOne(l0,20);//var-arg method
        methodOne(l0);//general method
    }
}
```

In general var-arg method will get least priority that is if no other method matched then only var-arg method will get executed.

Single Dimensional Array Vs Var-Arg Method

Wherever single dimensional array present we can replace with var-arg parameter.

Func(int[] i) ⟶ func(int ... i) (valid)

Wherever var-arg parameter present we can't replace with single dimensional array.

Func(int…i) ⟶ func(int [] i) (invalid)

Main Method

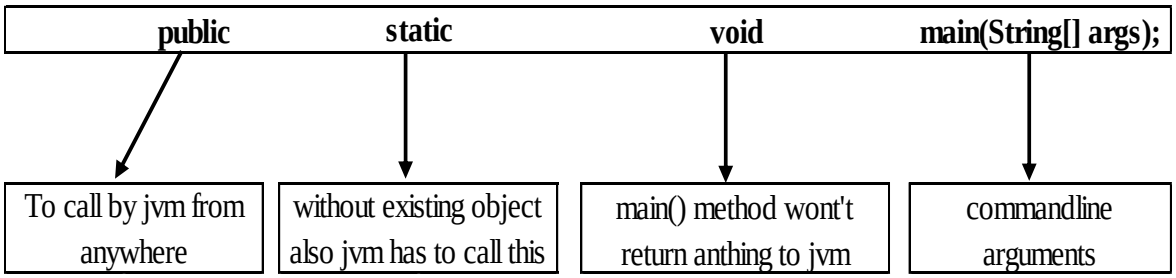
Whether the class contains main() method or not and whether it is properly declared or not these checking's are not responsibilities of the compiler, at runtime JVM is responsible for this. If jvm unable to find the required main() method then we will get runtime exception
NoSuchMethodError: main.

```
class Test
{
}
```

Output

```
 javac Test.java
java Test
R.E: NoSuchMethodError: main
```

VM always searches for the main() method with the following signature.



If we are performing any changes to the above signature then the code won't run and will get Runtime exception saying NoSuchMethodError. Anyway the following changes are acceptable to main() method.

1. The order of modifiers is not important that is instead of public static we can take static public.
2. We can declare string[] in any acceptable form.
    - String[] args
    - String []args
    - String args[]
3. Instead of args we can use any valid java identifier.
4. We can replace string[] with var-arg parameter.

Example: main(String ... args)

5. main() method can be declared with the following modifiers. • final, synchronized, strictfp.

Overloading of the main() method is possible but JVM always calls string[] argument main() method only.

Example

```
class Test
{
        public static void main(String[] args)
        {
                        System.out.println("String[] array main method")
        }
        public static void main(int[] args)
        {
                System.out.println("int[] array main method");
        }
}
```

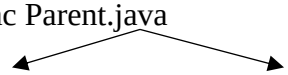
Output

```
String[] array main method
```

The other overloaded method we have to call explicitly then only it will be executed. Inheritance concept is applicable for static methods including main() method hence while executing child class if the child class doesn't contain main() method then the parent class main() method will be executed.

Example

```
class Parent
{
        public static void main{String[] args}
        {
                System.out.println("parent main"};
        }
}
class Child extends Parent
{
}
Javac Parent.java
```



```
Parent.class            Child.class
Java Parent
Parent main
Java child
Parent main
```

Example

```
class Parent
{
        public static void main(String[] args)
        {
                System.out.println("parent main");
                Child.m();

