JAVA introduction:-

Author : James Gosling
Vendor : initially by Sun Micro System after it owned by Oracle Corporation in January 27, 2010.
Project name : Green Project
Type : open source & free software
Initial Name : OAK language
Present Name : java
Extensions : .java & .class & .jar
Initial version : jdk 1.0 (java development kit)
Present version :java 21 2023
Operating System :multi Operating System
Implementation Lang : c, cpp……
Symbol : coffee cup with saucer
Objective : To develop web applications
SUN : Stanford Universally Network
Slogan/Motto : WORA (write once run anywhere)
Ex:-

```
Class Test
{
Public static void main (String [] args)
{
System.out.println (“welcome to java language”);
}
}
```

Compilation :- javac FileName.java
Execution :- java Class Name
Output :- welcome to java language

Importance of core java:-

According to the Oracle Corporation 3 billion devices run on the java language only.

1) Java is used to develop Desktop Applications such as MediaPlayer, Antivirus etc.
2) Java is used to Develop Web Applications such as saiganapthi.edu.in, irctc.co.in , makemytrip.com etc.
3) Java is used to Develop Enterprise Application such as banking applications.
4) Java is used to Develop Mobile Applications.
5) Java is used to Develop Embedded System.
6) Java is used to Develop SmartCards.
7) Java is used to Develop Robotics.
8) Java is used to Develop Games etc.

JAVA VERSIONS:-

Java Alpha & beta	:	1995	
JDK 1.0	:	1996	Codename **Java 1.**
JDK1.1	:	1997	codename **Java 1.**
J2SE 1.2	:	1998	Codename **Playground**.
J2SE 1.3	:	2000	Codename **Kestrel.**
J2SE 1.4	:	2002	Codename **Merlin.**
J2SE 1.5	:	2004	Codename **Tiger.**
JAVA SE 6	:	2006	Codename **Mustang.**
JAVA SE 7	:	2011	Codename **Dolphin.**
Java SE 8	:	2014	Codename **Spider.**
Java SE 9	:	2017	Codename Noname
Java SE 10	:	2018	Codename Noname
Java SE 11	:	2018	Codename Noname
Java SE 12	:	2019	Codename Noname
Java SE 13	:	2019	Codename Noname
Java SE 14	:	2020	Codename Noname
Java SE 15	:	2020	Codename Noname
Java SE 16	:	2021	Codename Noname
Java SE 17	:	2021	Codename Noname
Java SE 18	:	2022	Codename Noname
Java SE 19	:	2022	Codename Noname
Java SE 20	:	2023	Codename Noname
Java SE 21	:	2023	Codename Noname
Java SE 22	:	2024	Codename Noname

As per the sun micro system standard the java language is divided into three types.

1) **J2SE/JSE(java 2 standard edition)**
2) **J2EE/JEE(java 2 enterprise edition)**
3) **J2ME/JME(java 2 micro edition)**

J2SE:-

By using j2se we are able to develop the standalone applications.
Ex:- notepad, WordPad, paint, Google Talk…..... etc

Standalone applications:-

1) Standalone applications are the java applications which don't need the client server Architecture.
2) The standalone applications applicable for the only one desktop hence it is called desktop applications or window-based applications.
3) For the standalone applications doesn't need internet connections.
4) It is a local application it doesn't need any other external application support.
5) This type of the applications we can launch by using the command line or by using the executable jar.

J2EE:-

Client Database

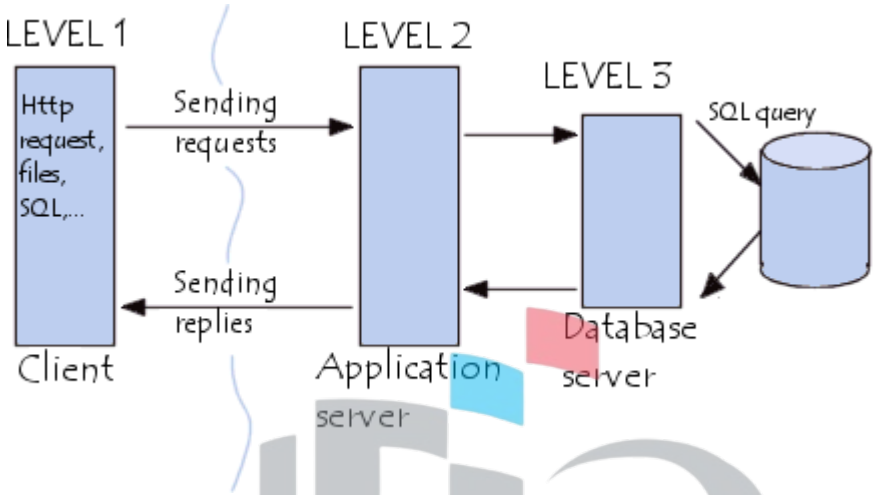
By using j2ee we are able to develop the web based applications. Ex:- Gmail, yahoo mail, bank, reservation..... etc

Web-applications:-

1) Web applications are the java applications which needs client and server concept.
2) Web applications must need the internet connections to access the application.

3) The application which is present in the internet is called the web application.
4) Web application can be launched by using HTTP driven. HTTP request sent to the Servlet present in the server side.

Web-application architecture:-



Client:-

The person who is sending the request is called client. All web browsers come under the clients.
Ex:- InternetExploral, MozillaFrefox, opera.....etc

Server:-
The server contains the applications. The main purpose of the server is
a. It will contain the application.
b. Take the request from the client.
c. Based on the taken request it will identify the project resource and execute that project resource.
d. By executing the project some response will be generated that response is dispatched to the client browser.
Ex: - Tomcat, GlassFish, WebLogic, JBOSS, WebSphere..... etc

DataBase:-
DataBase is used to store the details like client details, application details, and registration details……etc.
Ex: - Oracle, MySql .....etc

J2ME:-
By using j2me we are able to develop the applications that applications only run on mobile devices.

