

Introduction to Java

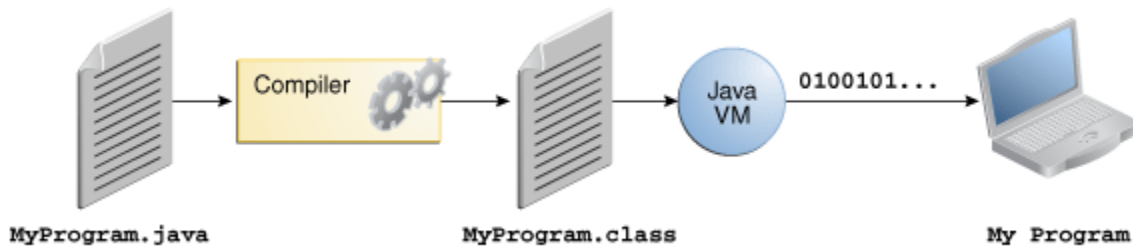
History of Java:

4. In 1990, Sun Micro Systems Inc. (US) was conceived a project to develop software for consumer electronic devices that could be controlled by a remote. This project was called Stealth Project but later its name was changed to Green Project.
5. In January 1991, Project Manager James Gosling and his team members Patrick Naughton, Mike Sheridan, Chris Wrath, and Ed Frank met to discuss about this project.
6. Gosling thought C and C++ would be used to develop the project. But the problem he faced with them is that they were system dependent languages. The trouble with C and C++ (and most other languages) is that they are designed to be compiled for a specific target and could not be used on various processors, which the electronic devices might use.
7. James Gosling with his team started developing a new language, which was completely system independent. This language was initially called **OAK**. Since this name was registered by some other company, later it was changed to **Java**.
8. James Gosling and his team members were consuming a lot of coffee while developing this language. Good quality of coffee was supplied from a place called "Java Island". Hence they fixed the name of the language as Java. The symbol for Java language is cup and saucer.
9. Sun formally announced Java at Sun World conference in 1995. On January 23rd 1996, JDK1.0 version was released.

Features of Java (Java buzz words):

1. **Simple:** Learning and practicing java is easy because of resemblance with c and C++.
2. **Object Oriented Programming Language:** Unlike C++, Java is purely OOP.
3. **Distributed:** Java is designed for use on network; it has an extensive library which works in agreement with TCP/IP.
4. **Secure:** Java is designed for use on Internet. Java enables the construction of virus- free, tamper free systems.
5. **Robust (Strong/ Powerful):** Java programs will not crash because of its exception handling and its memory management features.
6. **Interpreted:** Java programs are compiled to generate the byte code. This byte code can be downloaded and interpreted by the interpreter. .class file will have byte code instructions and JVM which contains an interpreter will execute the byte code.
7. **Portable:** Java does not have implementation dependent aspects and it yields or gives same result on any machine.
8. **Architectural Neutral Language:** Java byte code is not machine dependent, it can run on any machine with any processor and with any OS.
9. **High Performance:** Along with interpreter there will be JIT (Just In Time) compiler which enhances the speed of execution.
10. **Multithreaded:** Executing different parts of program simultaneously is called multithreading. This is an essential feature to design server side programs.
11. **Dynamic:** We can develop programs in Java which dynamically change on Internet (e.g.: Applets).

Environment of the java programming development:-



Class Contains Five elements:-

Class Test

```
{  
    1. variables  
    2. methods  
    3. constructors  
    4. instance blocks  
    5. static blocks  
}
```

TOKENS:-Smallest individual part in a java program is called Token. It is possible to provide any number of spaces in between two tokens.

Ex:-Class Test

```
{  
    Public static void main(String[] args)  
    {  
        int a=10;  
        System.out.println("java tokens");  
    }  
}
```

Tokens are----->class,test,{,},[,-----etc

Identifiers:-

any name in the java program like variable name,class name,method name,interface name is called identifier.

<pre>class Test { void add() { int a=10; int b=20; } };</pre>	<pre>} Test----->identifier add----->identifier a----->identifiers b----->identifiers</pre>
---	---

Rules to declare identifiers:-

1. the java identifiers should not start with numbers, it may start with alphabet symbol and underscore symbol and dollar symbol.
 - a. `Int abc=10;----->valied`
 - b. `Int 2abc=20;---->not valied`
 - c. `Int _abc=30;---->valied`
 - d. `Int $abc=40;---->valied`
 - e. `Int @abc=50;--->not valied`

2. The identifier will not contains symbols like

`+, -, ., @, #, *`

3. The identifier should not duplicated.

`class Test`

`{`

`void add()`

`{`

`int a=10;`

`int a=20;`

`}`

the identifier should not be duplicated.