        }
}
class Child extends Parent
{
        public static void main(String[] args)
        {
                System.out.println("Child main");
        }
}
```

It seems to be overriding concept is applicable for static methods but it is not overriding it is method hiding.

Java.lang package

For writing any java program the commonly required classes and interfaces are encapsulated in the separate package called java.lang package. It is not required to import java.lang package because by default it is available to every java program.The following are some of important classes present in java.lang package.

- Object
- String
- StringBuffer
- StringBuilder
- All wrapper classes
- Execption API
- Thread APL.etc

Java.lang.Object class

Any java object whether it is predefined or customized the most commonly required methods are encapsulated into a separate class which is nothing but object class. As object class acts as a root (or) parent (or) super for all java classes, by default its methods are available to every java class. Object class defines 12 methods.

Note:

If our class does not extend any other class then only our class is the direct child class of object.

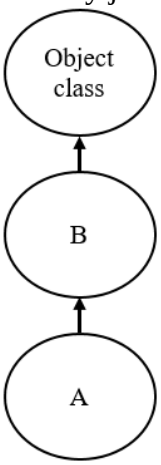
```
Class A
{
}
```

Class A is a child of class. If class extends any other class then our class is indirect child class of object.

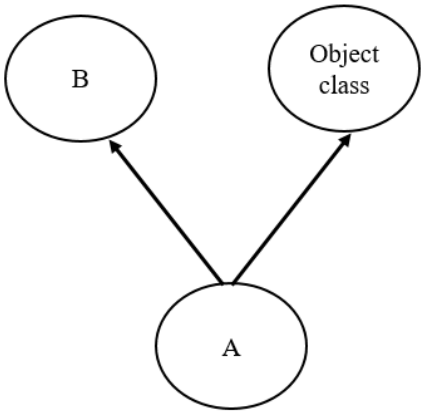
```
Class A extends B
{

}
```

Either directly or indirectly java does not support multiple inheritance.



Multilevel inheritance



Multiple inheritance (Not possible)

toString() method

We can use this method to get string representation of an object. Whenever we try to print any object reference internally toString() method will be executed. If our class doesn't contain toString() method then Object class toString() method will be executed.

Example

System.out.println(s1);⟶super(s1.toString());

Example

```
class Student
{
```

```
        String name;
        int rollno;
        Student(String name, int rollno)
        {
                this.name=name; this.rollno=rollno;
         }
        public static void main(String args[])
        {
                Student sl=new Student("rohan",101);
                Student s2=new Student("mohan",102);
                System.out.println(sl);
                System.out.println(sl.toString());
                System.out.println(s2);
        }
}
```

Output

```
Student@3e25a5
Student@3e25a5
Student@19821f
```

In the above program Object class toString() method got executed and is implemented as follows:

```
public String toString()
{
        return getClass().getName() + "@" + lnteger.toHexString(hashCode());
}
```

To provide our own String representation we have to override toString() method in our class.

```
public String toString()
{
        return name+" ..... "+rollno;
        return “name of student is:”+name+”and roll no is: “+rollno;
}
```

In String class, StringBuffer, StringBuilder, wrapper classes and in all collection classes toString() method is overridden for meaningful string representation.

```
class student
{
        public String toString()
        {
                return "student";
        }
        public static void main(String[] args)
        {
                Integer i=new lnteger(l0);
                String s=new String("CSE");
                ArrayList l=new ArrayList();
                l.add(“x”);
                l.add(“y”);
                student t=new student();
                System.out.println(i);
                System.out.println(s);
                System.out.println(t);
                System.out.println(l);
```

```
	}
}
```
Output:
10
CSE
student
[x,y]

hashCode() method

- For every object jvm will generate a unique number which is nothing but hashCode.
- HashCode of an object will be used by jvm while saving objects into HashSet, HashMap, and Hashtable etc.
- If the objects are stored according to hashCode searching will become very efficient (search algorithm hashing work based on hashCode).
- Based on our programming requirement we can override hashCode() method in our class.
- If we didn't override hashCode() method then Object class hashCode() method will be executed which generates hashCode based on address of the object but it doesn't mean hashCode represents address of the object.
- Overriding hashCode() method is said to be proper if and only if for every object we have to generate a unique number.

Example

```
class Student
{
	public int hashCode()
	{
		return 100;
	}
}
```
It is improper way of overriding hashCode() method because for every object we are generating same hashcode.