Difference Between c , C++ and Java

1) Language->C is a procedural language,
c++ is a object oriented language .
Java is a pure object oriented language.

2) Header Files->In C we use stdio.h header file .
In C++ we use iostream.h,conio.h headerfile but Java does not support any header files.

3) Platform Independent->C & C++ both are platform dependent that means you can't run the execute code in any other operating system.
Java is a platform independent language.

4) Pointers->Pointers are used in C & C++ language .Java will not support for pointers.

5) Operator overloading->Operator overloading concept is not used in Java but in C++ we use operator overloading.

6) Multiple Inheritance->We use multiple inheritance in C++ not in Java .In Java we use Interface instead of multiple inheritance.

7) Translator->Java uses compiler and interpreter but in C & C++ uses compiler only.

8)Web Application->Internet programming like Frame,Applet is used in Java not in C,C++.

9)Operator->Dot (.)operator is used instead of scope resolution operator.

10) Package->We can create our own package in Java(set of classes) but not in c and c++.

OOP Concepts In Java

Object means real-world things such as pen, paper, chair, etc. OOP is a technique that helps in designing a program more effectively using classes and objects. It provides the following concepts.

Class
Object
Inheritance
Polymorphism
Abstraction
Encapsulation
Advantages

The following are the advantages of OOP over procedure-oriented programming languages:

- makes development of programs easier; in procedure-oriented languages it's difficult to manage the coding of the program when the program increases in size.
- provides a way to interact with real-world data more effectively.
- allows development of solutions for real-world problems.
- provides hiding of codes but in a procedure-oriented language the data can be accessed from anywhere.

There are four main features of OOP; they are:

1. **Encapsulation**
2. **Inheritance**
3. **Polymorphism**
4. **Abstraction**

1. Encapsulation

Encapsulation means binding all methods and classes in a single class. Encapsulation is the technique of making the fields in a class private and providing access to the fields via public methods. If a field is declared private then it cannot be accessed by anyone outside the class, thereby hiding the fields within the class. The main benefit of encapsulation is the ability to modify our implemented code without breaking the code of others who use our code.
Encapsulation provides maintainability, flexibility and extensibility of our code.

Advantages

The following are a few advantages of using Encapsulation:

1. Provides the ability to change one part of code without affecting another part of code.
2. Controls the access of the user interface.
3. With new requirements it is easy to change encapsulated code.
4. Helps to write immutable classes in Java that are beneficial in multi-threading environments.
5. Encapsulation in Java makes unit testing easy.
6. Reduces the coupling of modules since all pieces of the same type are encapsulated in one place.

2. Inheritance

The main feature of Inheritance is code re-usability. So when making a new class we can use a previously written class and further extend it. In Java, classes may inherit or acquire the properties and methods of other classes. A class derived from another class is called a subclass, whereas the class from which a subclass is derived is called a super class. A subclass can have only one super class, whereas a super class may have one or more sub-classes.

Example:

```
class StudentRec
  {
    //GET STUDENT RECORD.....
    String name;
    int rollno;
    int get(String n, int r)
      {
        name=n; rollno=r; return(0);
      }
    void showDetails()
      {
        System.out.println("Name : "+name);
      }
  }
class InClassDemo extends StudentRec
  {
    public static void main(String args[])
      {
        //CREATE OBJECT OF STUDENT RECORD CLASS
        StudentRec studObj = new StudentRec();
        studObj.get("SANDEEP SHARMA", 92);
        studObj.showDetails();
      }
    void displayDetails()
      {
        System.out.println("Sample Info Display");
      }
  }
```

3. Polymorphism:

In Core Java Polymorphism is an easy concept to understand. Polymorphism in Greek is a combination of poly, which means many and morphism which means forms. It refers to the object's ability to be Polymorphic depending on its type.

There are two types of Polymorphism available in Java.

1) Static Polymorphism

2) Dynamic Polymorphism

Let's discuss *Static Polymorphism*. It's compile-time Polymorphism. We have two important concepts in Polymorphism, i.e Method Overloading and Method Overriding.

1. Method Overloading

In Java method overloading we define two or more methods with the same name but different parameters. The methods are said to be overloaded, and the process is referred to as method overloading.
Example for Method overloading

The following example program will make you understand Method Overloading:

```
class Sub
  {
    void add(int tamil, int english)
      {
        System.out.println("The total of tamil and english is "+(tamil+english));
      }
    void add(int tamil,int english,int maths)
      {
        System.out.println("The total of tamilenglish and maths is "+(tamil+english+maths));
      }
  }
public class MetOvlDemo
  {
    public static void main(String arg[])
      {
        //create Subjects class object
        Sub sb=new Sub();
        // we have to call add() method by passing 2 values
        sb.add(90, 80);

//here also we are calling add() method by passing 3 values, So the 3 arguments (parameters) met
hod will getexecute.
        sb.add(95,85,100);
      }
  }
```

2. Method Overriding

Now we will discuss what *dynamic polymorphism* is. It's run time polymorphism. We can also call it Method Overriding. In a class hierarchy, when a method in a sub class has the same name and type signature as a method in its superclass, then the method in the subclass is said to override the method in the superclass. This feature is called method overriding.

