Object in Java

An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tangible and intangible). The example of intangible object is banking system.

An object has three characteristics:

- o **state:** represents data (value) of an object.
- o **behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
- o **identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely.

For Example: Pen is an object. Its name is Reynolds, color is white etc. known as its state. It is used to write, so writing is its behavior.

Object is an instance of a class. Class is a template or blueprint from which objects are created. So object is the instance (result) of a class.

Object Definitions:

- o Object is a real world entity.
- o Object is a run time entity.
- o Object is an entity which has state and behavior.
- o Object is an instance of a class.

Class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

Class Contains Five elements:-

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

TOKENS:-Smallestindividual part in a java programiscalled Token. Itis possibletoprovideanynumber of spacesinbetweentwo tokens.

Ex:-ClassTest

```
{
        Publicstaticvoidmain(String[]args)
        {       inta=10;
                System.out.println("javatokens");
        }
}
```
Tokensare   □class,test,{,",[etc

Print()
vsPrintln()
Print ():-
        Print is used to print the statement into the console and the control is a vailable in the same
        line.
Ex:-   System.out.print("print method");

System.out.print("core java");

                Output:-print method core java
Println():-
        In the println statement Print is used to print the statement into the console and ln
        represent go to the newline now the control is available in the nextline.

Ex:-   System.out.println("println method");

System.out.println ("core java");

Output:println method

Core java

JAVACOMMENTS:-
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) Singleline Comments:-
        By using single line comments we are providing description
        about our program within a single line.
             Starts with.................... >//(double slash)
             Syntax:-   //description

   2) Multiline 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 stable.

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

Ex:-/*project name:-greenproject

teamsize:-   6 teamlead:- james

*/

Methods (behaviors):-

1) Methods are used to provide the business logic of the project.

2) The methods like a functions in C-language called functions, in java language is called methods.

3) 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 fallow 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:-

Ex:-

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

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   implementation/body statement-4

}

Ex:-

Void deposit()

{   defining a method

System.out.println("money deposit logic");   functionality of a method

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

}

There are two types of methods:-

Instance method

Static method

Ex :-instance methods without arguments.

```
class Test
{
      voiddisplay()
      {
            System.out.println ("display method behavior");
      }
      publicstaticvoidmain (String[]args)
      {
            Testt=newTest ();
            t.display ();
      }
}
```

Ex :-instance methods with arguments.

```
classTest
{
        void m1(int i,char ch)
        {
                System.out.println (i+" ------ "+ch);
        }                                              instancemethods
        Void m2 (float f, String str)
        {
                System.out.println (f+" ----- "+str);
        }
        Public static void main(String[] args)
        {
                System.out.println ("programstarts");
                Testt=newTest ();
                t.m1(10,'a');
                t.m2(10.2f,"student");
        }
}
```

Ex :-static methods without arguments.

```
classTest
{
        void display()
        {
                System.out.println ("display method behavior");
        }
        public static void main (String[]args)
        {
                display ();
        }
}
```

Ex:-static methods with parameters

```
        classTest
        {
                static void  m1(int i,char ch)
                {
                        System.out.println (i+" ------ "+ch);
                }                                      //static methods
                Static void  m2(float f,String str)
                {
                        System.out.println (f+" ----- "+str);
                }
```

```
        public static void main(String[] args)
        {
                m1 (10,'a');
                m2 (10.2f,"student");
        }
}
```