`}`

`};`

4. In the java applications it is possible to declare all the predefined class names and predefined interfaces names as a identifier. But it is not recamanded to use.

JAVA NAMING CONVENTIONS:-

Java is a case sensitive language so the way of writing code is important.

1. All Java classes, Abstract classes and Interface names should start with uppercase letter ,if any class contain more than one word every innerword also start with capital letters.

Ex: `String`

`StringBuffer`

`FileInputStream`

2. All java methods should start with lower case letters and if the method contains more than one word every innerword should start with capital letters.

Ex :-

`post()`

`toString()`

`toUpperCase()`

3. All java variables should start with lowercase letter and inner words start with uppercase letter.

Ex:

`pageContent`

`bodyContent`

4. All java constant variables should be in uppercase letter.

Ex: `MIN_PRIORITY`

`MAX_PRIORITY`

`NORM_PRIORITY`

5. All java packages should start with lower case letters only.

Ex: `java.awt`

`Java.io`

NOTE:-

The coding standards are applicable for predefined library not for user defined library .But it is recommended to follow the coding standards for user defined library also.

JAVA COMMENTS :-

To provide the description about the program we have to use java comments.

There are 3 types of comments present in the java language.

1) Single line Comments:-

By using single line comments we are providing description about our program within a single line.

Starts with.....>// (double slash)

Syntax:- //description

2) Multi line Comments:-

This comment is used to provide description about our program in more than one line.

Syntax: - /*line-1
 line-2
 */

3) Documentation Comments:-

This comment is used to provide description about our program in more than one page.

In general we are using document comment to prepare API kind of documents but it is not sujestable.

Syntax: - /*line-1
 *line-2
 *line-3
 */

Data Types:-

- 1) Data types are used to represent the type of the variable and type of the expression.
- 2) Data types are used to specify the how much memory is allocated for variables.

Data type	Size	Range	Default values
Byte	1	-128 to 127	0
short	2	-32768 to 32767	0
int	4	-2147483648 to 2147483647	0
long	8	-2 ³¹ to 2 ³¹ -1	0
float	4	-3.4e38 to 3.4e308	0.0
double	8	-1.7e308 to 1.7e308	0.0
char	2	0 to 65535	Single space character
Boolean	NA	Not Applicable	False

Syntax:- **data-type name-of-variable=value/literal;**

Ex:- int a=10;
 Int----->Data Type
 a----->variable name
 =----->assignment
 10----->constant value
 ; ----->statement terminator

Types of variables:-

- ❓ Variables are used to store the values. By storing that values we are achieving the functionality of the project.
- While declaring variable we must specify the type of the variable by using data type's concept.

In the java language we are having three types of variables

1. Local variables
2. Instance variables
3. Static variables

Local variables:-

- a. The variables which are declare inside a method & inside a block & inside a constructor is called local variables
- b. The scope of local variables are inside a method or inside a constructor or inside a block.
- c. We are able to use the local variable only inside the method or inside the constructor or inside the block only.

Ex:-

```
class Test
```

```

{
    public static void main(String[] args)
    {
        int a=10;    } Local variables
        int b=20;    }
        System.out.println(a+b);
    }
}

```

Instance variables:-

1. The variables which are declare inside a class and outside of the methods is called instance variables.
2. We are able to access instance variables only inside the class any number of methods.

```

class Test
{
    int a=10;
    int b=20;
    void add()
    {
        System.out.println(a+b);
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        System.out.println(t.a+t.b);
        t.add();
    }
}

```

3. static variables:-

- The instance variables which are declared as a static modifier such type of variables are called static variables.
- We are able to access static variables within the class any number of methods.

```

class Test
{
    static int a=10;
    static int b=20;
    public static void main(String[] args)
    {
        System.out.println(a+b);
    }
    void add()
    {
        System.out.printl(a+b);
    }
}

```

Calling of static variables:-

- a. Directly possible.
- b. By using class name possible.
- c. By using reference variable possible.

```

class Test

```

```

{
    static int x=100;
    public static void main(String[] args)
    {
        //1-way(directly possible)
        System.out.println(a);
        //2-way(By using class name)
        System.out.println(Test.a);
        //3-way(By using reference variable)
        Test t=new Test();
        System.out.println(t.a);
    }
};

