UNIT-III

JAVA

What is Java?

Java is a **programming language** and a **platform Independnet**. Java is a high level, robust, object-oriented and secure programming language.

Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995. *James Gosling* is known as the father of Java. Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

History of Java

James Gosling initiated Java language project in June 1991 for use in one of his many set-top box projects. The language, initially called ‘Oak’ after an oak tree that stood outside Gosling's office, also went by the name ‘Green’ and ended up later being renamed as Java, from a list of random words.

Sun released the first public implementation as Java 1.0 in 1995. It promised **Write Once, Run Anywhere** (WORA), providing no-cost run-times on popular platforms.

On 13 November, 2006, Sun released much of Java as free and open source software under the terms of the General Public License (GPL).

On 8 May, 2007, Sun finished the process, making all of Java's core code free and open-source, aside from a small portion of code to which Sun did not hold the copyright.

Tools You Will Need

You will need the following softwares −

- Linux or Windows operating system
- Java JDK 25
- Microsoft Notepad or any other text editor (netbeans/ eclipse)

Application

Some of them are as follows:

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

FEATURES OF JAVA:

- **Object Oriented** − In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- **Platform Independent** − Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
- **Simple** − Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
- **Secure** − With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **Architecture-neutral** − Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many proc..essors, with the presence of Java runtime system.
- **Portable** − Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable.
- **Robust** − Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
- **Multithreaded** − With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.

- **Interpreted** − Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
- **High Performance** − With the use of Just-In-Time compilers, Java enables high performance.
- **Distributed** − Java is designed for the distributed environment of the internet.
- **Dynamic** − Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

Types of Java Applications

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

1) Standalone Application

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, 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. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

4) Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

Java Platforms / Editions

There are 4 platforms or editions of Java:

1) Java SE (Java Standard Edition)

It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

2) Java EE (Java Enterprise Edition)

It is an enterprise platform that is mainly used to develop web and enterprise applications. It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, etc.

3) Java ME (Java Micro Edition)

It is a micro platform that is dedicated to mobile applications.

4) JavaFX

It is used to develop rich internet applications. It uses a lightweight user interface API.

C++ vs Java

There are many differences and similarities between the C++ programming language and Java. A list of top differences between C++ and Java are given below:

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is

		widely used in Windows-based, web-based, enterprise, and mobile applications.
Design Goal	C++ was designed for systems and applications programming. It was an extension of the C programming language.	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed to be easy to use and accessible to a broader audience.
Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by using interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write a pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses both compiler and interpreter. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform-independent.
Call by Value and Call by	C++ supports both call by	Java supports call by value only. There is no call by reference in

reference	value and call by reference.	java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comments.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not to override a function.	Java has no virtual keyword. We can override all non-static methods by default.
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.
Object-oriented	C++ is an object-oriented language. However, in the C language, a single root hierarchy is not possible.	Java is also an object-oriented language. However, everything (except fundamental types) is an object in Java.

Note

- Java doesn't support default arguments like C++.
- Java does not support header files like C++. Java uses the import keyword to include different classes and methods.

C++ Program Example

File: main.cpp

```
1. #include <iostream.h>
```

```
#include<conio.h>
void main() {
   cout << "Hello C++ Programming";
   getch()
}
```

Output:

```
Hello C++ Programming
```

Java Program Example

File: Simple.java

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

Output:

```
Hello Java
```

Save the above file as Simple.java.

To compile: javac Simple.java

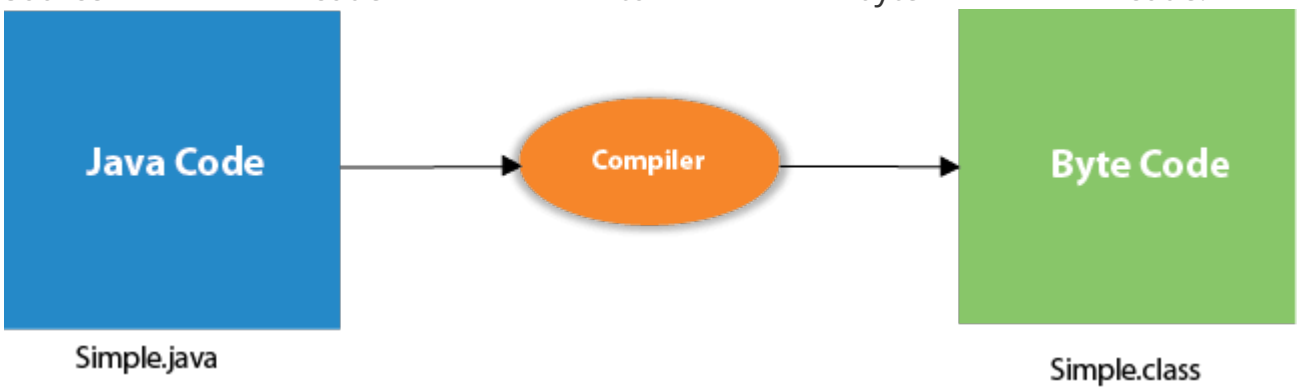
To execute: java Simple

Output:

```
Hello Java
```

Compilation Flow:

When we compile Java program using javac tool, the Java compiler converts the source code into byte code.