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 method");
```

```
}
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);
}}
```

- **For the java methods return type is mandatory otherwise the compilation will raise error return type required.**
- **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**
- **Declaring the class inside the class is called inner classes concept supported by java. Declaring a method inside a method called inner methods concept is not supporting in java.**

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);

}

}

Steps Executed when each program interpreted by JVM:

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.
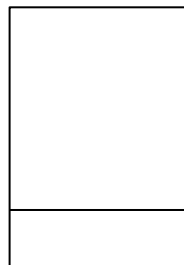
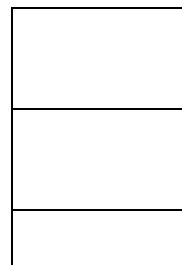**Step 9:-** the empty stack is deleted from the memory.
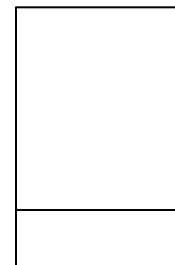
Step 1	**Step 2**	**Step 3**	**Step 4**
Empty Stack	**Stack**	**Stack**	**Stack**
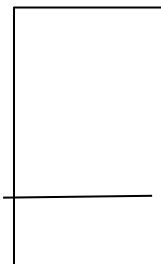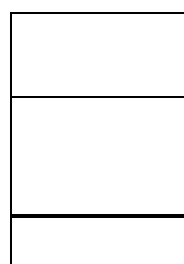
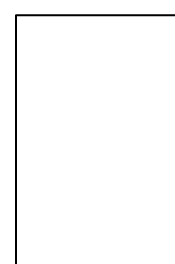| **Step 5**	**Step 6**	**Step 7**	**Step 8**
Stack	**Stack**	**Stack**	**Empty Stack**

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("Java"); 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
It is used to store Objects.	It is used to store the function calls and local variables.
If there is no memory to create new object it will generate outOfMemoryError.	If the there is no memory in the stack to store method calls or local variables the JVM will throw the StackOverflowError.
Having more memory compared with the stack memory.	Stack memory is very less memory when compared with the heap memory.
Heap memory is also known as public memory. This is applicable to all the objects. This memory is shared by all threads.	Stack memory also known as private Memory. This is applicable only for owners.
The objects are created in the heap memory with the help of new operator.	JVM will create this statck memory when method called and Destroy when the method is completed.

Complete example on methods:-

```
class Test
{
int a=10;
int b=20;
static int c=30;
static int d=40;
void m1(char ch,Stringstr)
{
System.out.println(ch); System.out.println(str);
}
void m2(inta,doubled,boolean b)
{
System.out.println(a); System.out.println(d); System.out.println(b);
}
static void m3(String str)
{
```

```java
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',"student 1");
t.m2(100,10.8,true);
m3("student");
m4('d','w');
}
}
```

CONSTRUCTORS:-

Constructor is also one type of method which is having same name as class-name and which does not have any return type and it is called whenever the object is created.

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 constructors.

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 withempty implementation.

4) The compiler generated default constructor that 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()           Compiler provided
         {                 default constructor
         }
        */
        public static void main (String[] args)
        {
        Test t=new Test ();
        }
}
```

Ex2:- user provided parameterized constructor is executed.

```
class Test
{
Test(int i)
{
System.out.println (i);
}
void good()
```

```java
{
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.

```java
class Test
{
Test()
{
System.out.println ("this is zero argument constructor");
}
Test(int i)
{
System.out.println (i); //constructor overloading
}
Test(inti,Stringstr)
{
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,"student");
}
}
```

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(inta,inta,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)
```

```java
{
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("student");
m4("marks", "rank");
}
};
```
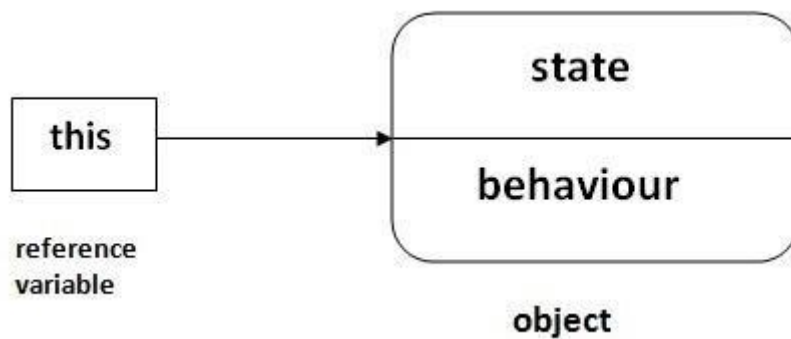
This Keyword:-

There can be a lot of usage of **java this keyword**. In java, this is a **reference variable** that refers to the current object.

Usage of java this keyword

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.

2. this can be used to invoke current class method (implicitly)

3. this() can be used to invoke current class constructor.

4. this can be passed as an argument in the method call.

5. this can be passed as argument in the constructor call.

6. this can be used to return the current class instance from the method.

Suggestion: If you are beginner to java, lookup only three usage of 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;

```java
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);
}
};
```

Note: - this keyword is required

```java
class Test
{
int a=10;
int b=20;
void add (int a, int b)
{
System.out.println (a+b);
System.out.println (this.a+this.b);
}
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.