```

Instance vs Static variables:-

1. Instance variable for the each and every object one separate copy is maintained.
2. Static variable for all objects same copy is maintained. One Object change the value another object is affected.

```

class Test
{
    int a=10;
    static int b=20;
    public static void main(String... ratan)
    {

        Test t1=new Test();
        System.out.println(t1.a);//10
        System.out.println(t1.b);//20
        t1.a=444;
        t1.b=555;
        Test t2=new Test();
        System.out.println(t2.a);//10
        System.out.println(t2.b);//555
        t2.b=111;
        System.out.println(t2.b);//111
        Test t3=new Test();
        System.out.println(t3.a);//10
        System.out.println(t3.b);//111
        Test t4=new Test();
        System.out.println(t4.a);//10
        System.out.println(t4.b);//111
    }
};

```

Variables default values:-

Case 1:- for the instance variables the JVM will provide the default values.

```

class Test
{
    int a=10;
    int b;
    public static void main(String[] args)
    {

```

```

        Test t=new Test();
        System.out.println(t.a);//10
        System.out.println(t.b);//0
    }
}

```

Case 2:- for the static variables the JVM will provide the default values.

```

class Test
{
    static double d=10.9;
    static boolean b;
    public static void main(String[] args)
    {
        System.out.println(d);//10.9
        System.out.println(b);//false
    }
}

```

Case 3:- for the local variables the JVM won't provide the default values before using those variables we have to provide the default values.

```

class Test
{
    public static void main(String[] args)
    {
        byte a=10;
        int b;
        System.out.println(a);
        System.out.println(b);
    }
}

```

Compilation error:- Variables b might not have been initialized
 System.out.println(b);

Practice example:-

```

class Test
{
    //2-instance variables
    int a=10;
    boolean b;
    //2-static variables
    static int c=20;
    static double d;
    //1-instance methods
    void m1()
    {
        System.out.println(a);//10
        System.out.println(b);//false
        System.out.println(c);//20
        System.out.println(d);//0.0
    }
    //1-static method
}

```



```

        static void m2()
        {
            Test t=new Test();
            System.out.println(t.a);//10
            System.out.println(t.b);//false
            System.out.println(c);//20
            System.out.println(d);//0.0
        }
        public static void main(String[] args)
        {
            Test t=new Test();
            t.m1();
            m2();
        }
    }

```

Class vs Object:-

- ❓ Class is a group of objects that have common property.
- ❓ Java is classes based language we are able to design the program by using classes and objects.
- ❓ Object is a realworld entity. Object orientation is methodology to design a program by using classes and objects.
- ❓ Object is physical entity where as class is a logical entity.
- ❓ A class is a template or blue print from which type of objects are created.
- ❓ Object is nothing but instance of a class.

Every objects contains 3 characterstics

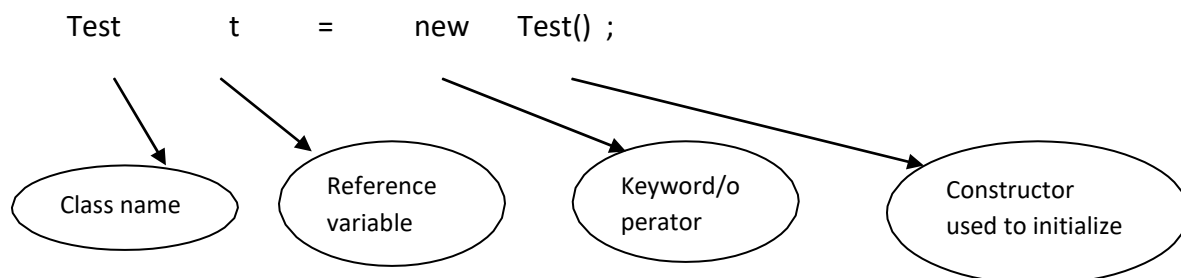
1. State(represent data of an object)
2. Behavior(represent behavior of an object)
3. Identity(used to identify the objects uniquely).

PEN(object):-

State:- name raynolds,color red etc.....

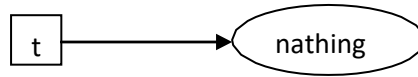
Behaviour:- used to write

Declaration :- it is representing the variable associated with object.
Instantiation :- The new keyword is a Java operator that creates the object.
Initialization :- The new operator is followed by a call to a constructor, which initializes the the new object.



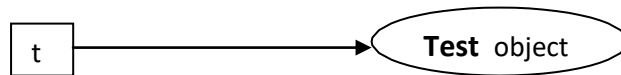
Test t;

- It notifies the compiler refers the Test data with t reference variable.
- Declaring reference variable doesn't mean creating object we must use new operator to create object.
t is reference variable pointing to nothing.