Example

```
class MathsSqr1
  {
    void calculate(double price)
      {
        System.out.println("Sqare value "+(price*price));
      }
  }
class MathsSqr2 extends MathsSqr1
  {
    void calculate(double price)
      {
        System.out.println("Sqare value "+(Math.sqrt(price)));
      }
  }
public class Metovrr
  {
    public static void main(String arg[])
      {
        MathsSqr2 msd=new MathsSqr2();
        msd.calculate(25);
      }
  }
```

4. Abstraction

When we hide the unnecessary detail and defining the useful (relevant) detail, then the procedure is said to be *abstraction*. An interface or abstract class is something that is not concrete,something that is incomplete. Another way of providing a simple explanation is to use a more complex system as an example. One does not want to understand how an engine works. Similarly one does not need to understand the internal implementation of the software objects.

Abstraction in Java is done by using an interface and abstract class in Java. In order to use an interface or abstract class we need to explain the methods of an interface or abstract class in a sub-class.

Abstraction Example: Engine, Driving.

Main advantage: the user gets data according to their needs but they don't need to use unnecessary data. The following example will show Abstraction.

Example

```
public class Abstraction
  {
    private int accNO;
    private String custName;
    private float accBlnc;
    private float profit;
    private float loan;
    public void dislayClerkInfo()
      {
        System.out.println("Accout number "+accNo);
        System.out.println("Customer name "+custName);
        System.out.println("Account Balance "+accBlnc);
      }
  }
```

Note: This class only defines the structure but not any implementation of them.

Where it is used?

According to Sun, 3 billion devices run java. There are many devices where Java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus etc.
2. Web Applications such as irctc.co.in etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games etc.

Types of Java Applications

There are mainly 4 types of applications that can be created using java programming:

1) Standalone Application

It is also known as desktop application or window-based application. An application that we need to install on every machine such as media player, antivirus etc. AWT and Swing are used injava for creating standalone applications.

2) Web Application

An application that runs on the server side and creates dynamic page, is called web application. Currently, servlet, jsp, struts; jsf etc. technologies are used for creating web applications in java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications etc. It has the advantage of high level security, load balancing and clustering. In java, EJB is used for creating enterprise applications.

4) Mobile Application

An application that is created for mobile devices. Currently Android and Java ME are used for creating mobile applications.

JAVA Features or Buzzwords:-

1. Simple
2. Object Oriented
3. Platform Independent
4. Architectural Neutral
5. Portable
6. Robust
7. Secure
8. Dynamic
9. Distributed
10. Multithread
11. Interpretive
12. High Performance

1. Simple:-

Java is a simple programming language because:

Java technology has eliminated all the difficult and confusion oriented concepts like pointers, multiple inheritance in the java language.

The c,cpp syntaxes easy to understand and easy to write. Java maintains C and CPP syntax mainly hence java is simple language.

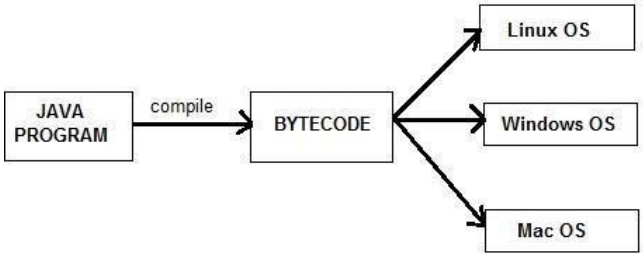
Java tech takes less time to compile and execute the program.

2. Object Oriented:-

Java is object oriented technology because to represent total data in the form of object. By using object reference we are calling all the methods, variables which is present in that class.

3. Platform Independent:-

Compile the Java program on one OS (operating system) that compiled file can execute in any OS (operating system) is called Platform Independent Nature. The java is platform independent language. The java applications allows its applications compilation one operating system that compiled (.class) files can be executed in any Operating system.



4.Architectural Neutral:-

Java tech applications compiled in one Architecture (hardware --- RAM, Hard Disk) and that Compiled program runs on any hardware architecture (hardware) is called Architectural Neutral.

5. Portable:-

In Java technology the applications are compiled and executed in any OS(operating system)
and any Architecture (hardware) hence we can say java is a portable language.

6. Robust:-

Any technology if it is good at two main areas it is said to be ROBUST

1 Exception Handling
2 Memory Allocations

JAVA is Robust because

a. JAVA is having very good predefined Exception Handling mechanism whenever we are getting exception we are having meaning full information.
b. JAVA is having very good memory management system that is Dynamic Memory (at runtime the memory is allocated) Allocation which allocates and deallocates memory for objects at runtime.

7. Secure:-

To provide implicit security Java provide one component inside JVM called Security Manager.
To provide explicit security for the Java applications we are having very good predefined library in the form of java.Security.package.
Web security for web applications we are having JAAS(Java Authentication and Authorization Services) for distributed applications.

8. Dynamic:-

Java is dynamic technology it follows dynamic memory allocation(at runtime the memory is allocated) and dynamic loading to perform the operations.

9. Distributed:-

By using JAVA technology we are preparing standalone applications and Distributed applications.
Standalone applications are java applications it doesn't need client server architecture.
web applications are java applications it need client server architecture.
Distributed applications are the applications the project code is distributed in multiple number of jvm's.

10. Multithreaded: -

Thread is a light weight process and a small task in large program.

If any task allows executing single thread at a time such type of technologies is called single threaded technology.

If any technology allows creating and executing more than one thread called as Multithreaded technology called JAVA.

11. Interpretive:-

JAVA tech is both Interpretive and Completive by using Interpretation we are converting source code into byte code and the interpretator is a part of JVM.

12. High Performance:-

If any technology having features like Robust, Security, Platform Independent, Dynamic and so on then that technology is high performance.

Install the software and set the path :-

Download the software from internet based on your operating system. The software is different from 32-bit operating and 64-bit operating system.