```java
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 mehods we have to use this keyword The both examples gives same output(both examples are same)

```java
class Test
{
void m1()
{
```

```java
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:- Inside 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();
}}
```

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

```
importjava.util.*;
class Student
{
String sname;
intsno;
Student()
{
Scanner s=new Scanner(System.in);
System.out.println("enter sname:");
String sname=s.next();
this.sname=sname;
System.out.println("enter no");
intsno=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.

```java
importjava.util.*;

class Student

{

String sname;

intsno;

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();

}

}

}
```

The **static keyword** in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be:

1. variable (also known as class variable)

2. method (also known as class method)

3. block

4. nested class

1) Java static variable

If you declare any variable as static, it is known static variable.

- o The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees, college name of students etc.

- o The static variable gets memory only once in class area at the time of class loading.

Advantage of static variable

It makes your program **memory efficient** (i.e it saves memory).

Understanding problem without static variable

```
class Student{
    int rollno;
    String name;
    String college="ITS";
}
```

Suppose there are 500 students in my college, now all instance data members will get memory each time when object is created.All student have its unique rollno and name so instance data member is good.Here, college refers to the common property of all objects.If we make it static,this field will get memory only once.

Example of static variable

//Program of static variable 2.

```java
class Student8{
  int rollno;
  String name;
  static String college ="JAVA";
  Student8(int r,String n){
  rollno = r;
  name = n;
   }
  void display (){System.out.println(rollno+" "+name+" "+college);}
  public static void main(String args[]){
  Student8 s1 = new Student8(111,"ABC");
  Student8 s2 = new Student8(222,"CDE");
  s1.display();
  s2.display();
  }
  }
```

2) Java static method

If you apply static keyword with any method, it is known as static method.

- o A static method belongs to the class rather than object of a class.

- o A static method can be invoked without the need for creating an instance of a class.

- o static method can access static data member and can change the value of it.

Example of static method

```java
//Program of changing the common property of all objects(static field).
class Student9{
    int rollno;
    String name;
    static String college = "JAVA";
    static void change(){
    college = "Vignan";
    }
    Student9(int r, String n){
    rollno = r;
    name = n;
    }
    void display (){System.out.println(rollno+" "+name+" "+college);}
    public static void main(String args[]){
    Student9.change();
    Student9 s1 = new Student9 (111,"ABC");
    Student9 s2 = new Student9 (222,"CDE");
    Student9 s3 = new Student9 (333,"EFG");
    s1.display();
    s2.display();
    s3.display();
    }  }
```

Restrictions for static method

There are two main restrictions for the static method. They are:

1. The static method can not use non static data member or call non-static method direc

2. this and super cannot be used in static context.

3) Java static block

o Is used to initialize the static data member.

o It is executed before main method at the time of classloading.

Example of static block

```
class A2{

static{System.out.println("static block is invoked");}

public static void main(String args[]){

 System.out.println("Hello main");

}

}
```

Instance Blocks:-

➢ The instance blocks are executed irrespective of any condition.

➢ The instance blocks and instance variables are executed before constructor execution. If you are giving a chance to the constructors then only instance blocks are executed.

➢ In the class it is possible to take the any number of instance blocks but the execution order is top to bottom.

➢ Instance blocks are executed based on the object creation. If we are creating ten objects ten times instance blocks will be executed.

➢ If the source file contains inheritance concept at that situation first parent class instance block will be executed then child class instance blocks will be executed.

Static Blocks:-

➢ The static blocks are executed at the time of class loading into the memory.

➢ Whenever we are using (java className) the class is loaded into the memory at that moment the static blocks are loaded.

➢ The static blocks are executed at only one time for each and every class loading. But the instance blocks are executed based number of constructor's execution.

➢ Without using main method it is possible to print some statements into the console in java language with the help of static blocks. but this rule is applicable only up to 1.5 version if we are using higher versions if we want to execute static block the main method is mandatory.

➢ In the higher version the static blocks are executed only if the class contains main method.

➢ Based on the above reason we can say in the higher version it is not possible to print some statements into the console without using main method.

➢ Whenever we are loading child class into the memory then automatically parent class is loaded hence the parent class static block is executed first and then child class static blocks are loaded.