Test t=new Test();

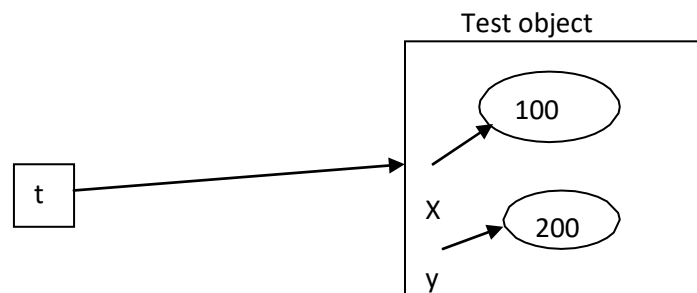
- The new operator instantiating class by allocating memory for new object and the reference variable pointing to that object.



Test t=new Test(100,200);

Constructors are executed as part of the object creation. If we want perform any type of initializations at the time of object creation use constructors.

```
Class Test
{
    int x;
    int y;
    //Constructor
    Test(int a,int b)
    {
        x=a;
        y=b;
    }
    public static void main(String args[])
    {
        Test t=new Test(100,200)
    }
};
```



Methods (behaviors):-

- Methods are used to provide the business logic of the project.
- The methods like a functions in C-language called functions, in java language is called methods.
- Inside the class it is possible to declare any number of methods based on the developer

requirement.

- 4) As a software developer while writing method we have to follow the coding standards like the method name starts with lower case letters if the method contains two words every inner word also starts uppercase letter.
- 5) It will improve the reusability of the code. By using methods we can optimize the code.

Syntax:-

[modifiers-list] return-Type Method-name (parameter-list) throws Exception

Ex:-

Public void m1()

Public void m2(int a,int b)

Method Signature:-

The name of the method and parameter list is called Method Signature. Return type and modifiers list not part of a method signature.

Ex:- m1(int a,int b)-----→ Method Signature
 m2();-----→ Method signature

Method declaration:-

Ex:-

void m1() → declaration of the method.
{

statement-1
statement-2
statement-3
statement-4 } implementation/body

}

Ex:-

Void deposit()

{

System.out.println("money deposit logic");

System.out.println("congrats your money is deposited");
}

defining a method

functionality of a method

There are two types of methods:-

Instance method

Static method

Instance method:-

void m1()

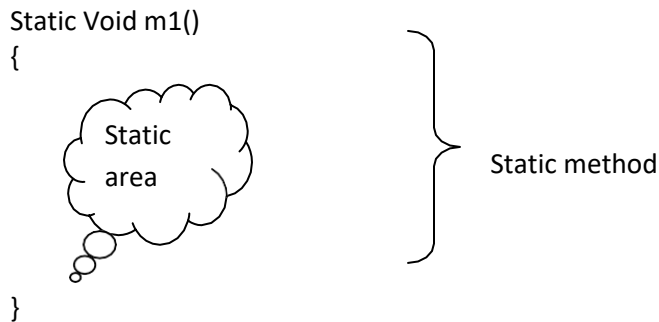
{

Instance
area

}

Instance method

Static Method:-



Ex :- instance methods without arguments.

```
class Test
{
    void ()
    {
        System.out.println("software solutions");
    }
    void soft()
    {
        System.out.println("software solutions");
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.();
        t.soft();
    }
}
```

Ex:-instance methods with parameters.

```
class Test
{
    void m1(int i,char ch)
    {
        System.out.println(i+"-----"+ch);
    }
    void m2(float f,String str)
    {
        System.out.println(f+"-----"+str);
    }
    public static void main(String[] args)
    {
        System.out.println("program statrs ");
        Test t=new Test();
        t.m1(10,'a');
        t.m2(10.2f,"ratna");
    }
}
```

instance methods

Ex :- static methods without parameters.

```
class Test
{
    static void ()
    {
```

```

        System.out.println("software solutions");
    }
    static void soft()
    {
        System.out.println("software solutions");
    }
    public static void main(String[] args)
    {

        () soft()
    }
}

```

Ex:-static methods with parameters

```

class Test
{
    static void m1(int i,char ch)
    {
        System.out.println(i+"-----"+ch);
    }
    static void m2(float f,String str)
    {
        System.out.println(f+"-----"+str);
    }
    public static void main(String[] args)
    {
        m1(10,'a');
        m2(10.2f,"ratna");
    }
}

```

} static methods

Ex :-while calling methods it is possible to provide the variables as a parameter values to the methods.

```

class Test
{
    void m1(int a,int b)
    {
        System.out.println(a);
        System.out.println(b);
    }
    void m2(boolean b)
    {
        System.out.println(b);
    }

    public static void main(String[] args)
    {
        int a=10;
        int b=20;
        boolean b=true;
        Test t=new Test();
        t.m1(a,b);
        t.m2(b);
    }
}