To download the software open the fallowing web site.

http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

for 32-bit operating system please click on

Windows x86 :- 32- bit operating system

for 64-bit operating system please click on

Windows x64 :- 64-bit operating system

After installing the software the java folder is available in the fallowing location

Local Disk c: ------program Files--------java --- jdk (java development kit), jre (java runtime environment)

To check whether the java is installed in your system or not go to the command prompt. To open the command prompt

Start ----------run------open: cmd ---- ok

Command prompt is opened.

In the command prompt type: - javac

'Javac' is not recognized is an internal or external command, operable program or batch file.

Whenever we are getting above information at that moment the java is installed but the java is not working properly.

C :/> javac

Whenever we are typing javac command on the command prompt

1) Operating system will pickup javac command search it in the internal operating system calls. The javac not available in the internal command list.

2) Then operating system goes to environmental variables and check is there any path is sets or not. up to now we are not setting any path. So operating system don't know anything about javac command Because of this reason we are getting error message.

Hence we have to environmental variables. The main aim of the setting environmental variable is to make available the fallowing commands javac, java, javap (softwares) to the operating system.

To set the environmental variable:-

My Computer (right click on that) ---->properties----->Advanced--->Environment Variables >

User variables--new---->variable name : Path

Variable value: C:\programfiles\java\jdk x.x._x\bin;.;

.....ok.....ok

Now the java is working well in your system. Open the command prompt to check once

C:>javac --------now list of commands will be displayed

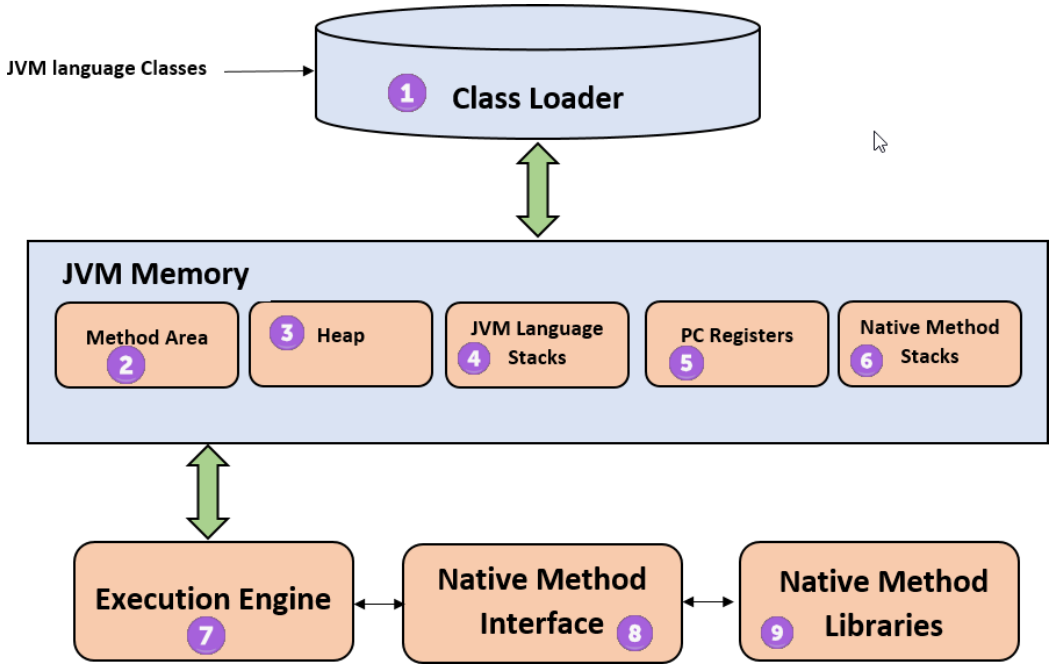
JVM (Java Virtual Machine)

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed. JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

What is JVM?

It is:

1. A specification where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Sun and other companies.
2. An implementation Its implementation is known as JRE (Java Runtime Environment).
3. Runtime Instance Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.



What it does

The JVM performs following operation:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

Internal Architecture of JVM

1) Classloader

Classloader is a subsystem of JVM that is used to load class files.

2) Class(Method) Area

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

3) Heap

It is the runtime data area in which objects are allocated.

4) Stack

Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return. Each thread has a private JVM stack, created at the same time as thread. A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

5) Program Counter Register

PC (program counter) registers. It contains the address of the Java virtual machine instruction currently being executed.

6) Native Method Stack

It contains all the native methods used in the application.

7) Execution Engine

It contains:

1) A virtual processor

2) **Interpreter:** Read bytecode stream then execute the instructions

3) Just-In-Time(JIT) compiler: It is used to improve the performance.JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here the term? Compiler? Refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU

Structure of program:-

In this page, we will learn how to write the simple program of java. We can write a simple hello java program easily after installing the JDK.

To create a simple java program, you need to create a class that contains main method. Let's understand the requirement first.

Creating hello java example

Let's create the hello java program:

```
class Simple {
   public static void main (String args[]){
   System.out.println ("Hello Java");
   }
}
```

save this file as Simple.java

To compile: javac Simple.java

To execute: j a v a S i m p l e

Output: Hello Java

Understanding first java program

Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- **static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.
- **void** is the return type of the method, it means it doesn't return any value.
- **main** represents startup of the program.
- **String[]** args is used for command line argument. We will learn it later.
- **System.out.println()** is used print statement.

How many ways can we write a java program

There are many ways to write a java program. The modifications that can be done in a java program are given below:

1) By changing sequence of the modifiers, method prototype is not changed.

Let's see the simple code of main method.

1. static public void main(String args[])

2) subscript notation in java array can be used after type, before variable or after variable.

Let's see the different codes to write the main method.

1. public static void main(String[] args)
2. public static void main(String []args)
3. public static void main(String args[])

3) You can provide var-args support to main method by passing 3 ellipses (dots)

Let's see the simple code of using var-args in main method.

public static void main(String... args)

4) Having semicolon at the end of class in java is optional.

