Java Tutorial for Beginners

Published On: September 20, 2024

Java Tutorial for Absolute Beginners

Java is a popular programming language used to build mobile apps, desktop apps, web apps, games, and scientific-related applications. We provide here a comprehensive understanding with real-time examples in our Java tutorial for the benefit of all Java aspirants. We suggest you work out the given programs to gain expertise in core concepts of Java at the end of this tutorial.

Download Java Tutorial PDF

Introduction to Java

Java is a class-based, secured, concurrent, general-purpose, and object-oriented programming language that makes it simple and easy to develop applications by beginners and professionals. You will learn the following in this Java tutorial:

- Overview of Java Programming
- Installation and Configuration

- Core Concepts of Java
- Control Statements in Java
- OOP Concepts in Java
- Other Java Concepts
- Networking in Java
- AWT in Java
- Swing in Java
- Applet in Java
- Database Connection in Java with JDBC

If you are new to Java, check out our **core Java training** in Chennai.

Java Interview Questions and Answers

Overview of Java Programming

As a high-level programming language, Java was introduced by Sun Microsystems in 1995 and is compatible with all major operating systems. Java is the trending programming language that brings a promising career for students to become software developers in top companies.

Java has the following advantages for the learners who learn with hands-on exposure:

- **OOPs-Based:** Java is an object-oriented programming language that can be extended limitless since everything in Java is based on an object.
- **Platform Independent:** JVM (Java Virtual Machine) is the reason for Java's write once, run anywhere feature. Java programs and projects developed in one operating system can be executed in any operating system. It supports all platforms, like Windows, macOS, iOS, Android, Linux, and UNIX.
- **Easy to Learn:** Java has an easy learning curve as it has a simple syntax and all the basic concepts are understandable by beginners, which makes them master core concepts quickly.

MERN Stack Tutorial for Web Development Aspirants
Published On: October 14, 2024

MERN Stack Tutorial for Web Development Aspirants There is a growing need for competent MERN...

Tableau Developer Salary in Chennai
Published On: October 12, 2024

Introduction A Tableau Developer designs, develops, and maintains dashboards and visualizations using Tableau software. Key...

VMware Tutorial for Cloud Computing Aspirants
Published On: October 12, 2024

VMware Tutorial for Cloud Computing Aspirants VMware software allows you to run a virtual machine...

VBA Macros Tutorial for Beginners
Published On: October 10, 2024

- **Robust and Secure:** Java can handle errors during the execution and manage incorrect input data efficiently. It utilizes strong memory management. Java is a very secured programming language to develop virus-free and tamper-free applications with authentication facilities through public-key encryption.
- **Architecture-Neutral:** Java has a strong compiler that generates an architectural-neutral object file format. It provides the feature to execute the project on many processors, along with the presence of the Java Runtime System.
- **Portable:** Java is highly portable as it is architecture-neutral and has no implementation-dependent aspects. The compiler of Java is written in ANSI C.

VBA Macros Tutorial for Beginners VBA macros are programs that automate repetitive operations in Microsoft...

Applications of Java Programming

According to the Sun Microsystems report, nearly 3 billion devices are running with Java.

It is used as,

- Desktop applications such as Acrobat Reader, antivirus software, and media players
- Web applications like IRCTC.in
- Enterprise applications like banking processes
- Numerous mobile applications
- Embedded system applications
- Smart card processing
- Robotics
- Game applications

It is categorized into the following four major types.

- **Standalone Applications:** Desktop and Windows-based applications are called standalone applications and they require to be installed in every individual machine.
    - *AWT (Abstract Window Toolkit) and Swing Concepts* of Java are used to develop standalone applications.

- **Web Applications:** Websites and web applications are developed in Java for creating server-side concepts and dynamic pages.
    - Java concepts such as *JSP (Java Server Page), Servlet, Spring, Struts, JSF (Java Server Faces), and Hibernate* are implemented to develop web applications.
- **Enterprise Applications:** Enterprise applications are developed to satisfy the requirements of large organizations such as schools, banks, businesses, internet-based groups, clubs, government projects, and charities. These applications have high-level security, clustering, and load-balancing features to provide the best service for the customers.
    - *EJB (Enterprise Java Beans)* of Java is used to build highly scalable and robust enterprise-level applications.
- **Mobile Applications:** Applications that are used in mobile devices are called mobile applications.
    - *Android and Java ME (Micro Edition)* are used to create mobile-based applications.