```
class Student
{
	int rollno;
	public int hashCode()
	{
		return rollno;
	}
}
```

It is proper way of overriding hashcode() method because for every object we are generating a different hashcode.

toString() method vs hashCode() method

Example:

```
class Test
{
	int i;
	Test(int i)
	{
		this.i=i;
	}
	public static void main(String[] args)
	{
		Test tl=new Test(l0);
```

```
                Test t2=new Test(l00);
                System.out.println(tl);
                System.out.println( t2 );
        }
}
```

Object class → toString() called.
Object class → hashCode() called.

In this case Object class toString() method got executed which is internally calls Object class hashCode() method.

Example

```
class Test
{
        int i;
        Test(int i)
        {
                this.i=i;
        }
        public int hashCode()
        {
                return i;
        }
        public static void main(String[] args)
        {
                Test tl=new Test(l0);
                Test t2=new Test(l00);
                System.out.println(tl);                 //Test@a
                System.out.println{t2);                 //Test@64
        }
}
```

Object →toString() called.
Test→hashCode() called.

In this case Object class toString() method got executed which is internally calls Test class hashCode() method.

Example

```
class Test
{
        int     i;
        Test(int i)
        {
                this.i=i;
        }
        public int hashCode()
        {
                return i;
        }
         public String toString()
        {
                return i+"";
        }
        public static void main(String[] args)
        {
                Test tl=new Test(l0);
```

```
                Test t2=new Test(l00);
                System.out.println(tl);
                System.out.println(t2);
        }
}
```

if we are giving the chance to object class toString() method it will internally call hashcode() but if we are overriding toString() method then our toString() method may not call hascode() method().We can use toString() method while printing object references and we can use hashCode() method while saving objects into HashSet or Hashtable or HashMap.

equals() method

We can use this method to check equivalence of two objects. If our class doesn't contain equals () method then object class .equals() method will be executed which is always meant for reference comparison.

```
class Student
{
        String name;
        int rollno;
        Student(String name,int rollno)
        {
                this.name=name;
                this.rolIno=rollno;
        }
        public static void main(String[] args)
        {
                Student sl=new Student("ravi",101);
                Student s2==new Student("rahul",102);
                Student s3=new Student("ravi",101);
                Student s4=sl;
                System.out.println(sl.equals(s2));
                System.out.println(sl.equals(s3));
                System.out.println(sl.equals(s4));
        }
}
```

Output

False
False
True

- In the above program Object class .equals() method got executed which is always meant for reference compression that is if two references pointing to the same object then only .equals() method returns true.
- In object class .equals() method is implemented as follows which is meant for reference comparison.

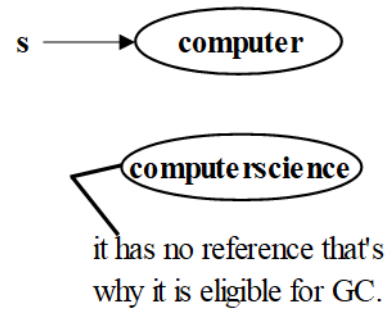
```
public boolean equals(Object obj)
 {
return (this== obj);
 }
```

Strings classes

Case1:

```
String s=new String("computer");
s.concat("science");
System.out.println(s) ;          //computer
```

Once we create a String object we can't perform any changes in the existing object. If we are try to perform any changes with those changes a new



object will be created. This behavior is called immutability of the String object.

```
StringBuffer sb=new StringBuffer("computer");
sb.append("science");
System.out.println(sb);
```

Once we created a StringBuffer object we can perform any changes in the existing object. This behavior is called mutability of the StringBuffer object.

Case2:

```
String sl=new String("computer");
String s2=new String("computer");
System.out.println(sl==s2);//false
System.out.println(s1.equals(s2));                    //true
```

In String class .equals() method is overridden for content comparison hence if the content is same .equals() method returns true even though objects are different.

String s=new String("computer");

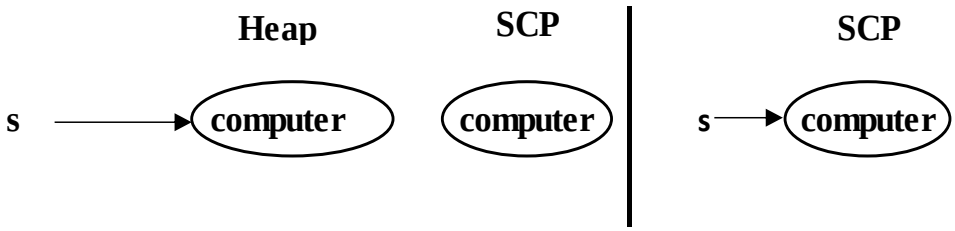
In this case two objects will be created one is on the heap the other one is SCP(String constant pool) and s is always pointing to heap object.

String s="computer";

In this case only one object will be created in SCP and s is always referring that object.

```
StringBuffer sbl=new StringBuffer("computer");
StringBuffer sb2=new StringBuffer("computer");
System.out.println(sbl==sb2);                         //false
System.out.println(sbl.equals(sb2));                  //false
```

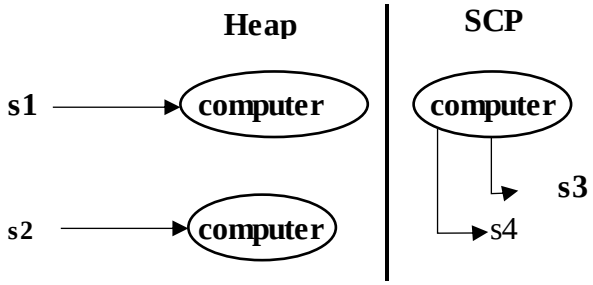
In StringBuffer class .equals() method is not overridden for content comparison hence Object class .equals() method got executed which is always meant for reference compression. Hence if objects are different .equals() method returns false even though content is same.



1. Object creation in SCP is always optional first JVM will check is there any object already created with required content or not. If it is already available then it will reuse existing object instead of creating new object. If it is not already there then only a new object will be created. Hence there is no chance of existing two objects with same content on SCP that is duplicate objects are not allowed in SCP.
2. Garbage collector can't access SCP area hence even though object doesn't have any reference still that object is not eligible for GC if it is present in SCP.
3. All SCP objects will be destroyed at the time of JVM shutdown automatically.
4. This rule is applicable for SCP only

Example

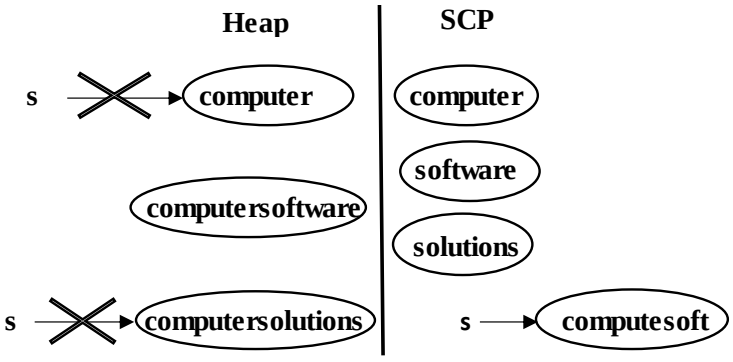
```
String sl=new String("computer");
String s2=new String("computer");
String s3="computer";
String s4="computer";
```



Whenever we are using new operator compulsory a new object will be created in the heap area hence there may be a chance of existing two objects with same content in the heap area but not in SCP. Duplicate objects are possible in the heap area but not in SCP.

Example

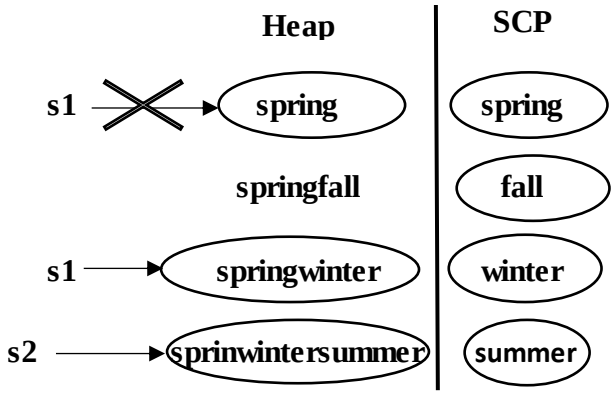
```
String s=new String("computer");
s.concat("software");
s=s.concat("solutions");
s="computersoft";
```



For every String Constant one object will be created in SCP. Because of runtime operation if an object is required to create compulsory that object should be placed on the heap but not SCP.