Let's see the simple code.

```
class A{
static public void main(String... args){
System.out.println ("hello java4");
}
};
```

Valid java main method signature

1. public static void main(String[] args)
2. public static void main(String []args)
3. public static void main(String args[])
4. public static void main(String... args)

5. static public void main(String[] args)
6. public static final void main(String[] args)
7. final public static void main(String[] args)
8. final strictfp public static void main(String[] args)

Invalid java main method signature

1. public void main(String[] args)
2. static void main(String[] args)
3. public void static main(String[] args)
4. abstract public static void main(String[] args)

Types of variables:-

• Variables are used to store the values. By storing that values we are achieveing 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 but these variables are methods can't be accessed inside static content.
3. We can declare instance variables without initializing a value to the variable also for those variables it will take default values of data type.
4. Instance variables can be accessed by using object and using this operator.
5.

```
class Test
{
int a=10; // instance variable
int b=20; // instance variable
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.
- ➢ We can declare static variables without initializing a value to the variable also for those variables it will take default values of data type.
- ➢ These variables can be accessed by using class name and also using object name.

```
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... args)
{
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

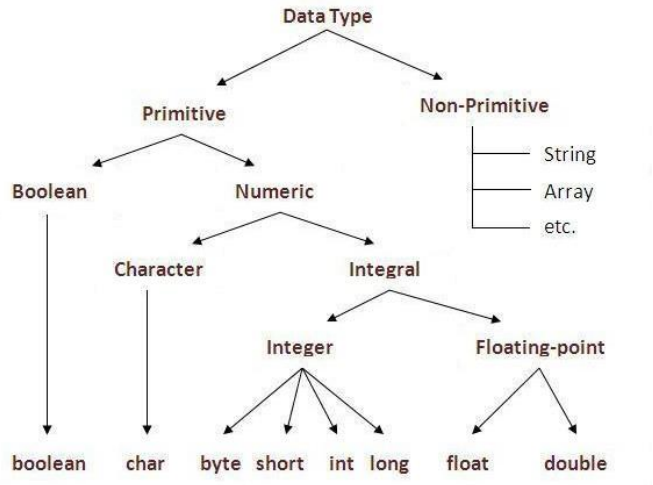
we have to provide the default values.

```
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);

Data Types in Java



Data types represent the different values to be stored in the variable. In java, there are two types of data types:

- Primitive data types
- Non-primitive data types

Why char uses 2 byte in java and what is \u0000 ?

It is because java uses Unicode system than ASCII code system. The \u0000 is the lowest range of Unicode system. To get detail explanation about Unicode visit next page.

Datatype	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

Java Variable Example: Add Two Numbers

```
1. class Simple{
2. public static void main(String[] args){
```

3. **int** a=10;
4. **int** b=10;
5. **int** c=a+b;
6. System.out.println(c);
7. }}

Java Variable Example: Widening

```
class Simple{
public static void main(String[] args){
int a=10;
float f=a;
System.out.println(a);
System.out.println(f);
}}
```

Output:

10
10.0

Java Variable Example: Narrowing (Typecasting)

```
class Simple{
public static void main(String[] args){
float f=10.5f;
//int a=f;//Compile time error
int a=(int)f;
System.out.println(f);
System.out.println(a);
}
}
```

Output:

10.5
10

Java Variable Example: Overflow

```
class Simple{

public static void main(String[] args){

//Overflow
int a=130;

byte b=(byte)a;

System.out.println(a);

System.out.println(b);

}

}
```

Output:

```
130
-126
```

Java Variable Example: Adding Lower Type

```
class Simple{

public static void main(String[] args){

byte a=10;

byte b=10;

//byte c=a+b;//Compile Time Error: because a+b=20 will be int

byte c=(byte)(a+b);

System.out.println(c);

}
}
```

Java Keywords

Keywords are predefined, reserved words used in Java programming that have special meanings to the compiler. For example:

```
int score;
```

Here, `int` is a keyword. It indicates that the variable `score` is of integer type (32-bit signed two's complement integer).

You cannot use keywords like `int`, `for`, `class` etc as variable name (or identifiers) as they are part of the Java programming language syntax. Here's the complete list of all keywords in Java programming.

Keywords in Java				
abstract	default	if	private	this
assert	do	implements	protected	throw
boolean	double	import	public	throws
break	else	instanceof	return	transient
byte	enum	int	short	try
case	extends	interface	static	void
catch	final	long	strictfp	volatile
char	finally	native	super	while
class	float	new	switch	
continue	for	package	synchronized	

Beside these keywords, you cannot also use true, false and null as identifiers as they are literals.

Java identifiers

Identifiers are the name given to variables, classes, methods etc. Consider the above code;

```
int score;
```

Here, score is a variable (an identifier). You cannot use keywords as variable name. It's because keywords have predefined meaning. For example,

```
int float;
```

The above code is wrong. It's because float is a keyword and cannot be used as a variable name.

Rules for Naming an Identifier

- Identifier cannot be a keyword.
- Identifiers are case-sensitive.
- It can have a sequence of letters and digits. However, it must begin with a letter, $ or _. The first letter of an identifier cannot be a digit.
- It's convention to start an identifier with a letter rather and $ or _.
- Whitespaces are not allowed.
- Similarly, you cannot use symbols such as @, #, and so on.

Here are some valid identifiers:

- score
- level
- highestScore

- number1

Here are some invalid identifiers:

- class
- float
- 1number
- highest Score
- @pple

Literals in Java

Literal: Any constant value which can be assigned to the variable is called as literal/constant.