Do you want to kickstart your career in **mobile app development**? Try our specialized courses.

Download Java Syllabus PDF

Major Platforms / Editions of Java

There are four major platforms or editions in the Java programming language

Java SE (Java Standard Edition)

This is the popular edition that includes APIs like Java.lang, Java.net, Java.sql, Java.io, Java.math, Java.util, and so on.

Java SE consists of core Java concepts such as

OOPs, RegEx, inner classes, strings, multithreading, AWT, swing, networking, I/O streams, collections, exceptions, reflection, and so on.

Java EE (Java Enterprise Edition)

This is used for developing enterprise-level applications and it is built on the top of SE Edition.

Java EE consists of JSP, EJB, Web Services, Servlet, JPA, and so on. J2EEE (Java Enterprise Edition Environment) is the platform that provides features and specifications to extend Java SE concepts to develop distributed computing and web services.

Java ME (Java Micro Edition)

It is used for developing mobile applications and embedded applications, along with IoT device development.

Java ME provides a flexible platform with robust and secure features for developing efficient mobile, embedded, and IoT applications like microcontrollers, gateways, personal digital assistance, TV set-top boxes, mobile phones, printers, and sensors, etc.

Java FX

JavaFX is the popular and trending platform for creating rich internet and desktop applications with special effects. It contains the features of GUI and Swing to run any systems and devices with the same effects.

Java FX can be implemented for developing applications with animations, event delivery mechanisms, render tree interfaces, geometry and shapes, and logging support.

Popular Terminologies of Java

Some of the frequently used Java terms and its applications:

JVM or Java Virtual Machine

Java Virtual Machine is an abstract machine that doesn't physically exist and provides a runtime environment for the Java bytecode to be executed.

- It is also used to run programs that are written in any other language and compiled in Java ByteCode.
- It is platform-independent and has three notions, like specification, instance, and implementation.
- JVM performs various tasks such as loading code, verifying code, executing code, and runtime environment.

JRE or Java Runtime Environment

Java Runtime Environment is a collection of software tools used to develop Java applications with JVM implementations. It exists physically and consists of numerous libraries and files to be used by JVM.

JDK or Java Development Kit

JDK is a software development environment used for developing applications and applets in Java. It contains JRE and development tools.

It physically exists and it includes JDK as a private JVM, interpreter, loader (java), compiler (javac), archiver (jar), and document generator (Javadoc).

These all concepts are used to develop full-fledged Java applications.

History of Java

Java Programming Language is developed by James Gosling and released by Sun Microsystems in the year 1995.

- It was first named Oak to indicate a tree, then later changed to the name "Java" to indicate coffee bean.
- It was recognized as one of the top ten products of 1995 by Time Magazine as it came

with the “Write Once, Run Anywhere” feature.
- Java was available as open-source software by Sun Microsystems since 2007.
- Java updates its features in the following versions that come every year and the stable version is Java SE 15, released in September 2020.
- Java SE 21 is the most recent version of the Java Standard Edition. It was updated on 19 Sep 2023.
    - *As Java developed and gained popularity, many configurations were created to accommodate different kinds of platforms. J2ME for mobile applications and J2EE for enterprise applications are two examples.*
- **Newly Added Features:**
    - String Templates (Preview)
    - Sequenced Collections
    - Generational ZGC
    - Record patterns
    - Pattern matching for switches.

Prerequisites for learning Java

Here are the basic prerequisites for learning Java as a beginner:

System Requirements:

- The learner should have a computer system with Windows XP 7 or above or Linux 7.1 with JDK 8 to learn Java Programming Language.
- Microsoft Notepad or any other text editor will be used for writing codes.

Prior Knowledge:

- Basic knowledge in computer systems and OOP concepts to learn core Java concepts to Java Advanced techniques.
- Fundamental understanding of GUI, networking, and web application development.

If you find happiness in creating impressive websites, our **web design courses** are the best start.

Java Developer Salary in Chennai

Installation and Configuration Guide

Check the system if Java is installed in the system already by providing the following code in command prompts (cmd.exe).

C:\Users\Your Name> java –version

If Java is installed, you will see the following results:

Java version "11.0.1" 2018-10-16

Java(TM) SE Runtime Environment 18.9 (build 11.0.1+13 – LTS)

Java HotSpot (TM) 64-Bit Server VM 18.9 (build 11.0.1+13 – LTS, mixed mode)

If Java is not installed on your computer, it can be downloaded for free at Oracle.com.