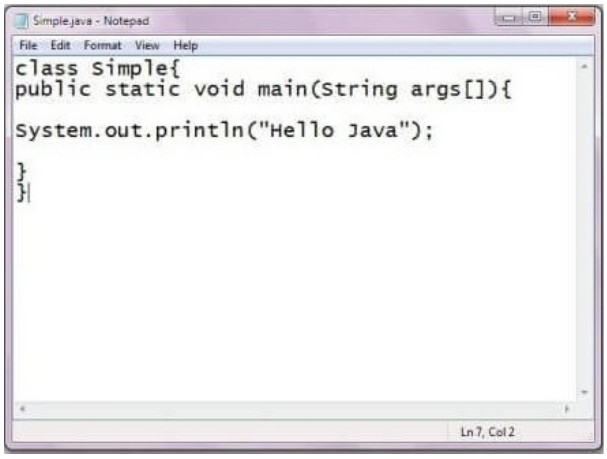


Parameters used in First Java Program

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 that represents visibility. It means it is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an 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 the starting point of the program.
- **String[] args** or **String args[]** is used for command line argument
- **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class.

To write the simple program, you need to open notepad by **start menu -> All Programs -> Accessories -> Notepad** and write a simple program as we have shownbelow:



Difference between JDK, JRE, and JVM

JDK

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets

. It physically exists. It contains JRE + development tools.

JRE

JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

JVM

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist.

JVMs are available for many hardware and software platforms.

JVM, I.E., JAVA VIRTUAL MACHINE.

- JVM is the engine that drives the Java code.
- Mostly in other Programming Languages, compiler produce code for a particular system but Java compiler produce Bytecode for a Java Virtual Machine.
- When we compile a Java program, then bytecode is generated. Bytecode is the source code that can be used to run on any platform.
- Bytecode is an intermediary language between Java source and the host system.
- It is the medium which compiles Java code to bytecode which gets interpreted on a different machine and hence it makes it Platform/Operating system independent.

JVM's work can be explained in the following manner

- Reading Bytecode.
- Verifying bytecode.
- Linking the code with the library.

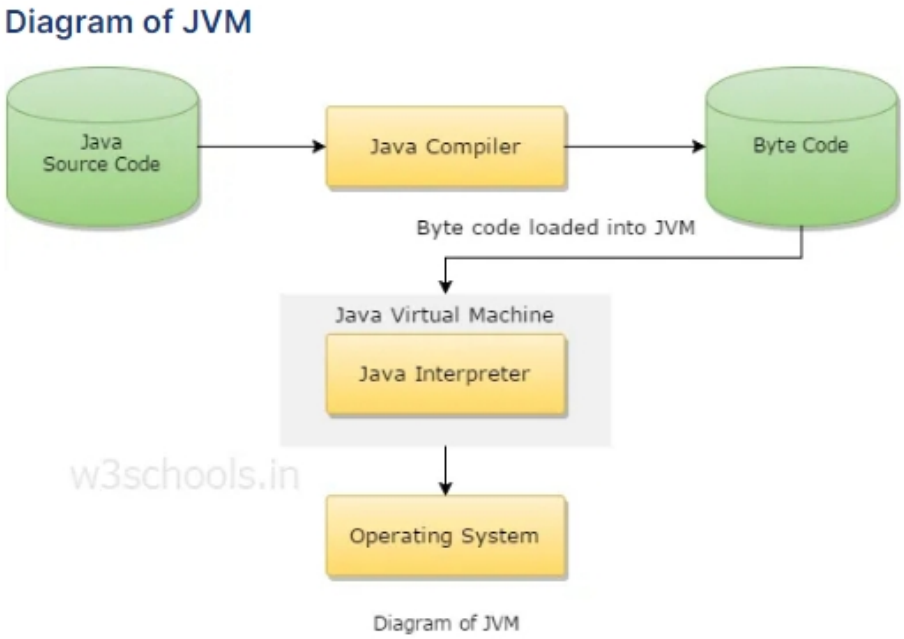


Diagram of JVM

Import Packages:

- The concept of a header file C++ does not exist in the Java programming language.
- In terms of Java, if you want to use classes within a package, you would need to import that package.

Types Of Java Package

Package can be built-in and user-defined, Java provides rich set of built-in packages in form of API that stores related classes and sub-packages.

- **Built-in Package:** math, util, lang, i/o etc are the example of built-in packages.
- **User-defined-package:** Java package created by user to categorize their project's classes and interface are known as user-defined packages. (next unit in detail)

Variables:

- A variable is a data name that may be used to store a data values. Unlike constants that remain unchanged during the execution of a program, a variable may take different values at different times during execution.

Constants:

- Constants is refered to fixed values that do not change during the execution of a program.

Character Set

The collection of characters which are used in C++ programming language are called character set. It consists of

1. Letters : a to z, A to Z

2. Digits : 0 to 9

3. White spaces

- Horizontal Tab \t

- New Line \n

4. Special characters

Rules for forming identifiers / variables:

- The first character should be an alphabet or underscore.
- Subsequent characters may be any combination of letters, digits and underscores.
- Special characters are not allowed
- Blank spaces are not permitted
- Reserved words cannot be used as identifiers.

