* Method overloading :-

→ If a class have multiple methods by same name but different parameters, it is known as Method overloading.

→ In java whenever a method is being called, first the name of the method is matched and then, the number and type of arguments passed to that methods are matched.

→ Method overloading is a feature that allows a class to have two or more methods having same name but different parameters.

There are two different ways of method overloading

- Method overloading by changing data type of arguments
- Method overloading by changing no. of arguments.

Example :-
Method overloading by changing datatype of arguments.

```
Class Calculate
{
 void  sum( int a, int b)
 {
  System.out.println ("sum is :" + (a+b));
 }
 void  sum(float a, float b)
 {
  System.out.println (" sum is :" + (a+b));
 }
 public static void main (String args[])
 {
  Calculate  cal = new Calculate();
  cal. sum (8, 5); // sum (int a, int b) is method is called.
  cal. sum (4.6f, 3.8f); // sum (float a, float b) is called.
 }
}
```

```
%% javac calculate.java
      java Calculate
      Sum is 13
      sum is 8.4
```

Example :-

Method overloading by changing no. of argument.

```
class Calsum
{
 void sum(int a, int b)
 {
  System.out.println(a+b);
 }
 void sum(int a, int b, int c)
 {
  System.out.println(a+b+c);
 }
 public static void main(String args[])
 {
  Calsum obj = new Calsum();
  obj.sum(10,10,10); // sum() with 3 parameters
  obj.sum(20,20); // sum() with 2 parameters
 }
}
```

o/p: javac Calsum.java
java Calsum
30
40

* Constructor :-

→ constructor in java is a special type of method that is used to initialize the object. Java constructor is invoked at the time of object creation.

→ It constructs the values i.e provides data for the object that is why it is known as constructor.

→ Java constructors are the methods which are used to initialized objects.

There are basically two rules defined for the constructor

→ constructor name must be same as its class name.

→ Constructor must have no explicit return type.

There are two types of constructors

- Default constructor
- parameterized constructor

- **Default constructor :-**

A constructor that have no parameter is known as default constructor. This is used to provides default values to an object.

Example :-

```
class DefaultConstr
{
DefaultConstr() // constructor method
{
System.out.println("Default Constructor method called.");
}
public static void main(String args[])
{
DefaultConstr dc = new DefaultConstr();
}
}
```

output:- javac DefaultConstr.java
java DefaultConstr
Default Constructor method called.

- **parameterized constructor :-**

→ A parameterized constructor is a constructor, that has parameters. (or) A constructor that have parameters is known as parameterized constructor.

→ This is used to provide different values to the distinct objects.

Example :-

In the following example, we have created the constructor of student class that have two parameters.

```
class Students
{
 int id;
 String name;
 Students (int i, String n) // parameterized constructor.
 {
  id = i;
  name = n;
 }
 void display() // Method
 {
  System.out.println( id + "  " + name);
 }
 public static void main(String args[])
 {
  Students S1 = new Students (521, "Madhu");
  Students S2 = new Students (512, "Hari");
  S1.display();
  S2.display();
 }
}
```

output:-

```
javac Students.java
java Students
521  Madhu
512  Hari
```

* Constructor Overloading :-

Constructor overloading is a technique in java in which a class can have any number of constructors that differ in parameter lists.

The compiler differentiates those constructors by taking into account the number of parameters in the list and their type.

Example

```
class Sum
{
  int a,b,c;
Sum(int x, int y) // with 2 parameters   constructor
{
  a= x;
  b= y;
}
Sum (int x, int y, int z) // constructor with 3 parameter
{
  a= x;
  b= y;
  c = z;
}
void display()
{
 System.out.println("sum is" + (a+b+c));
}
public static void main(String args[])
{
  Sum s1 = new Sum(10, 11); // With 2 parameters
  Sum s2 = new Sum(10,20,30); // with 3 parameters
  s1. display();
  s2. display();
}
}
```

output: javac Sum.java

java Sum

Sum is 21

Sum is 60

There are some differences between constructors and methods. These are,

Constructor	Method
* Constructor is used to initia-lize values to an object.	* Method is used to expose behavior of an object.
* Constructor must not have return type.	* Method must have return type.
* Constructor is invoked implicitly	* Method is invoked explicity
* constructor name must be same as the class name.	* Method name may or may not be same as class name.