To install and configure the Java application setup, you can follow the below guidelines.

For Windows:

Step 1: Go to "*System Properties" in Start > Control Panel > System and Security > System > Advanced System"* Settings.

Step 2: Click the *"Environmental Variables"* button of the "*Advanced*" tab.

Step 3: Choose the "*Path*" variable in system variables and click the "*Edit*" button.

Step 4: Click the "*New*" button and add the path of the location where the Java application is installed, followed by *\bin. Ex: C:\Program Files\Java\jdk-11.0.1\bin*, then click "*OK*" to save the settings.

Step 5: Open the command prompt (cmd.exe) and type java-version to ensure Java is running in your computer system.

Sample Program

A Java application should begin with a class name and that class should match the filename.

The following is the sample Java program and we name it “sample.”

The program can be typed in Notepad or any other text editor.

Sample.java

```
public class Sample
{
        public static void main(String[] args)
        {
                System.out.println(“Hai There, Welcome”);
        }
}
```

Output:

Hai There, Welcome

Step-by-step explanation

In this sample program, we name it “Sample.Java.”.

Therewith, the class name began with Sample.

- Save the code in Notepad or text editor with

"Sample.Java."

- Open Command Prompt (cmd.exe) and navigate to the directory where you have saved your file and type "javac Sample.java."
- C:\Users\Your Name>javac Sample.java
- Once it compiles the code, then it takes you to the next line. Type "java sample."
    - *C:\Users\Your Name>java Sample*
- Then the output will be
    - *Hai there, welcome*

Congrats! Now you have written and executed the first sample program. Way to go!

Setting up the configuration and path for Linux and UNIX

Environmental variable PATH must be configured to point to the location of the Java binaries installed. If you use bash as your shell, it should be as follows:

bashrc: export PATH = /path/to/java:$PATH

Java Editors

There are some popular Java editors available to write Java programs and they are called IDEs (Integrated Development Environments).

- **Notepad and TextPad:** It can be utilized in Windows machines for writing the programs.
- **Netbeans:** This is a Java IDE available at open-source and free. It can be downloaded from *http://www.netbeans.org/index.html.*
- **Eclipse:** This is also a popular IDE of Java developed by Eclipse, the open-source community and it can be downloaded from *http://www.eclipse.org*.

Core Concepts of Java

In the Java programming language, the following are considered building blocks:

- **Object:** An object is the instance of a class. It has states and behavior. Ex: A flower has states

like color, name, and variety and behavior like smell, and benefits.

- **Class:** A class is the collection of objects and it can be described as the blueprint that indicates the state and behavior of an object.
- **Method:** A method is an independent behavior of an object such as logic, data manipulation, and actions that are executed in a class. A class can contain multiple methods.
- **Instance Variables:** An object has a unique set of instance variables and it is created by the values assigned to that particular instance variable.

Syntax Rules for Java Programming

Java syntax is simple to learn and the following are the important things to learn to use syntax in a Java program.

- **Case Sensitive:** Hi and hi are different in Java
- **Class Names:** The first letter and the inner words of a class name should be in upper case.
  - **Example:** SampleJavaProgram
- **Method Names:** All method names must begin with lowercase letters, and the inner words should start with uppercase.
  - **Example:** mySampleJavaProgram()
- **Program File Name:** The filename of the Java program should match with the class name and append with .java.
  - **Example:** If a class name is Sample, then the file should be saved as SampleJavaProgram.java.
- **Public static void main (String args[]):** This is the mandatory part of every Java program.

Identifiers of Java

Identifiers are the symbolic names used to identify through class name, variable name, package name, method name, constant name, etc.
Identifiers should be given as follows:

- The identifier should start with any upper or lower case, currency character, or underscore.
- Keywords cannot be used as identifiers.
- It is case sensitive and it can have any combination of characters.

Examples:

Legal: Age, gender, $salary, _name, _2_height

Illegal: 12_Red, -domain, int.

Modifiers in Java

Modifiers are the things used to set the access level of classes, attributes, constructors, and methods. It has the following two categories: access modifiers such as default, public, private, and protected, and non-access modifiers such as final, strictfp, and abstract.

Access Modifiers

- **public:** The class is accessible by any other classes within the project.
- **private:** The class is accessible only within the declared class.
- **protected:** The class is accessible only from derived classes.
- **default:** The class is accessible only within the package.