Examples of valid identifiers:

- r
- sumOfRoots1and2
- _XYZ
- maxThrowsPerTurn
- TURNS_PER_GAME
- R2D2

Examples of invalid identifiers:

- 2hotToHandle // does not start with a letter
- Net Pay // contains a space
- ALPHA;BETA // contains an invalid character ;

Types of Variables

There are three types of variables in Java

- local variable
- instance variable
- static variable

1) Local Variable

A variable declared inside the body of the method is called local variable.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called an instance variable.

3) Static variable

A variable that is declared as static is called a static variable. It cannot be local.

Java Variable Example: Widening

```
1. public class Simple
2. {
3. public static void main(String[] args)
4. {
5. int a=10;
6. float f=a;
7. System.out.println(a);
8. System.out.println(f);
9. }
10. }
```

Java Variable Example: Narrowing (Typecasting)

```
1. import java.io.*;
2. import java.util.*;
3. public class Simple
4. {
5. public static void main(String[] args)
6. {
7. float f=10.5f;
8. //int a=f;//Compile time error
9. int a=(int)f;
10. System.out.println(f);
11. System.out.println(a);
12. }
13. }
```

Output:

```
10.5
10
```

Java Command Line Arguments

The java command-line argument is an argument i.e. passed at the time of running the java program.

The arguments passed from the console can be received in the java program and it can be used as an input.

Simple example of command-line argument in java

In this example, we are receiving only one argument and printing it. To run this java program, you must pass at least one argument from the command prompt.

```
 import java.io.*;
import java.util.*;

class CommandLineExample {

public static void main(String args[]) {
System.out.println("Your first argument is: "+args[0]);
}
}
```

1. compile by > javac CommandLineExample.java
2. run by > java CommandLineExample SJUIT

```
Output: Your first argument is: SJUIT
```

Example of command-line argument that prints all the values

In this example, we are printing all the arguments passed from the command-line.

For this purpose, we have traversed the array using for loop.

```
import java.io.*;
import java.util.*;
class line {
public static void main(String args[]) {

for(int i=0;i<args.length; i++)
System.out.println(args[i]);
}
}
```

compile by > javac line.java

1. run by > java line sjuit Bsc Sem 2

```
Output: sjuit
Bsc
Sem
```

Data Types in Java (similar to C++- Refer C++ notes)

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

Operators in Java (similar to C++- Refer C++ notes)

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 Ternary Operator

Java Ternary operator is used as one line replacement for if-then-else statement and used a lot in Java programming. It is the only conditional operator which takes three operands.

Java Ternary Operator Example

```
import java.io.*;
import java.util.*;
public class OperatorExample
{
public static void main(String args[])
{
int a=2;
int b=5;
int min=(a<b) ? a : b;
System.out.println("minimum value is " + min);
}
}
```

Output:

```
2
```

Java Keywords (new in Java)

- **abstract:** Java abstract keyword is used to declare an abstract class.
- **extends:** Java extends keyword is used to indicate that a class is derived from another class or interface.
- **final:** Java final keyword is used to indicate that a variable holds a constant value.
- **finally:** Java finally keyword indicates a block of code in a try-catch structure.

- **implements:** Java implements keyword is used to implement an interface.
- **import:** Java import keyword makes classes and interfaces available and accessible to the current source code.
- **interface:** Java interface keyword is used to declare an interface. It can have only abstract methods.
- **package:** Java package keyword is used to declare a Java package that includes the classes.

Java Control Statements | Control Flow in Java (similar to C++: Refer C++ notes)

Java provides three types of control flow statements.

1. Decision Making statements
    - if statements
    - switch statement
2. Loop statements
    - do while loop
    - while loop
    - for loop
3. Jump statements
    - break statement
    - continue statement

Java Program to print your college name 'n ' times using for loop

```
import java.io.*;
import java.util.*;
class Maxoftwo
{
    public static void main(String args[])
    {
int n=5;

for( i=0;i<n;i++)

    {

  System.out.println(" SJUIT");
```

```
    }

}}
```

Java Program to extract individual digit of a given number

```
import java.io.*;
import java.util.*;

public class Exp1

{
   public static void main(String args[])
   {
       int n=364,mod;
    System.out.print("Individual digits of 3 digit no are: ");
       while(n!=0)
       {
          mod=n%10;
          n=n/10;
          System.out.print(mod+ "\t");
       }
   }
}
```

Output

```
Individual digits of 3 digit no are: 4       6       3
```

Java Program to print maximum of two numbers using command line arguments

```
import java.io.*;
import java.util.*;
class Maxoftwo

{
    public static void main(String args[])
    {
        //taking value as command line argument.
        //Converting String format to Integer value
        int i = Integer.parseInt(args[0]);

        int j = Integer.parseInt(args[1]);

        if(i> j)

            System.out.println(i+" is greater than "+j);
        else
            System.out.println(j+" is greater than "+i);
    }
}
```

Java Program to print maximum of two numbers given during run time

```
import java.util.*;
import java.io.*;

public class Codes
{
   public static void main(String[] args)
   {
      int number1, number2, largest;
      Scanner scan = new Scanner(System.in);

      System.out.print("Enter the First Number: ");
      number1 = scan.nextInt();
      System.out.print("Enter the Second Number: ");
      number2 = scan.nextInt();

      if(number1>number2)
         largest = number1;
      else
         largest = number2;

      System.out.println("\nLargest = " +largest);
```

```
    }
}
```

Java Arrays

Normally, an array is a collection of similar type of elements which has contiguous memory location.