```
// Here 100 is a constant/literal.
int x = 100;
```

Integral literals

For Integral data types (bye, short, int, long), we can specify literals in 4 ways:-

1. **Decimal literals (Base 10):** In this form the allowed digits are 0-9.
2. int x = 101;
3. **Octal literals (Base 8):** In this form the allowed digits are 0-7.
4. // the octal number should be prefix with 0.
   int x = 0146;
5. **Hexa-decimal literals (Base 16):** In this form the allowed digits are 0-9 and characters are a-f. We can use both uppercase and lowercase characters. As we know that java is a case-sensitive programming language but here java is not case-sensitive.
6. // The hexa-decimal number should be prefix
   // with 0X or 0x.
   int x = 0X123Face;

7. **Binary literals:** From 1.7 onward we can specify literals value even in binary form also, allowed digits are 0 and 1. Literals value should be prefixed with 0b or 0B.
8. int x = 0b1111;

Example:

```
// Java program to illustrate the application of Integer literals

public class Test {
   public static void main(String[] args)
   {
         int a = 101; // decimal-form literal
         int b = 0100; // octal-form literal
         int c = 0xFace; // Hexa-decimal form literal
         int d = 0b1111; // Binary literal
         System.out.println(a);
         System.out.println(b);
         System.out.println(c);
         System.out.println(d);
   }
}
```

Output:

```
101
64
64206
15
```

NOTE: By default, every literal is of int type, we can specify explicitly as long type by suffixed with l or L. There is no way to specify byte and short literals explicitly but indirectly we can specify. Whenever we are assigning integral literal to the byte variable and if the value within the range of byte then the compiler treats it automatically as byte literals.

Floating-Point literal

For Floating-point data types, we can specify literals in only decimal form and we cant specify in octal and Hexa-decimal forms.

Decimal literals (Base 10): In this form the allowed digits are 0-9.

```
double d = 123.456;
// Java program to illustrate the application of
// floating-point literals
public class Test {
    public static void main(String[] args)
    {
        int a = 101.230; // decimal-form literal
        int b = 0123.222; // It also acts as decimal literal
        int c = 0x123.222; // Hexa-decimal form
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}
```

Output:

```
101.230
123.222
Error: malformed floating point literal
```

NOTE: By default every floating point literal is of double type and hence we can't assign directly to float variable. But we can specify floating point literal as float type by suffixed with f or F. We can specify explicitly floating point literal as double type by suffixed with d or D. Of course this convention is not required.

Char literal

For char data types we can specify literals in 4 ways:

1. **Single quote:** We can specify literal to char data type as single character within single quote.
2. char ch = 'a';

3. **Char literal as Integral literal:** we can specify char literal as integral literal which represents Unicode value of the character and that integral literals can be specified either in Decimal, Octal and Hexadecimal forms. But the allowed range is 0 to 65535.
4. char ch = 062;

5. **Unicode Representation:** We can specify char literals in Unicode representation '\uxxxx'. Here xxxx represents 4 hexadecimal numbers.
6. char ch = '\u0061';// Here /u0061 represent a.

7. **Escape Sequence:** Every escape character can be specify as char literals.
8. char ch = '\n';

Example:

```
// Java program to illustrate the application of char literals
public class Test {
public static void main(String[] args)
  {
    char ch = 'a'; // single character literal within single quote
    char b = 0789; // It is an Integer literal with octal form
    char c = '\u0061'; // Unicode representation
    System.out.println(ch);
    System.out.println(b);
    System.out.println(c);
    // Escape character literal
```

```
        System.out.println("\"  is a symbol");
    }
}
```

```
a
error: Integer number too large
a
"  is a symbol
```

String literal

Any sequence of characters within double quotes is treated as String literals.

```
String s = "Hello";
```

String literals may not contain unescaped newline or linefeed characters. However, the Java compiler will evaluate compile time expressions, so the following String expression results in a string with three lines of text:

Example:

```
String text = "This is a String literal\n"
            + "which spans not one and not two\n"
            + "but three lines of text.\n";
```

```
// Java program to illustrate the application of String literals
public class Test {
    public static void main(String[] args)
    {
        String s = "Hello";
        // If we assign without "" then it treats as a variable
        // and causes compiler error
        String s1 = Hello;
        System.out.println(s);
        System.out.println(s1);
```

```
    }
}
```

```
Hello
error: cannot find symbol
symbol: variable Hello
location: class Test
```

boolean literals

Only two values are allowed for Boolean literals i.e. true and false.

```
boolean b = true;
```

```
// Java program to illustrate the application of boolean literals

public class Test
{
                public static void main(String[] args)
        {
            boolean b = true;
             boolean c = false;
             boolean d = 0;
             boolean b = 1;
               System.out.println(b);
               System.out.println(c);
               System.out.println(d);
               System.out.println(e);
          }
    }
```

Output:

```
true
false
error: incompatible types: int cannot be converted to boolean
error: incompatible types: int cannot be converted to boolean
```

NOTE: When we are performing concatenation operations, then the values in brackets are concatenated first. Then the values are concatenated from the left to the right. We should be

careful when we are mixing character literals and integers in String concatenation operations and this type of operations are known as **Mixed Mode operation**.

Output:

Simple48255

Explanation: Whenever we are performing addition between a string and integer then the overall result is converted into string. In the above program evaluation is done in the following way:

"Simple + first + '2' + second

"Simple " + 48 + '2' + 55

"Simple48" + '2' + 55

"Simple482" + 55

"Simple48255"

Operators in java

Operator in java is a symbol that is used to perform operations. For example: +, -, *, / etc.
There are many types of operators in java which are given below:

- Unary Operator,
- Arithmetic Operator,
- shift Operator,
- Relational Operator,

- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Java Operator Precedence

Level	Operators	Description	Associativity
15	() [] .	Function Call Array Subscript Member Selection	Left to Right
14	++ --	Postfix Increment / Decrement	Right to Left
13	++ -- + - ! ~ (type)	Prefix Increment / Decrement Unary plus / minus Logical negation / bitwise complement Casting	Right to Left
12	* / %	Multiplication Division Modulo	Left to Right
11	+ -	Addition / Subtraction	Left to Right
10	<< >> >>>	Bitwise Left Shift Bitwise Right Shift with sign extension Bitwise Right Shift with zero extension	Left to Right
9	< <= > >= instance of	Relational Less Than / Less than Equal To Relational Greater / Greater than Equal To Type Comparison for objects	Left to Right
8	== !=	Equality Inequality	Left to Right
7	&	Bitwise AND	Left to Right
6	^	Bitwise XOR	Left to Right
5	|	Bitwise OR	Left to Right
4	&&	Logical AND	Left to Right
3	||	Logical OR	Left to Right
2	?:	Conditional Operator	Right to Left
1	= += -= *= /= %= &= ^= |= <<= >>=	Assignment Operators	Right to Left

Now take a look at a statement

int myInt = 12 - 4 *

What will be the value of myInt? Will it be (12 - 4)*2, that is, 16? Or it will be 12 - (4 * 2), that is, 4?

When two operators share a common operand, 4 in this case, the operator with the highest precedence is operated first.

In Java, the precedence of * is higher than that of -. Hence, the multiplication is performed before subtraction, and the value of myInt will be 4.

The table above lists the precedence of operators in Java; higher it appears in the table, the higher its precedence.

Example: Operator Precedence

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

        int a = 10, b = 5, c = 1, result;result =
        a-++c-++b;

        System.out.println (result);
    }
}
```

When you run the program, the output will be:

```
2
```

The operator precedence of prefix ++ is higher than that of - subtraction operator. Hence,

```
result = a-++c-++b;
```

is equivalent to

```
result = a-(++c)-(++b);
```

When dealing with multiple operators and operands in an single expression, you can use parentheses like in the above example for clarity. The expression inside the parentheses is evaluated first.

Associativity of Operators in Java

If an expression has two operators with similar precedence, the expression is evaluated according to its associativity (either left to right, or right to left). Let's take and example.

```
a = b = c;
```

Here, the value of c is assigned to variable b. Then the value of b is assigned of variable a. Why? It's because the associativity of = operator is from right to left.

The table below shows the associativity of Java operators along with their associativity.

You don't need to memorize everything here. Most of the time, the precedence and associativity of operators makes sense in itself. You can always come back to this article for reference when in doubt. Also, you can use parenthesis if you think it makes your code easier to understand.

Java Expressions

Expressions consist of variables, operators, literals and method calls that evaluates to a single value. To learn about method calls, visit *Java methods*.

Let's take an example,

```
int score;
score = 90;
```

Here, score = 90 is an expression that returns int.

```
Double a = 2.2, b = 3.4, result;
result = a + b - 3.4;
```

Here, a + b - 3.4 is an expression.

```
if (number1 == number2)
System.out.println("Number 1 is larger than number 2");
```

Here, number1 == number2 is an expression that returns Boolean. Similarly, "Number 1 is larger than number 2" is a string expression.

Java Statements

Statements are everything that makes up a complete unit of execution. For example,

```
int score = 9*5;
```

Here, 9*5 is an expression that returns 45 and int score = 9*5; is a statement.

Expressions are part of statements.

Expression statements

Some expressions can be made into statement by terminating the expression with a ;. These are known as expression statements. For example:

```
number = 10;
```

Here, number = 10 is an expression where as number = 10; is a statement that compiler can execute.

++number;
Here, ++number is an expression where as ++number; is a statement.

Declaration Statements

Declaration statements declare variables. For example,
Double tax = 9.5;
The statement above declares a variable tax which is initialized to 9.5.

Java Blocks

A block is a group of statements (zero or more) that is enclosed in curly braces { }. For example,

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

        String band = "Beatles";

        if (band == "Beatles") { // start of block
                System.out.print ("Hey ");
                System.out.print ("Hello");
        } // end of block
   }
}
```

There are two statements System.out.print ("Hey "); and System.out.print ("Jude!"); inside the mentioned block above.
A block may not have any statements. Consider these examples:

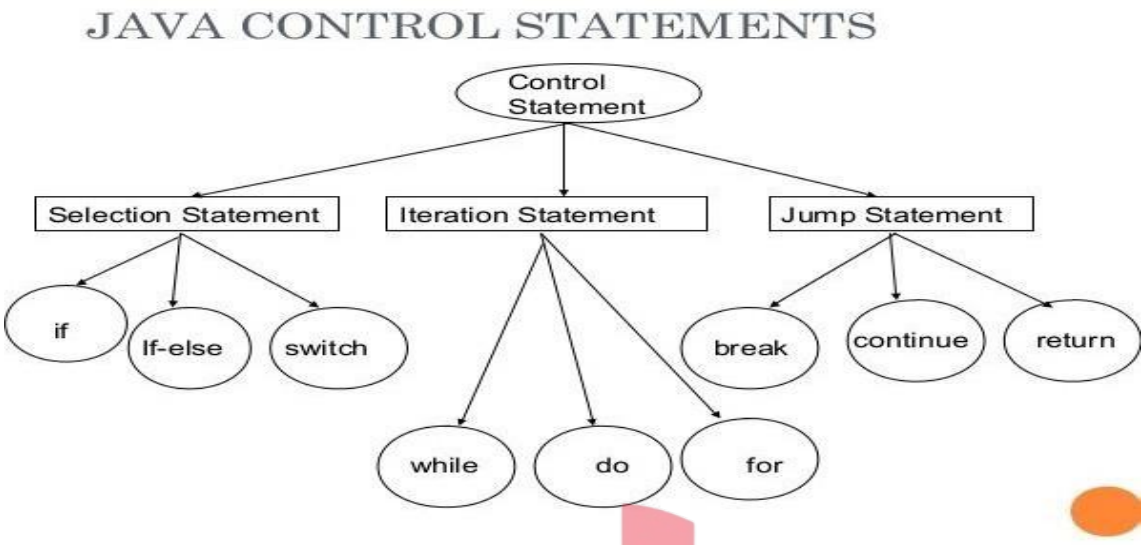
```
class AssignmentOperator {
   public static void main(String[] args) {

         if (10 > 5) { // start of block

         } // end of block
   }
}
class AssignmentOperator {
   public static void main (String[] args)
{// start of block