```

```

    }
}

```

m1()--→**calling** --→**m2()**----→**calling**-----→ **m3()**

m1()<-----after completion-**m2()**<-----after completion **m3()**

Ex:-calling of methods

```

class Test
{
    void m1()
    {
        m2();
        System.out.println("m1 ratan");
    }
    void m2()
    {
        m3(100);
        System.out.println("m2 ");
        M3(200);
    }
    void m3(int a)
    {
        System.out.println(a);
        System.out.println("m3 naresh");
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.m1();
    }
}

```

Ex :- methods with return type.

```

class Test
{
    static int add(int a,int b)
    {
        int c=a+b;
        return c;
    }
    static float mul(int a,int b)
    {
        int c=a*b;
        return c;
    }
    public static void main(String[] args)
    {
        int a=add(10,20);
        System.out.println(a);
        float b=mul(100,200);
        System.out.println(b);
    }
}

```

Ex :-for the java methods return type is mandatory otherwise the compilation will raise error return type required.

```

class Test
{

```

```
void m1()
{
    System.out.println("hi m1-method");
}
m2()
{
    System.out.println("hi m2-method");
}
```

```

        public static void main(String[] args)
        {
            Test t=new Test();
            t.m1();
            t.m2();
        }
    }

```

Ex :-Duplicate method signatures are not allowed in java language if we are declaring duplicate method signatures the compiler will raise compilation error m1() is already defined in test class Test

```

class Test
{
    void m1()
    {
        System.out.println("rattaiah");
    }
    void m1()
    {
        System.out.println("");
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.m1();
    }
}

```

Ex :-

- a. **Declaring the class inside the class is called inner classes concept java supports inner methods concept.**
- b. **Declaring the methods inside the methods inner methods concept java not supporting inner methods concept.**

```

class Test
{
    void m1()
    {
        void m2()
        {
            System.out.println("this ia a m2-static method");
        }
        System.out.println("this ia a m1-instance method");
    }
    public static void main(String[] args)
    {
        Test t1=new Test();
        t.m1();
    }
};

```

Stack Mechanism:-

- 1) Whenever we are executing the program stack memory is created.
- 2) The each and every method is called by JVM that method entries are stored in the stack memory and whatever the variables which are available within the method that variables are stored in

the stack.

- 3) If the method is completed the entry is automatically deleted from the stack means that variables are automatically deleted from the stack.
- 4) Based on the 1 & 2 points the local variables scope is only within the method.

Ex :-class Test

```
{
    void deposit(int accno)
    {
        System.out.println("money is deposited into    "+accno);
    }
    void withdraw(float amount)
    {
        System.out.println("money withdraw is over amount    "+amount);
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.deposit(111);
        t.withdraw(10000);
    }
}
```

Step 1:- One empty stack is created

Step 2:-

Whenever the JVM calls the main method the main method entry is stored in the stack memory. The stored entry is deleted whenever the main method is completed.

Step 3:-

Whenever the JVM is calling the deposit () the method entry is stored in the stack and local variables are store in the stack. The local variables are deleted whenever the JVM completes the Deposit () method execution. Hence the local variables scope is only within the method.

Step 4 :-

The deposit method is completed Then deposit () method is deleted from the stack.

Step 5 :-

Only main method call present in the stack.

Step 6:-

Whenever the JVM calls the withdraw method the entry is stored in the stack.

Step 7:-

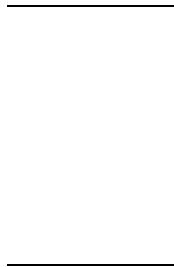
Whenever the Withdraw method is completed the entry is deleted from the stack.

Step 8:-

Whenever the main method is completed the main method is deleted from the stack.

Step 9:- the empty stack is deleted from the memory.

Step 1:-



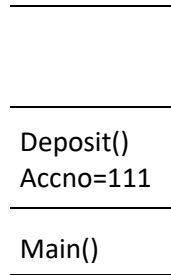
Empty stack

step 2 :-



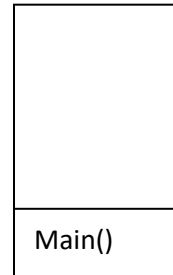
stack

step 3:-



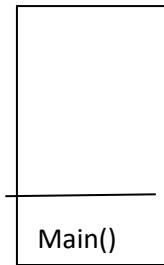
stack

step 4:-



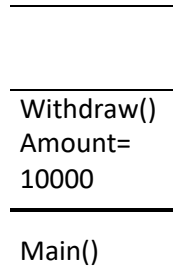
stack

Step 5:-



Stack

step 6:-



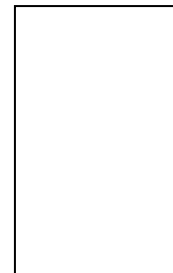
stack

step 7:-



stack

step 8 :-



empty stack

Ex :-we are getting StackOverflowError

class Test

```
{
    void m1()
    {
        System.out.println("rattaiah");
        m2();
    }
    void m2()
    {
        System.out.println(""); m1();
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.m1();
    }
}
```