> **Java array** is an object which contains elements of a s+imilarsimilar data type.

Advantages

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

Disadvantages

- **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array in Java

Syntax to Declare an Array in Java

1. dataType[] arr;
   (or)
2. dataType []arr;

(or)

3. dataType arr[];

Instantiation of an Array in Java (creating a memory space)

1. datatype arrayRefVar=**new** datatype[size];

Example of Java Array

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

```
1. //Java Program to illustrate how to declare, instantiate, initialize
2. //and traverse the Java array.
3. class Testarray
4. {
5. public static void main(String args[])
6. {
7. int a[]=new int[5];//declaration and instantiation
8. a[0]=10;//initialization
9. a[1]=20;
10. a[2]=70;
11. a[3]=40;
12. a[4]=50;
13. //traversing array
14. for(int i=0;i<5;i++)
15. System.out.println(a[i]);
16. }
17. }
```

Output:

```
10
20
70
40
50
```

Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

```
1. int a[]={33,3,4,5};//declaration, instantiation and initialization
```

Let's see the simple example to print this array.

```
//Java Program to illustrate the use of declaration, instantiation
1. //and initialization of Java array in a single line
2. int a[]={33,3,4,5}; //declaration, instantiation and initialization

3. class Testarray  {
4. public static void main(String args[])  {
5. //printing array
6. for(int i=0;i<a.length;i++) //length is the property of array
7. System.out.println(a[i]);
8. } }
```

Output:

```
33
3
4
5
```

Java Program to find largest and smallest value using array

```
import java.util.*;
import java.io.*;
class Large_Small{
public static void main (String args[]) {
Scanner scan=new Scanner(System.in);
System.out.print("Enter the number of elements in an array: ");
```

```
int min,  max;

int arr[]=new int[5]; //declaring an array of n elements
        //for loop takes input from user
System.out.println("Enter 5 elements");
for(int i=0; i<5; i++)
{
arr[i]=scan.nextInt();//takes input from user for array
}
min=arr[0];//assume first element as smallest value
max=arr[0];//assume first element as largest value
for(int i=1; i<n; i++)
{
if(arr[i]<min)
{
min=arr[i];                    //loop for find minimum elements
}
if (arr[i]>max)
{
max=arr[i];    //loop for find maximum elements
}
}
System.out.print("\nThe smallest value is: "+min);
System.out.print("\nThe largest value is: "+max);
}//display result on the result
}
```

When the above code is compiled and executed, it produces the following results

```
Enter 5  elements
45
96
12
34
65
```

```
The smallest value is:12
The largest value is:96
```

Multidimensional Array in Java

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in Java

dataType[][] arrayRefVar; (or)

dataType [][]arrayRefVar; (or)
dataType arrayRefVar[][]; (or)
dataType []arrayRefVar[];

Example to instantiate Multidimensional Array in Java

```
1. int[][] arr=new int[3][3]; //3 row and 3 column
```

Example to initialize Multidimensional Array in Java

```
1. arr[0][0]=1;
2. arr[0][1]=2;
3. arr[0][2]=3;
4. arr[1][0]=4;
5. arr[1][1]=5;
6. arr[1][2]=6;
7. arr[2][0]=7;
8. arr[2][1]=8;
9. arr[2][2]=9;
```

Example of Multidimensional Java Array

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

```
//Java Program to illustrate the use of multidimensional array
class Testarray3 {
public static void main(String args[]) {
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
//printing 2D array
for(int i=0;i<3;i++)
 {
 for(int j=0;j<3;j++)
{
  System.out.print (  arr[i][j]+ " ");
  }
System.out.println();
}
} }
```

Output:

```
1 2 3
2 4 5
4 4 5
```

```
// Java Program to print total marks of ‘N’ students
import java.io.*;
import java.util.*;
class Student
{
   Scanner sc=new Scanner(System.in);
   int RegNo,Total=0,subjects;
   String name;
   int marks[];
   Student()
   {
      System.out.print("Enter Registration No.: ");
      RegNo=sc.nextInt();
      System.out.print("Enter Student Name: ");
      name=sc.next();;
      getDisMarks();
   }
   public void getDisMarks()
   {
       marks=new int[3];
       System.out.print("Enter marks of Physics: ");
       marks[0]=sc.nextInt();
       System.out.print("Enter marks of Chemistry: ");
       marks[1]=sc.nextInt();
       System.out.print("Enter marks of Maths: ");
       marks[2]=sc.nextInt();
       for(int i=0;i<3;i++)
       {
          Total+=marks[i];
       }
```

Objects and Classes in Java

What is an 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).

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

Object Definitions:

- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object is *an entity which has state and behavior*.
- The object is *an instance of a class*.