```
String sl=new String("spring");
        sl.concat("fall");
```

```
        sl=sl +"winter";
String s2=sl.concat("summer"); //because of runtime operation if object is created it will be created in heap
        System.out.println(sl);
        System.out.println(s2);
```




```
class StringDemo
{
        public static void main(String[] args)
        {
                String sl=new String("you cannot change me!");
                String s2=new String("you cannot change me!");
                System.out.println(sl==s2);                     //false
                String s3="you cannot change me!";
                System.out.println(sl==s3);                     //false
                String s4="you cannot change me!";
                System.out.println(s3==s4);                     //true
                String s5="you cannot "+"change me!";
                System.out.println(s3==s5);                     //true
                String s6="you cannot";
                String s7=s6+"change me!";
                System.out.println(s3==s7);                     //false
                final String s8="you cannot";
                String s9=s8+"change me!";
                System.out.println( s3==s9};                    //false
                System.out.println( s6==s8};                    //true
        }
}
```

Importance of String constant pool (SCP)

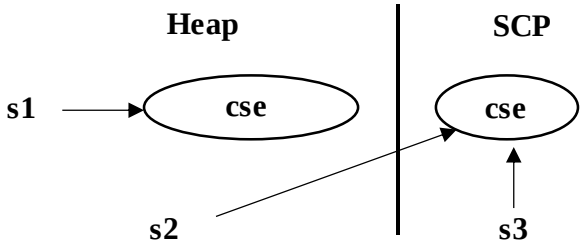
- In our program if any String object is required to use repeatedly then it is not recommended to create multiple object with same content it reduces performance of the system and effects memory utilization.
- We can create only one copy and we can reuse the same object for every requirement. This approach improves performance and memory utilization we can achieve this by "SCP".
- In SCP several references pointing to same object the main disadvantage in this approach is by using one reference if we are performing any change the remaining references will be impacted. To parent this sun people declared String objects as immutable.
- According to this once we creates a String object we can't perform any changes in the existing object if we are trying to perform any changes with those changes a new String object will be created hence immutability is the main disadvantage of SCP.

interning of String objects

By using heap object reference, if we want to corresponding SCP object reference we should go for intern () method.

```
class StringInternDemo
{
        public static void main(String[] args)
        {
                String sl=new String(“cse");
                String s2=s1.intern();
                String s3="cse”;
                System.out.println(s2==s3);   //true

        }
}
```



If the corresponding object is not there in SCP then intern() method itself will create that object and returns it.

String class constructors:

String s=new String();

Creates an empty String Object. (String object having zero length)

String s=new String(String literals);

To create an equivalent String object for the given String literal on the heap.

String s=new String(StringBuffer sb);

Creates an equivalent String object for the given StringBuffer.

String s=new String(char[] ch);

Creates an equivalent String object for the given char array.

Example:

```
class StringDemo
{
        public static void main(String[] args)
        {
                char[] ch={'a','b','c'};
                String s=new String(ch);
                System.out.println(ch);          //abc
        }
}
```

String s=new String(byte[] b);

For the given byte[] we can create a String.

Example

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

```
	{
		byte[] b={l00,101,102};
		String s=new String(b);
		System.out.println(s);			//def
	}
}
```

Important methods of String class

public char charAt(int index)

Returns the character locating at specified index.

Example:

```
class StringlnternDemo
{
	public static void main (String [] args)
	{
		String s="computer";
		System.out.println(s.charAt(3 ));	//p
		System.out.println(s.charAt( 100)) ;	//StringlndexOutOfBoundsException
	}
}
```

public String concat(String str);

Example:

```
class StringlnternDemo
{
	public static void main(String[] args)
	{
		String s="computer";
		s=s.concat("software");
		//s=s+"software";
		/ /s+="software";
		System.out.println(s);					//computersoftware
	}
}
```

The overloaded"+" and"+=" operators also meant for concatenation purpose only.

public boolean equals(Object o);

For content comparison where case is important.

It is the overriding version of Object class .equals() method.

public boolean equalslgnoreCase(String s);

For content comparison where case is not important.

Example:

```
class StringlnternDemo
{
	public static void main(String[] args)
	{
		String s="computer";
		System.out.println(s.equals("computer"));			//true
		System.out.printl n(s.equalslgnoreCase("COMPUTER"));		//true
	}
}
```

public String substring(int begin);

Return the substring from begin index to end of the string.

Example

```
class StringInternDemo
{
        public static void main(String[] args)
        {
                String s="systemsoftware";
                System.out.println(s.substring(6) );                    //software
        }
}
```

public String substring(int begin, int end);

Returns the substring from begin index to end-1 index.

Example

```
class StringInternDemo
{
        public static void main(String[] args)
        {
                String s="systemsoftware";
                System.out.println(s.substring(6) );
                System.out.println(s.substring(3, 7));
        }
}
```

public int length();

Returns the no of characters present in the string.

Example:

```
class StringInternDemo
{
        public static void main(String[] args)
        {
                String s="systemsoftware";
                System.out.println(s.length());                 //14
                //System.out.println(s.length);                 //compile time error
                //StringInternDemo.java:7: cannot
                find symbol
                //symbol : variable length
                //location: class java.lang.String
        }
}
```

public String replace(char old,char new)

To replace every old character with a new character.

Example:

```
class StringtnternDemo
{
        public static void main(String[J args)
        {
                String s=”abababb”;
                System.out.println(s.replace(‘a’,’b'));         //bbbbbb
```

```
    }
}
```

public String tolowerCase()

Converts the all characters.of the string to lowercase.

Example

```
class StringInternDemo
{
    public static void main(String[J args)
    {
        String s="CSE";
        System.out.println(s.tolowerCase()) ;          //cse
}
```

public String toUpperCase()

Converts the all characters of the string to uppercase.

Example

```
class StringInternDemo
{
    public static void main(String[] args)
    {
        String s="cse";
        System.out.println(s.toUpperCase() );          //CSE
    }
}
```

public String trim()

We can use this method to remove blank spaces present at beginning and end of the string but not blank spaces present at middle of the String.

Example

```
class StringInternDemo
{
    public static void main(String[] args)
    {
        Strings=" com puter ";
        System.out.println(s.trim());                  //com puter
    }
}
```

public int indexOf(char ch)

It returns index of 1st occurrence of the specified character if the specified character is not available then return -1.

Example

```
class StringInternDemo
{
    public static void main(String[] args)
    {
        String s="infosys";
        System.out.println(s.indexOf('s'));            //4
        System.out. println(s.indexOf('e') );          //-1
    }
}
```

public int lastlndexOf(Char ch)

It returns index of last occurrence of the specified character if the specified character is not available then return -1.

Example

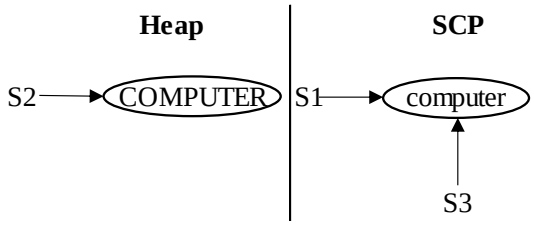
```
class StringlnternDemo
{
        public static void main(String[] args)
        {
                String s="infosys";
                System.out.println(s.lastl ndexOf('s')) ;          //6
                System.out.println(s.indexOf('z'));//              -1
        }
}
```

Note

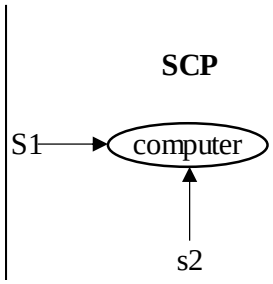
- Because of runtime operation if there is a change in content then with those changes a new object will be created only on the heap but not in SCP.
- If there is no change in content no new object will be created the same object will be reused.

Example