Stack



Ex:-

```

class Operations
{
    int a;
    int b;
    void add()
    {
        System.out.println(a+b);
    }
};
class Test
{
    public static void main(String[] args)
    {
        Operations o1=new Operations();
        Operations o2=new Operations();
        o1.a=100;
        o1.b=200;
        o2.a=1000;
        o2.b=2000;
        o1.add();
        o2.add();
    }
};

```

Ex:-there are two types of instance methods

- 1) Accessor methods just used to read the data. To read the reading the data use getters methods.
- 2) Mutator methods used to store the data and modify the data for the storing of data use setters methods.

```

class Test
{
    String name;
    int id;

    //mutator method we are able to access and the data
    void setName(String name)
    {
        this.name=name;
    }
    void setId(int id)
    {
        this.id=id;
    }

    //accessor methods are used to read the data
    String getName()
    {
        return name;
    }
    int getId()

```

```

    {
        return id;
    }

    public static void main(String[] args)
    {
        Test t=new Test();
        t.setName("ratan");
        t.setId(12345);
        String name=t.getName();
        System.out.println(name);
        int id=t.getId();
        System.out.println(id);
    }
}

```

Local variable vs instance variable vs static variable :-

Heap memory	Stack memory
1) It is used to store the objects	1) It is used to store the function calls and local variables.
2) We are getting outOfMemoryError	2) If there is no memory in the stack to store method calls or local variables the JVM will throw the StackOverflowError.
3) Having more memory compared with the stack memory.	3) Stack memory is very less memory when compared with the heap memory
4) Heap memory is also known as public memory. This is applicable all the objects. This memory is shared by all threads.	4) Stack memory also known as private memory. This is applicable only for owners.
5) the objects are created in the heap memory with the help of new operator	Destroy when the method is completed. JVM is creating stack memory

Methods practice example:-

```

class Test
{
    int a=10;
    int b=20;

    static int c=30;
    static int d=40;

    void m1(char ch,String str)
    {
        System.out.println(ch);
        System.out.println(str);
    }
    void m2(int a,double d,boolean b)

```

```

    {
        System.out.println(a);
        System.out.println(d);
        System.out.println(b);
    }
    static void m3(String str)
    {
        System.out.println(str);
    }

    static void m4(char ch,char ch1)
    {
        System.out.println(ch);
        System.out.println(ch1);
    }

    public static void main(String[] args)
    {
        Test t=new Test();
        System.out.println(t.a);
        System.out.println(t.b);
        System.out.println(c);
        System.out.println(d);
        t.m1('a',"ratan");
        t.m2(100,10.8,true);
        m3("anu");
        m4('d','w');
    }
}

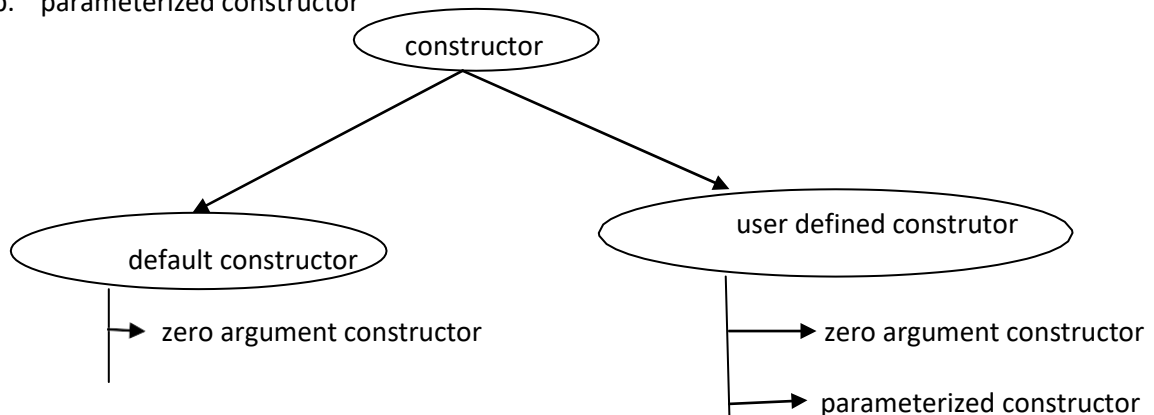
```

CONSTRUCTORS:-

- 1) Constructors are executed as part of the object creation.
- 2) If we want to perform any operation at the time of object creation the suitable place is constructor.
- 3) Inside the java programming the compiler is able to generate the constructor and user is able to declare the constructor. so the constructors are provided by compiler and user.