Syntax: Object creation

Classname objectname=new classname();

Eg:

Calculation c=**new** Calculation();

What is a 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.

A class in Java can contain:

- **Fields**
- **Methods**
- **Constructors**
- **Blocks**
- **Nested class and interface**

Syntax to declare a class:

```
class class_name
{
    attributes;
    method;
}
```

Instance variable in Java

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

Method in Java

In general, a **method** is a way to perform some task. Similarly, the **method in Java** is a collection of instructions that performs a specific task.

Advantage of Method

- Code Reusability
- Code Optimization

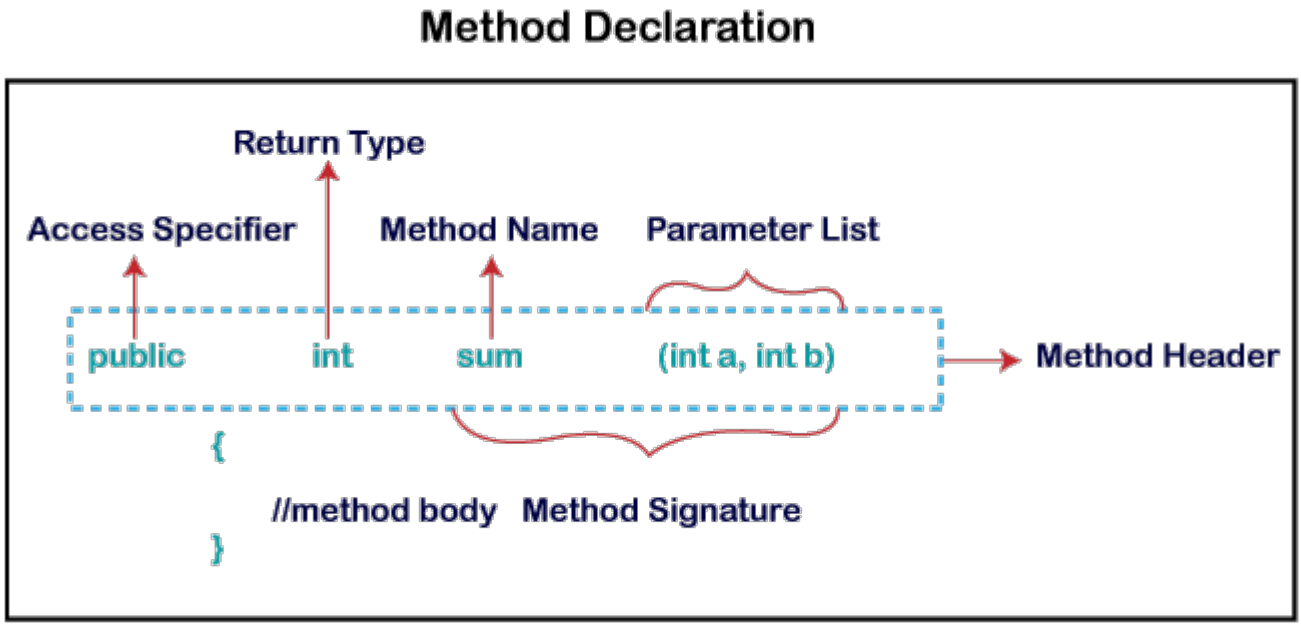
What is a method in Java?

A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation.

The most important method in Java is the **main()** method.

Method Declaration

The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments. It has six components that are known as **method header**, as we have shown in the following figure.



Method Signature: Every method has a method signature. It is a part of the method declaration. It includes the **method name** and **parameter list**.

Access Specifier: Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **four** types of access specifier:

- **Public:** The method is accessible by all classes when we use public specifier in our application.
- **Private:** When we use a private access specifier, the method is accessible only in the classes in which it is defined.
- **Protected:** When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
- **Default:** When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

Return Type: Return type is a data type that the method returns. If the method does not return anything, we use void keyword.

Method Name: It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method.

Parameter List: It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, left the parentheses blank.

Method Body: It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

Naming a Method

If the method name has more than two words, the first name must be a verb followed by adjective or noun. In the multi-word method name, the first letter of each word must be in **uppercase** except the first word. For example:

Single-word method name: sum(), area()

Multi-word method name: areaOfCircle(), stringComparision()

Types of Method

There are two types of methods in Java:

- Predefined Method
- User-defined Method

Predefined Method

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the **standard library method** or **built-in method**. We can directly use these methods just by calling them in the program at any point.

Some pre-defined methods are **length(), equals(), compareTo(), sqrt(),** etc. When we call any of the predefined methods in our program, a series of codes related to the corresponding method runs in the background that is already stored in the library.

Let's see an example of the predefined method.

Demo.java

```
1. public class Demo
2. {
3. public static void main(String[] args)
4. {
5. // using the max() method of Math class
6. System.out.print("The maximum number is: " + Math.max (9,7) );
```

```
7. }
8. }
```

Output:

```
The maximum number is: 9
```

In the above example, we have used three predefined methods **main(), print(),** and **max()**. We have used these methods directly without declaration because they are predefined. The print() method is a method of **PrintStream** class that prints the result on the console. The max() method is a method of the **Math** class that returns the greater of two numbers.