* Java Garbage collection :-

In Java, garbage means unreferenced objects. Garbage collection is process of destroyed the 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.

An object can be unreferenced in following ways

→ By nulling the reference

Example :- Employee e = new Employee();
e = null;

→ By assigning a reference to another

Example :- Employee e1 = new Employee();
Employee e2 = new Employee();
e1 = e2

In java, the garbage collector of JVM collects only those objects that are created by new keyword. so if you have created any object without new, you can use finalize Method to perform cleanup processing (destroying remaining objects).

finalize() method:

This method invoked each time before the object is garbage collection. This method is used to destroying the objects which are not created by "new" keyword.

gc() method:

The gc() method is used to invoke the garbage collector to perform cleanup processing. But this method does not guarantee that JVM will perform the garbage collection. It is only request to the JVM for garbage collection.

Example:-

```
class GarbageDemo
{
public void finalize()
  {
   System.out.println("object is garbage collected");
  }
  public static void main(String args[]
  {
   GarbageDemo g1 = new GarbageDemo();
   GarbageDemo g2 = new GarbageDemo();
    g1 = null;
    g2 = null;
    System.gc();
  }
}
```

o/p:- javac GarbageDemo.java
java Garbage
object is garbage collected
object is garbage collected

* String class :-

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.

→ In Java, An array of characters works same as java String.

for example:

```
char[] ch={'M','a','d','h','u'};
String s = new string(ch);
```

is same as:

```
String s="Madhu";
```

→ In Java, string is an object, it can be created by using java.lang.String class.

There are two ways to create String object

* By String literal :-

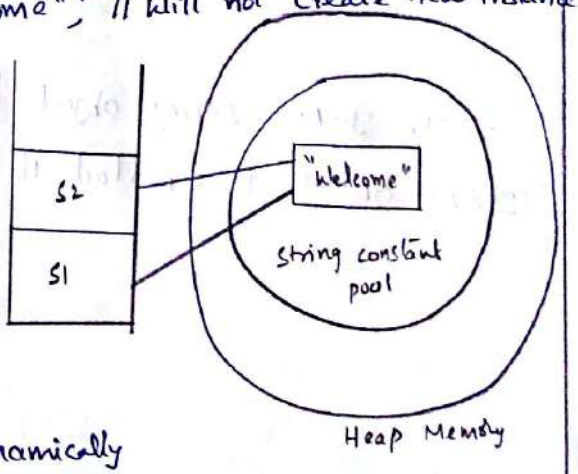
Java String literal is created by using double quotes

Ex:- `String s="Welcome";`

When we create a string literal, the JVM checks the string constant pool first, if the string already exists in the pool, a reference to the pooled instance is returned.
If String doesn't exist in the pool, a new string instance is created and placed in the pool.

for example:

```
String s1="Welcome";
String s2="Welcome"; // will not create new instance.
```



Note:- String objects are stored in a special memory area known as string constant pool. It is an constant table to store String literals.

Note:- A heap is a general term used to (for) any Memory that is allocated dynamically and randomly. (It is allocated by O.S).

* By new Keyword :-

In This, We can string object by using "new" Keyword.

for example :

String s = new String ("Welcome");

In this case, JVM will create a new string object in heap memory and the literal "Welcome" will be placed in the string constant pool.

Example :-

```
class StringExample
{
public static void main (String args[])
{
 String s1 = "Java"; // creating a 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 string by new keyword
 System.out.println (s1);
 System.out.println (s2);
 System.out.println (s3);
 }
}
```

output :- javac StringExample.java
java StringExample
Java
Strings
example.