```
class StringlnternDemo
{
        public static void main(String[) args)
        {
                String sl="computer";
                String s2=sl.toUpperCase();
                String s3=s1.tolowerCase();
                System.out.println(s1==s2);                //false
                System.out.println(sl==s3);                //true
        }
}
```




```
class StringlnternDemo
{
        public static void main(String[] args)
        {
                String sl="computer";
                String s2=s1.toString();
                System.out.printl n( s 1==s2);          //true
        }
}
```



StringBuffer

If the content in the application are changing frequently then never recommended to go for String object because for every change a new object will be created internally. Therefore, to handle this type of requirement we should go for StringBuffer concept. The main advantage of StringBuffer over String is, all required changes will be performed in the existing object only instead of creating new object.

Constructors

StringBuffer sb=new StringBuffer();

It will create an empty StringBuffer object with default initialcapacity "16" and once StringBuffer object reaches its maximum capacity a new StringBuffer object will be created with

Newcapacity = (currentcapacity+1)*2.

Example

```
class StringBufferDemo
{
        public static void main{String[] args)
        {
                StringBuffer sb=new StringBuffer();
                System.out.println(sb.capacity());              //16
                sb.append("abcdefghijklmnop");
                System.out.println(sb.capacity());              //16
                sb.append("q");
                System.out.println(sb.capacity{));              //34
        }
}
```

StringBuffer sb=new StringBuffer(int initialcapacity)

Creates an empty StringBuffer object with the specified initial capacity.

Example:

```
class StringBufferDemo
{
        public static void main(String[] args)
        {
                StringBuffer sb=new StringBuffer(19);
                System.out.println (sb.capacity()) ;            //19
        }
}
```

StringBuffer sb=new StringBuffer(String s)

Creates an equivalent StringBuffer object for the given String with capacity=s.length()+l6;

Example:

```
class StringBufferDemo
{
        public static void main(String[] args)
        {
                StringBuffer sb=new StringBuffer("computer");
                System.out.println(sb.capacity());                      //24
        }
}
```

Important methods of StringBuffer

public int length();

Return the no of characters present in the StringBuffer.

public int capacity();

Returns the total no of characters but a StringBuffer can accommodate(hold).

public char charAt(int index);

It returns the character located at specified index.

```
class StringBufferDemo
{
        public static void main(String[J args)
        {
                StringBuffer sb=new StringBuffer(“computerscience”);
                System.out.println(sb.length());                    //15
                System.out.println(sb.capacity(});                  //31
                System.out. pri ntl n( sb.charAt(14)) ;             //e
        }
}
```

public void setCharAt(int index,char ch);

To replace the character locating at specified index with the provided character.

Example:

```
class StringBufferDemo
{
        public static void main(String[] args)
        {
                StringBuffer sb=new StringBuffer("comouter");
                sb.setCharAt(3,’p’);
                System.out.println(sb);
        }
}
```

public StringBuffer append(String s);

```
public StringBuffer append(int i);
public StringBuffer append(long I);
public StringBuffer append(boolean b);
public StringSuffer append(double d);
public StringBuffer append(float f);
```

All these are overloaded methods.