There are two types of constructors

- 1) Default Constructor.
 - a. Zero argument constructor.
- 2) User defined Constructor
 - a. zero argument constructor
 - b. parameterized constructor



Default Constructor:-

- 1) in the java programming if we are not providing any constructor in the class then compiler provides default constructor.
- 2) The default constructor is provided by the compiler at the time of compilation.
- 3) The default constructor provided by the compiler it is always zero argument constructors with empty implementation.
- 4) The compiler generated default constructor default constructor is executed by the JVM at the time of execution.

Ex:- Before compilation

```
class Test
{
    void good()
    {
        System.out.println("good girl");
    }
    public static void main(String[] args)
    {
        Test t=new Test();
        t.good();
    }
}
```

After compilation:-

```
class Test
{
    Test()
    {
        default
        constructor
        provided by
        compiler
    }
    void good()
    {
        System.out.println("good girl");
    }
    public static void main(String[] args)
    {
        Test t=new
        Test(); t.good();
    }
}
```

User defined constructors:-

Based on the user requirement user can provide zero argument constructor as well as parameterized constructor.

Rules to declare a constructor:-

- 1) Constructor name must be same as its class name.
- 2) Constructor doesn't have no explicit return type if we are providing return type we are getting any compilation error and we are not getting any runtime errors just that constructor treated as normal method.
- 3) In the class it is possible to provide any number of constructors.

user provided zero argument constructors.

```
class Test
{
    Test()
    {
        System.out.println("0-arg cons by user ");
    }
    public static void main(String[] args)
    {
        Test t=new Test();
    }
}
```

Compiler provided default constructor executed

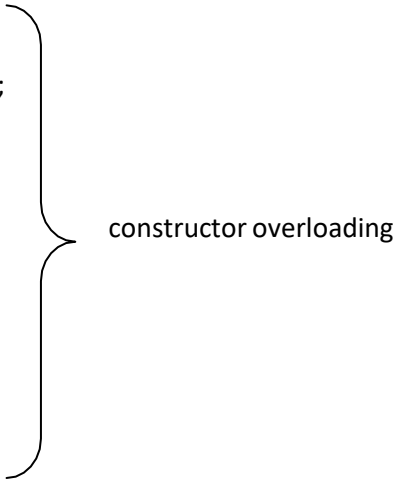
```
class Test
{
    /*Test()
    {
        }
    */
    public static void main(String[] args)
    {
        Test t=new Test();
    }
}
```

Ex 2:- user provided parameterized constructor is executed.

```
class Test
{
    Test(int i)
    {
        System.out.println(i);
    }
    void good()
    {
        System.out.println("good girl");
    }
    public static void main(String[] args)
    {
        Test t=new Test(10);
        t.good();
    }
}
```

Ex 3:-inside the class it is possible to take any number of constructors.

```
class Test
{
    Test()
    {
        System.out.println("this is zero argument constructor");
    }
    Test(int i)
    {
        System.out.println(i);
    }
    Test(int i,String str)
    {
        System.out.println(i);
        System.out.println(str);
    }
    public static void main(String[] args)
    {
        Test t1=new Test();
        Test t2=new Test(10);
        Test t3=new Test(100,"rattaiah");
    }
}
```



constructor overloading

Practice example:-

```
class Test
{
    //2-instance variables
    int a=1000;
    int b=2000;
    //2-static variables
    static int c=3000;
    static int d=4000;
    //2-instance methods
    void m1(int a,char ch)
    {
        System.out.println(a);
        System.out.println(ch);
    }
}
```



```

    }
    void m2(int a,int b,int c)
    {
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
    //2-static methods static
    void m3(String str)
    {
        System.out.println(str);
    }
    static void m4(String str1,String str2)
    {
        System.out.println(str1);
        System.out.println(str2);
    }
    //2-constructors
    Test(char ch,boolean b)
    {
        System.out.println(ch);
        System.out.println(b);
    }
    Test(int a)
    {
        System.out.println(a);
    }
    public static void main(String[] args)
    {
        //calling of constructors
        Test t1=new Test('s',true);
        Test t2=new Test(10);
        //calling of instance variables
        System.out.println(t1.a);
        System.out.println(t2.b);
        //calling of static variables
        System.out.println(c);
        System.out.println(d);
        //calling of instance methods
        t1.m1(100,'s');
        t1.m2(100,200,300);
        //calling of static methods
        m3("");
        m4("anu","dhanu");
    }
};