→ In Java, string object is immutable That means once a string object is created it cannot be altered.

In Java, String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace() and etc.

* charAt():

This method returns a char value at the given index number. The index starts from 0 (zero).

Syntax:- charAt(int index)

→ It returns char value.

Example:
```
String name = "Madhu";
Char ch = name.charAt(4);
S.o.p(ch);
```
o/p:- u

* concat():

This method is used to combine two strings. And it returns combined string.

Syntax:- concat(String str2)

Example:
```
String s1 = "java string";
s1 = s1.concat(" is immutable");
S.o.p(s1);
```
o/p:- Java string is immutable

* contains():

This method searches the sequence of characters in this string. It returns true if sequence of char values are found in the given string otherwise returns false.

Syntax:- contains(char sequence)

Example:
```
String name = "Madhu Naidu Tattikota";
S.o.p(name.contains("Naidu"));
S.o.p(name.contains("Reddy"));
```
o/p:- True
false

* equals():

This method is used to compares the two given strings. If both are equal it returns true otherwise it returns false.

Syntax:- equals(anotherstring)

Example:
```
String s1 = "madhu";
String s2 = "madhu";
String s3 = "MADHU";
S.O.P(s1.equals(s2));
S.O.P(s1.equals(s3));
```
O/P: true
false.

* length():

This method is used to find the length of string. It returns count of total numbers of characters.

Syntax:- int length();

Example:
```
String s1 = "Madhu";
String s2 = "Tatikota";
S.O.P(s1.length());
S.O.P(s2.length());
```
O/P: 5
9.

* substring():

This method returns a part of the string. It is used to extract some part of given string.

Syntax:- substring(int startIndex)
and
substring(int startindex, int endindex)

Example:-
```
String s1 = "Madhu";
S.O.P(s1.substring(2,4));
S.O.P(s1.substring(3));
```
O/P: dhu
hu

* startsWith():

This method checks if This string starts with given prefix. It returns true if this starts with given prefix else returns false.

Syntax:- startsWith(String prefix)

Where prefix is sequence of characters.

Example:-
```
String s1 = "Madhu";
s.o.p(s1.startsWith("Ma"));
s.o.p(s1.startsWith("dh"));
```
o/p:
```
True
false
```

* endsWith():

This method checks if this string ends with given prefix. It returns true if this ends with given prefix else returns false.

Syntax:- endsWith(string prefix)

Where prefix is sequence of characters.

Example:-
```
String s1 = "Madhu";
s.o.p(s1.endsWith("hu"));
s.o.p(s1.endsWith("dh"));
```
o/p:
```
True
false.
```

* replace():

This method is used to replace old characters with new characters in a given string.

Syntax:- replace(char oldchar, char newchar)

Example:-
```
String s1 = "Java is an OOP";
s.o.p(s1.replace('a','i'));
s.o.p(s1.replace("is","was"));
```
o/p:-
```
Jivi is in OOP
Java was an OOP
```

* indexOf():-

This method returns index of given character value or substring. If it is not found, it returns -1, The index counter starts from zero.

Syntax:- indexOf (char ch)
indexOf (char ch, int fromindex)
indexOf (String substring)
indexOf (String substring, int fromindex)

Example:
```
String s= "this is madhu";
S.O.P (s.indexOf ('s')); // it returns 3
S.O.P (s.indexOf ('s', 4)); // it returns 6
S.O.P (s.indexOf ("is")); // it returns 2
S.O.P (s.indexOf ("is", 4)); // it returns 5
```

* toLowerCase():-

This method is used to convert given string into lower case letter.

Syntax:- toLowerCase()

Example:
```
String name= "MAdhu";
S.O.P (name.toLowerCase());
```

O/P:- madhu

* toUpperCase():-

This method is used to convert all characters in given string into upper case letter.

Syntax:- toUpperCase()

Example:
```
String name= "madhu";
S.O.P (name.toUpperCase());
```

O/P:- MADHU.