   } // end of block
}
```

Flow Control:



The basic building blocks of programs - variables, expressions, statements, etc. - can be put together to build complex programs with more interesting behavior. CONTROL FLOW STATEMENTS break up the flow of execution by employing decision making,Looping, and branching, enabling your program to conditionally execute particular blocks of code. Decision-making statements include the if statements and switch statements. There are also looping statements, as well as branching statements supported by Java.

Decision-Making Statements

if statement

```
if (x > 0)
y++; // execute this statement if the expression (x > 0) evaluates to “true”
 // if it doesn’t evaluate to “true”, this part is just skipped
 // and the code continues on with the subsequent lines
```

If-else statement - - gives another option if the expression by the if part evaluates to “false”

```
if (x > 0)
y++; // execute this statement if the expression (x > 0) evaluates to “true”
else
z++; // if expression doesn’t evaluate to “true”, then this part is executed instead
if (testScore >= 90)
grade = ‘A’;
else if (testScore >= 80)
grade = ‘B’;
else if (testScore >= 70)
grade = ‘C’;
else if (testScore >= 60)
```

grade = ‘D’;

else

grade = ‘F’;

Switch statement - - can be used in place of a big if-then-else statement; works with primitive types byte, short,char, and int; also with Strings, with Java SE7, (enclose the String with double quotes); as well as enumerated types,

```
int month = 8;
String monthString;
switch (month) {
case 1:
monthString = "January";
 break;
 case 2:
monthString = "February";
break;
case 3:
monthString = "March";
break;
default:
monthString = "Invalid month";
break;
}
System.out.println(monthString);
intmonthNumber = 0;
switch (month) {
case “January”:
monthNumber = 1;
break;
case “February”:
monthNumber = 2;
break;
etc …
default:
monthNumber = 0;
break;
}
System.out.println(monthNumber);
enum Day { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY,
SUNDAY }
Day day;
```

```
switch (day) {
case MONDAY:
System.out.println("Mondays are bad.");
break;
case FRIDAY:
System.out.println("Fridays are better.");
break;
case SATURDAY:
case SUNDAY:
System.out.println("Weekends are best.");
break;
default:
System.out.println("When will the weekend get here?");
break;
}
```

Looping Statements

While:

The while statement continually executes a block of statements while a particular condition is true. The while statement
continues testing the expression and executing its block until the expression evaluates to false.

```
while (expression) {
statement(s)
}
```

Do-While:

The do-while statement evaluates its expression at the bottom of the loop instead of the top, so the statements within the do block are guaranteed to execute at least once.

```
do {
statement(s)
} while (expression); < - - notice the semi-colon at the end of the do-while statement - - every statement in Java ends in either a } or a ;
```

For:

The for statement provides a way to iterate repeatedly until a particular condition is satisfied.

```
for (initialization; termination; increment/decrement)
{
statement(s)
}
```

There is another form of the for statement designed for iteration through arrays, sometimes referred to as the enhanced for
statement, or for-each. In the following example, the variable item holds the current value from the numbers array.

```
int[] numbers = {1,2,3,4,5,6,7,8,9,10};
for (int item : numbers) {
```

```
System.out.println ("Count is: " + item);
}
```

Branching Statements or Jump Statements

Break:

The break statement has two forms: labeled and unlabeled. The unlabeled break is like the one used in a switch statement.

You can also used an unlabeled break to terminate a for, while, or do-while loop, although this practice is usually seen as sloppy programming and is discouraged by some.

```
for (i = 0; i < arrayOfInts.length; i++) {
if (arrayOfInts[i] == searchfor) {
foundIt = true;
break;
 }
}
```

A labeled break statement terminates an outer statement that is labeled by some word. For example, if you have nested for loops, labeled with the word “search” right before the first for loop, you can put the following statement break search; inside the inner for loop to break out of both when the condition is met, and control flow continues with the statement immediately following the labeled statement.

```
 search:
 for (i = 0; i < arrayOfInts.length; i++) {
for (j = 0; j < arrayOfInts[i].length; j++) {
if (arrayOfInts[i][j] == searchfor) {
foundIt = true;
break search;
 }
 }
 }
```

Continue:

The continue statement skips the current iteration of a loop. The unlabeled form skips to the end of the innermost loop’s body and evaluates the expression that controls the loop.

```
for (int i = 0; i < max; i++) {
 // interested only in p's
if (searchMe.charAt (i) != 'p')
 continue;
 // process p's - only increments if it found a ‘p’
 numPs++;
}
```

A labeled continue statement skips the current iteration of an outer loop marked with the given label.

```
 test:
 for (int i = 0; i <= max; i++) {
 int n = substring.length();
 int j = i;
 int k = 0;
 while (n-- != 0) {
if (searchMe.charAt(j++)
 != substring.charAt(k++)) {
continue test;
 }
 }
foundIt = true;
break test;
 }
```

Return:

The return statement exits from the current method and returns control back to where the method was invoked from.