```

this keyword:-

This keyword is used to represent

1. Current class variables.
2. Current class methods.
3. Current class constructors.

Current class variables:-

Ex 1:-No need of this keyword

```
class Test
{
    int a=10; int b=20;
    void add(int i,int j)
    {
        System.out.println(a+b); System.out.println(i+j);
    }
    public static void main(String[] args)
    {
        Test t=new Test(); t.add(100,200);
    }
};
```

Ex 1:-No need of this keyword

```
class Test
{
    int a=10; int b=20;
    void add(int i,int j)
    {
        System.out.println(a+b); System.out.println(i+j);
    }
    public static void main(String[] args)
    {
        Test t=new Test(); t.add(100,200);
    }
};
```

Ex:-conversion of local variables into the instance/static variables

```
class Test
{
    static int i; int j;
    void values(int a,int b)
    {
        i=a; j=b;
    }
    void add()
    {
        System.out.println(i+j);
    }
    void mul()
    {
        System.out.println(i*j);
    }
    public static void main(String[] args)
    {
        Test t=new Test(); t.values(100,200); t.add();
        t.mul();
    }
}
```

Ex:- to convert local variables into the instance/static variables need of this keyword. class Test

```
{
    static int a; int b;
    void values(int a,int b)
    {
        this.a=a; this.b=b;
    }
    void add()
    {
        System.out.println(a+b);
    }
    void mul()
    {
        System.out.println(a*b);
    }
    public static void main(String[] args)
    {
        Test t=new Test(); t.values(100,200); t.add();
                                t.mul();
    }
}
```

Ex:- to call the current class methods we have to use this keyword The both examples gives same output(both examples are same)

```
class Test
{
    void m1()
    {
        this.m2();
        System.out.println("m1 method");
    }
    void m2()
    {
        System.out.println("m2 method");
    }
    public static void main(String[] args)
    {
        Test t=new Test(); t.m1();
    }
}
```

```
class Test
{
    void m1()
    {
        m2();
        System.out.println("m1 method");
    }
    void m2()
    {
        System.out.println("m2 method");
    }
    public static void main(String[] args)
    {
```

```

        Test t=new Test(); t.m1();
    }
}

```

Calling of current class constructors:-

Syntax :-

This(parameter -list);

Ex:- this();-----→to call zero argument constructor

Ex:- this(10)-----→to call one argument constructor

Ex:- In side the Constructors this keyword must be first statement in constructors(no compilation error)

```

class Test
{
    Test()
    {
        this(10);
        System.out.println("0 arg");
    }
    Test(int a)
    {
        this(10,20);
        System.out.println(a);
    }
    Test(int a,int b)
    {
        System.out.println(a+"-----"+b);
    }
    public static void main(String[] args)
    {
        Test t=new Test();
    }
}

```

Compilation error:-

```

class Test
{
    Test()
    {
        System.out.println("0 arg");
        this(10);
    }
    Test(int a)
    {
        System.out.println(a);
        this(10,20);
    }
    Test(int a,int b)
    {
        System.out.println(a+"-----"+b);
    }
    public static void main(String[] args)
    {
        Test t=new Test();
    }
}

```

```
    }  
}
```

note:-

1. inside the constructors constructor calling must be the first statement of the constructor otherwise the compiler will raise compilation error.
2. The above rule applicable only for constructors.

Ex:- conversion of local variables to the instance variables by using this keyword

```
import java.util.*;  
class Student  
{  
    String sname; int sno;  
  
    Student()  
    {  
        Scanner s=new Scanner(System.in);  
        System.out.println("enter sname:");  
        String sname=s.next();  
        this.sname=sname;  
        System.out.println("enter no");  
        int sno=s.nextInt(); this.sno=sno;  
    }  
    void display()  
    {  
        System.out.println("**student details*****");  
        System.out.println("student name:---"+sname);  
        System.out.println("student no:---"+sno);  
    }  
    public static void main(String[] args)  
    {  
        Student s=new Student();  
        s.display();  
    }  
}
```

Ex:-providing dynamic input to instance variables directly.

```
import java.util.*; class Student  
{  
    String sname; int sno;  
  
    Student()  
    {  
        Scanner s=new Scanner(System.in);  
        System.out.println("enter sname:");  
        sname=s.next();  
        System.out.println("enter sno");  
        sno=s.nextInt();  
    }  
  
    void display()  
    {  
        System.out.println("**student deatils**");  
        System.out.println("student sname"+sname);  
    }  
}
```

```
        System.out.println("student sno"+sno);  
    }  
    public static void main(String[] args)  
    {  
        Student s=new Student();  
    }  
}
```