```
class StringBufferDemo
{
        public static void main(String[] args)
        {
                StringBuffer sb=new StringBuffer();
                sb.append("PI value is :");
                sb.append(3.14);
                sb.append(" this is exactly ");
                sb.append(true);
                System.out.println(sb);                 //PI value is :3.14 this is exactly true
        }
}
```

public StringBuffer insert(int index,String s);
public StringBuffer ir.sert(int index,int i);
public StringBuffer insert(int index,long I);
public StringBuffer insert(int index,double d);
public StringBuffer insert(int index,boolean b);
public StringBuffer insert(int index,float f);

All are overloaded methods

To insert at the specified location.

Example:

```
class StringBufferDemo
{
	public static void main(String[] args)
	{
		StringBuffer sb=new StringBuffer("CSE");
		sb.insert( 3, "morning");
		sb.insert(l0,"5");
		System.out.println(sb );		//CSEmorning5
	}
}
```

public StringBuffer delete(int begin,int end)

To delete characters from begin index to end n-1 index.

public StringBuffer deleteCharAt(int index)

To delete the character locating at specified index.

Example:

```
class StringBufferDemo
{
	public static void main(String[] args)
	{
		StringBuffer sb=new StringBuffer("surajmalinstitute");
		System.out.println(sb);				//surajmalinstitute
		sb.delete(8,17);
	}
	System.out.println(sb);					//surajmali
	sb.deleteCharAt(8);
	System.out.println(sb);					//surajmal
}
```

public StringBuffer reverse();

Example:

```
class StringBufferDemo
{
	public static void main(String[] arg ... )
	{
		StringBuffer sb=new StringBuffer("CSEMORNING");
		System.out.println(sb) ;				//CSEMORNING
		System.out.println(sb.reverse() );			//GNINROMESC
	}
}
```

public void setLength(int length)

Consider only specified no of characters and remove all the remaining characters.

Example

```
class StringBuff,erDemo
{
	public static void main(String[] args)
	{
		StringBuffer sb=new StringBuffer(“computerscience”);
		sb.setlength(8);
		System.out.println(sb);			//computer
	}
}
```

public void trimToSize{);

To deallocate the extra free memory such that capacity and size are equal.

Example

```
class StringBufferDemo
{
	public static void main(String[] args)
	{
		StringBuffer sb=new StringBuffer(l000);
		System.out.println(sb.capacity() );			//1000
		sb.append("computer");
		System.out.printl n(sb.capacity()) ;			//1000
		sb.trimToSize{);
		System.out.println(sb.capacity());			//8
	}
}
```

public void ensureCapacity(int initialcapacity)

To increase the capacity dynamically based on our requirement.

Example

```
class StringBufferDemo
{
	public static void main(String[] args)
	{
		StringBuffer sb=new StringBuffer();
		System .out.println(sb.capacity() );			//16
		sb.ensureCapacity(100);
		System.out.println(sb.capacity()).;			//100
	}
}
```

StringBuilder

Every method present in StringBuffer is declared as synchronized hence at a time only one thread is allowed to operate on the StringBuffer object due to this, waiting time of the threads will be increased and effects performance of the system. To overcome this problem sun introduced the concept StringBuilder in 1.5v. StringBuilder is exactly same as StringBuffer except the following differences.

StringBuffer	StringBuilder
• Every method present in StringBuffer is synchronized.	• No method present in StringBuilder is synchronized.
• At a time only one thread is allow to operate on the StringBuffer object hence StringBuffer object is Thread safe.	• Multiple Threads are allowed to operate simultaneously on the StringBuilder object hence StringBuilder is not Thread safe.
• It increases waiting time of the Thread and hence relatively performance is low.	• Threads are not required to wait and hence relatively performance is high.
• Introduced in 1.0 version.	• Introduced in 1.5 versions.

String vs StringBuffer vs StringBuilder

- If the content is fixed and won't change frequently then we should go for String.
- If the content will change frequently but Thread safety is required then we should go for StringBuffer.
- If the content will change frequently and Thread safety is not required then we should go for StringBuilder.