User-defined method

Let's see another program that return a value to the calling method.

In the following program, we have defined a method named **add()** that sum up the two numbers. It has two parameters n1 and n2 of integer type. The values of n1 and n2 correspond to the value of a and b, respectively. Therefore, the method adds the value of a and b and store it in the variable s and returns the sum.

Addition.java

```
1. public class Addition
2. {
3. public static void main(String[] args)
4. {
5. int a = 19;
6. int b = 5;
7. //method calling
8. int c = add(a, b);   //a and b are actual parameters
9. System.out.println("The sum of a and b is= " + c);
10. }
11. //user defined method
12. public static int add(int n1, int n2)   //n1 and n2 are formal parameters
13. {
14. int s;
15. s=n1+n2;
16. return s; //returning the sum
17. }
```

```
}
```

Output:

```
The sum of a and b is= 24
```

new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

Object and Class Example: main within the class

In this example, we have created a Student class which has two data members id and name. We are creating the object of the Student class by new keyword and printing the object's value.

Here, we are creating a main() method inside the class.

File: Student.java

```
//Java Program to illustrate how to define a class and fields
//Defining a Student class.
class Student {
                                    //defining fields
 int id;                              //field or data member or instance variable
 String name;
                                    //creating main method inside the Student class
 public static void main(String args[]) {
                                   //Creating an object or instance
  Student s1=new Student();           //creating an object of Student
                                    //Printing values of the object
  System.out.println(s1.id);              //accessing member through reference variable
  System.out.println(s1.name);
 }
}
```

Output:

```
0
null
```

2) Object and Class Example: Initialization through method

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

File: TestStudent4.java

```
1. class Student
2. {
3.  int rollno;
4.  String name;
5.  void insertRecord (int r, String n)
6. {
7.   rollno=r;
8.   name=n;
9.  }
10.  void displayInformation()
11. {
12. System.out.println(rollno+" "+name);
13. }
14. }
15. class TestStudent4
16. {
17.  public static void main(String args[])
18. {
19.   Student s1=new Student();
```

```
20.  Student s2=new Student();
21.  s1.insertRecord(111,"bahati");
22.  s2.insertRecord(222,"ibrahim");
23.  s1.displayInformation();
24.  s2.displayInformation();
25. }
26. }
```

Output:

```
111 bahati
222 ibrahim
```

Method Overloading in Java

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

> ***In Java, Method Overloading is not possible by changing the return type of the method only.***

1) Method Overloading: changing no. of arguments

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

In this example, we are creating static methods so that we don't need to create instance for calling methods.

```
1. class Adder{
2. static void add(int a,int b)
3. {  int c= a+b;
4. System.out.println( “c=”  + c);
5. }
6. static void add(int a,int b,int c)
7. {
8. int d=a+b+c;
9. System.out.println( “d=”  + d);
10. }
11. }
12. class TestOverloading1 {
13. public static void main(String[] args)   {
14. Adder x= new Adder ( ) ;
15. System.out.println(  “with two arguments);
16. x. add(11, 11);
17. System.out.println(“ with three arguments);
18. x. add(11,11,11);
19. } }
```

Output:

```
22
33
```

2) Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

```
1. class Adder{
2. static int add (int a, int b)
3. {
```

```
4.  return (a+b);
5.  }
6.  static double add(double a, double b)
7.  {
8.  return a+b;
9.  }
10. }
11. class TestOverloading2 {
12. public static void main(String[] args) {
13. Adder x = new Adder ( );
14. System.out.println(x.add(11,11));
15. System.out.println(x.add(12.3,12.6));
16. }}
```

Output:

```
22
24.9
```

Q) Why Method Overloading is not possible by changing the return type of method only?

In java, method overloading is not possible by changing the return type of the method only because of ambiguity. Let's see how ambiguity may occur:

```
1.  class Adder{
2.  static int add(int a,int b)
3.  {
4.  return a+b;
5.   }
6.  static double add(int a,int b)
7.  {
8.  return a+b;
9.  }
10. }
11. class TestOverloading3{
12. public static void main(String[] args){
13. Adder x= new Adder ( );
14. System.out.println(x.add(11,11));//ambiguity
```

15. }}

Output:

```
Compile Time Error: method add(int,int) is already defined in class Adder
```

System.out.println(Adder.add(11,11)); //Here, how can java determine which sum() method should be called?