Non-Access Modifiers

- **final:** The class cannot be accessed or inherited by any other classes.
- **abstract:** The class cannot be used to create objects.
- **static:** Attributes and methods of the class belong to the same class.

Example Program:

> *public class Sample*
>
> *{*

```
        final int v = 10;

        final double PI = 3.14

        public static void main(String[]
args)

        {

                Sample mySa = new
Sample ();

                mySa.x = 20; //It
shows error as we cannot assign a
value to a final variable

        System.out.println(mySa.x);

        }

}
```

Output

Sample.java: 8: error: cannot assign a value to final variable x

```
    mySa.x = 20;
```

Variables of Java

Variables are names of the memory location that reserved space for data. Java variables are generally of three types, such as local variables, static or class variables, and instance or non-static variables.

- **Local variables** are declared inside the body of a method and it can only use those variables within the particular method. It cannot be defined with the 'static' keyword.
- **Instance variables** are declared in the class

but outside of the method's body, as they can't be shared among instances and will not be declared with the 'static' keyword.
- **Static variables** are declared as static and they can be shared among all the instances of the class. Memory allocation of a static variable is allocated only once for all access.

Example for variables:

> *class F*
>
> *{*
>
> *int a = 12; //instance variable*
>
> *static int b = 100; //static variable*
>
> *void method()*
>
> *{*
>
> *int c = 20; //local variable*
>
> *}*
>
> *}*

Data Types in Java

Data types denote the various size and value types to be stored in the variable. Following are the two data types that can be used in Java.

Primitive Data Types

These are predefined data types in Java and they will be named by a keyword. There are eight primitive data types, as follows:

byte: 1 byte (whole numbers from -128 to 127),

Ex: byte b = 100;

short: 2 bytes (whole numbers from -32,768 to 32,767),

Ex: short b = 3000;

Int: 4 bytes (whole numbers from -2,147,483,648 to -2,147,483,647),

Ex: int b = 30000;

Long: 8 bytes (whole numbers from -9,223,372,036, 854, 775, 808 to 9,223,372,036, 854, 775, 807

Ex: long b = 130000000000L;

Float: 4 bytes (fractional numbers that store 6 to 7 decimal digits)

Ex: float n = 5.98f;

Double: 8 bytes (fractional numbers that store 15 decimal digits)

Ex: double n = 12.21d;

boolean: 1 bit (true or false values)

Ex: boolean isJavaSimple = True;

Boolean isJavaHard = false;

***char:** 2 bytes (a single character or ASCII values)*

Ex: char gGender = 'F';

Non-Primitive Data Types

This is a reference type, as it refers to objects. It will be created by the user to call methods to perform various operations. The value of non-primitive data types can be null and it begins with an uppercase letter. Non-primitive data types are all in the same size.

Strings: It is used in Java to store a set of characters in the text and it must be within the double quotes symbol.

Ex: String note = "Hi, There"

Arrays: Arrays are objects used to store multiple variables that have the same attributes. It can be initialized as an object.

Ex: int[] n = {11, 12, 13, 14};

Java Enums: Enums are used to restrict a variable to have predefined values and these values will be enumerated. It is used to reduce the number of bugs in the code. It can be declared inside the class.

Keywords of Java

Java keywords are the reserved words to identify the type of a variable or the function of a class, and so on.

All the keywords of Java begin with a small letter and some of the popular keywords are abstract, assert, break, byte, case, char, class, continue, default, do, else, enum, extends, float, for, goto, if, import, interface, int, long, new, package, private, protected, return, static, short, switch, throw, try, catch, void, and while, and so on.

Learn the programming fundamentals through our **C and C++ training** in Chennai.

Comments in Java

In the Java programming language, we can give comments in single line or multiline and these lines will not be compiled or executed.

- It is used to explain to the viewer what the particular line of code is all about.
- It will be started with /* and end with */ for multiple-line comments and // for a single-line comment.
- If the line that contains the comment without any code to execute is called white spaces.

Operators in Java

Operators in Java are used to perform various

operations on variables and values. It is divided into five major groups:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Bitwise Operators.