> ***Note: Compile Time Error is better than Run Time Error. So, java compiler renders compiler time error if you declare the same method having same parameters.***

Can we overload java main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only. Let's see the simple example:

```
1. class TestOverloading4{
2. public static void main(String[] args){System.out.println("main with String[]");}
3. public static void main(String args){System.out.println("main with String");}
4. public static void main(){System.out.println("main without args");}
5. }
```

Output:

```
main with String[]
```

Constructors in Java

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

Rules for creating Java constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

```
class_name( )
{  // statements;
}
```

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
//Java Program to create and call a default constructor
class Bike1
{
//creating a default constructor
Bike1( )
{ System.out.println("Bike is created");
}
//main method
public static void main(String args[])
{
//calling a default constructor
Bike1 b=new Bike1( );
```

13. }
14. }

Output:

```
Bike is created
```

> ***Rule: If there is no constructor in a class, compiler automatically creates a default constructor.***



Q) What is the purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Example of default constructor that displays the default values

```
1. //Let us see another example of default constructor
2. //which displays the default values
3. class Student3 {
4. int id;
5. String name;
6. //method to display the value of id and name
7. void display() {
8. System.out.println(id+" "+name);
9. }
10.
11. public static void main(String args[]) {
12. //creating objects
13. Student3 s1=new Student3();
```

```
14. Student3 s2=new Student3();
15. //displaying values of the object
16. s1.display();
17. s2.display();
18. }
19. }
```

Output:

```
0 null
0 null
```

Explanation: In the above class,you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
1. //Java Program to demonstrate the use of the parameterized constructor.
2. class salesmanager{
3.    int id;
4.    String name;
5.    //creating a parameterized constructor
6.    salesmanager(int i,String n ){
7.    id = i;
8.    name = n;
9.    }
```

```
10.    //method to display the values
11.    void display( ) {
12. System.out.println(id+" "+name);
13. }
14.
15.    public static void main(String args[]) {
16.    //creating objects and passing values
17.    salesmanager s1 = new salesmanager(002,"joseph");
18.    salesmanager s2 = new salesmanager(006,"edson");
19.    //calling method to display the values of object
20.    s1.display();
21.    s2.display();
22.   }
23. }
```

Output:

```
002 joseph
006 edson
```

Difference between constructor and method in Java

There are many differences between constructors and methods. They are given below.

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class	The method name may or may

name.	not be same as the class name.

```
// Java Program to print total marks of ‘N’ students using arrays of objects
import java.io.*;
import java.util.*;
class Student
{
   Scanner sc=new Scanner(System.in);
   int RegNo,Total=0,subjects;
   String name;
   int marks[];
   Student()
   {
      System.out.print("Enter Registration No.: ");
      RegNo=sc.nextInt();
      System.out.print("Enter Student Name: ");
      name=sc.next();;
      getDisMarks();
   }
   public void getDisMarks()
   {
       marks=new int[3];
       System.out.print("Enter marks of Physics: ");
       marks[0]=sc.nextInt();
       System.out.print("Enter marks of Chemistry: ");
       marks[1]=sc.nextInt();
       System.out.print("Enter marks of Maths: ");
       marks[2]=sc.nextInt();
```

```
       for(int i=0;i<3;i++)
       {
          Total+=marks[i];
       }
       System.out.println("Total Marks of student "+name+": "
+Total);
   }
}
class StudentDemo
{
    public static void main(String args[])
    {
        Student s[]=new Student[5];
        for(int i=0;i<5;i++)
              s[i]=new Student();
     }
}
```

Output

```
Enter Registration No.: 624
Enter Student Name: Irene
Enter marks of Physics: 95
Enter marks of Chemistry: 94
Enter marks of Maths: 95
Total Marks of student Irene: 284
Enter Registration No.: 527
Enter Student Name: Francis
Enter marks of Physics: 97
Enter marks of Chemistry: 93
```

```
Enter marks of Maths: 89
Total Marks of student Francis: 279
Enter Registration No.: 093
Enter Student Name: Lucy
Enter marks of Physics: 85
Enter marks of Chemistry: 90
Enter marks of Maths: 95
Total Marks of student Lucy: 270
Enter Registration No.: 123
Enter Student Name: Jerome
Enter marks of Physics: 69
Enter marks of Chemistry: 79
Enter marks of Maths: 89
Total Marks of student Jerome: 237
Enter Registration No.: 789
Enter Student Name: smith
Enter marks of Physics: 75
Enter marks of Chemistry: 65
Enter marks of Maths: 24
Total Marks of student smith: 164
```

Java static keyword

The **static keyword** in java is used for memory management mainly. We can apply static keyword with variables , methods, blocks and nested classes

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)

1) Java static variable

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

- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

Advantages of static variable

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

Java Program to count the no. of objects created for the class.

```
import java.io.*;
import java.util.*;

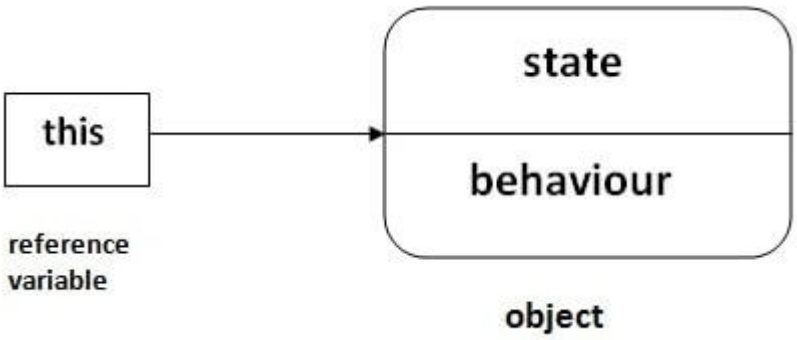
public class NumObj
      {
         static int count=0;

               NumObj()
                  {
                 count++;
                  }
         public static void main(String[] args)
         {
               Number_Objects obj1 = new Number_Objects();
               Number_Objects obj2 = new Number_Objects();
               Number_Objects obj3 = new Number_Objects();
               Number_Objects obj4 = new Number_Objects();
```

```
            System.out.println("Number of objects created:"+count);
        }
    }
```

this keyword in Java

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



1) this: to refer current class instance variable

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

Understanding the problem without this keyword

Let's understand the problem if we don't use this keyword by the example given below:

```
import java.io.*;
import java.util.*;
1. class customer {
2. int acct_no;
3. String name;
4. float balance;
5. Student(int acc_no,String name,float balance) {
6. acc_no=acc_no;
7. name=name;
8. balance=balance;
```

```
9.   }
10. void display()
11. {
12. System.out.println(acc_no+" "+name+" "+balance);
13. }
14. }
15.
16. class bank {
17. public static void main(String args[]) {
18. customer s1=new customer(111,"blue",5000f);
19. customer s2=new customer(112,"orange",6000f);
20. s1.display();
21. s2.display();
22. } }
```

Output:

```
0 null 0.0
0 null 0.0
```

In the above example, parameters (formal arguments) and instance variables are same. So, we are using this keyword to distinguish local variable and instance variable.

Solution of the above problem by this keyword

```
import java.io.*;
import java.util.*;
23. class customer {
24. int acct_no;
25. String name;
26. float balance;
27. Student(int acc_no,String name,float balance) {
28. this.acc_no=acc_no;
29. this.name=name;
30. this.balance=balance;
31. }
32. void display()
33. {
34. System.out.println(acc_no+" "+name+" "+balance);
```

```
35. }
36. }
37.
38. class bank {
39. public static void main(String args[]) {
40. customer s1=new customer(111,"blue",5000f);
41. customer s2=new customer(112,"orange",6000f);
42. s1.display();
43. s2.display();
44. } }
```

Output:

```
111 ankit 5000.0
112 sumit 6000.0
```

If local variables(formal arguments) and instance variables are different, there is no need to use this keyword .

Name Spaces (next unit)

Java packages are namespaces. They allow programmers to create small private areas in which to declare classes. The names of those classes will not collide with identically named classes in different packages.

Objects and classes in Java.lang .class:

Class	Description
Boolean	The Boolean class wraps a value of the primitive type `boolean` in an object.
Byte	The `Byte` class wraps a value of primitive type `byte` in an object.
Character	The `Character` class wraps a value of the primitive type `char` in an object.
Character.Subset	Instances of this class represent particular subsets of the Unicode character set.
Character.UnicodeBlock	A family of character subsets representing the character blocks in the Unicode specific
Class<T>	Instances of the class `Class` represent classes and interfaces in a running Java appli
ClassLoader	A class loader is an object that is responsible for loading classes.
ClassValue<T>	Lazily associate a computed value with (potentially) every type.
Compiler	The `Compiler` class is provided to support Java-to-native-code compilers and relate
Double	The `Double` class wraps a value of the primitive type `double` in an object.

Enum<E extends **Enum**<E>>	This is the common base class of all Java language enumeration types.
Float	The `Float` class wraps a value of primitive type `float` in an object.
InheritableThreadLocal<T>	This class extends `ThreadLocal` to provide inheritance of values from parent thread the child receives initial values for all inheritable thread-local variables for which the pa
Integer	The `Integer` class wraps a value of the primitive type `int` in an object.
Long	The `Long` class wraps a value of the primitive type `long` in an object.
Math	The class `Math` contains methods for performing basic numeric operations such as th and trigonometric functions.
Number	The abstract class `Number` is the superclass of classes `BigDecimal`, `BigIntege` and `Short`.
Object	Class `Object` is the root of the class hierarchy.
Package	`Package` objects contain version information about the implementation and specificat
Process	The **`ProcessBuilder.start()`** and **`Runtime.exec`** methods create a native of `Process` that can be used to control the process and obtain information about it.
ProcessBuilder	This class is used to create operating system processes.
ProcessBuilder.Redirect	Represents a source of subprocess input or a destination of subprocess output.
Runtime	Every Java application has a single instance of class `Runtime` that allows the applica application is running.
RuntimePermission	This class is for runtime permissions.
SecurityManager	The security manager is a class that allows applications to implement a security policy.
Short	The `Short` class wraps a value of primitive type `short` in an object.
StackTraceElement	An element in a stack trace, as returned by **`Throwable.getStackTrace()`**.
StrictMath	The class `StrictMath` contains methods for performing basic numeric operations su square root, and trigonometric functions.
String	The `String` class represents character strings.
StringBuffer	A thread-safe, mutable sequence of characters.
StringBuilder	A mutable sequence of characters.
System	The `System` class contains several useful class fields and methods.
Thread	A *thread* is a thread of execution in a program.
ThreadGroup	A thread group represents a set of threads.
ThreadLocal<T>	This class provides thread-local variables.

Throwable	The `Throwable` class is the superclass of all errors and exceptions in the Java langu
Void	The `Void` class is an uninstantiable placeholder class to hold a reference to the `Clas`