Arithmetic Operators in Java

These are used to perform fundamental mathematical operations as follows:

Operator	Name	Description	Example
+	Addition	Adds two values	a + b
–	Subtraction	Subtracts one value from another	a – b
*	Multiplication	Multiplies two values	a * b
/	Division	Divides one value by another value	a / b
%	Modulus	Returns the reminder of division	a % b
++	Increment	Increases the value of a variable by 1	++a
		Decreases	

—	Decrement	the value of a variable by 1	–a

Assignment Operators in Java

These are used to assign values to the variables with (=) ex: a = 10;

Operator	Example	Operation
=	a = 4	value 4 assigned to variable a
+=	a += 2	a = a + 2
-=	a -= 2	a = a – 2
*=	a *= 2	a = a * 2
/=	a /= 2	a = a / 2
%=	a %= 2	a = a % 2
&=	a &= 2	a = a & 2
| =	a | = 2	a = a | 2
^ =	a ^ = 2	a = a ^ 2
>> =	a >> = 2	a = a >> 2
<< =	a << = 2	a = a << 2

Comparison Operator in Java

These operators are used to compare two values.

Operator	Name	Example
==	Equal to	a == b
! =	Not equal	a ! = b
>	Greater than	a > b
<	Less than	a < b

>=	Greater than or equal to	a > = b
<=	Less than or equal to	a < = b

Logical Operators in Java

These are used to determine the logic between values and variables.

Operator	Name	Description	Example
&&	Logical AND	Returns true if both statements are true.	a < 3 && a < 5
||	Logical OR	Returns true if any one of the statements is true.	a < 3 || a > 9
!	Logical NOT	Reverse the result and return false if the result is true.	! (a < 3 && a < 9)

Java Project Ideas

Control Statements in Java

The control statement is a statement that decides which of the given two statements to be executed and it ensures the smooth flow of the program. There are various types of control statements in the Java programming language, as follows:.

Decision-Making Statement in Java

Statements that decide which condition to execute are called decision-making statements and they will be executed in four following ways:

- If Statement
- If-Else Statement
- If-Else-If Ladder
- Nested If Statement

If Statement

The block of statement will be executed if the given condition is true.

Syntax:

```
If (condition)

{

//Block of code to be executed

}
```

Example:

```
public class SampleIF

{

public static void main(String[] args)

{

int age = 21;

if (age > 18)

{

System.out.println("Eligible for Vote");

}
```

```
}

}
```

Output

Eligible for Vote

If-Else Statement

There will be two blocks of codes. If the given condition is true, the block of code under the if statement will be executed; otherwise, the else part will be executed.

Syntax:

```
If (Condition)

{

//If block of code

}

Else

{

//Else block of code

}
```

Example:

```
public class SampleIfElse

{

public static void main(String[] args)

{
```

```
int n = 12;

if(n>18)

{

System.out.println("Eligible for Vote")

}

else

{

System.out.println("Not Eligible to
Vote")

}

}

}
```

Output

Not Eligible to Vote

If-Else-If Ladder Statement

The statement will be executed through multiple conditions.

Syntax:

```
if (condition1)

{

//1st block of code

}

else if(condition 2)
```

```
{
//2nd block of code
}
else if (condition 3)
{
//3rd block of code
}
....
else
{
//nth block of code
}
```

Example

```
public class SampleIfElseLadder
{
public static void main(String[] args)
 {
    int n = 5;
if(n<3)
{
```

```
        System.out.println("Winter Season
");

    }

else if(n>3 && n<6)

{

        System.out.println("Spring
Season");

    }

    else if(n>=6 && n<9)

{

        System.out.println("Summer
Season ");

    }

    else if(n>=9 && n<12)

{

        System.out.println("Autumn
Season");

    }

else

{

        System.out.println("Invalid!");

    }
```

```
}

}
```

Output

Spring Season

Nested If Statement

The nested if in Java represents the block of code with an if condition that will be executed within another block of code that has an if condition. In this case, the inner if condition will be executed only when the outer if condition is true.

Syntax

```
If (condition)

{

//Block of code

If (condition)

{

//Another block of code

}

}
```

Example

```
public class SampleNestedIf

{

public static void main(String[] args)

{
```

```
int age = 20;

int weight = 70;

if(age >=20)

{

if(weight>45)

{

System.out.println("Eligible to donate organ")

}

}

}

}
```

Switch Statement in Java

The switch statement of Java is applied to execute a single statement from multiple conditions. It accepts statements with short, byte, long, and enum types of data. We can give n numbers of cases and those case values should not be replicated. It contains a break statement to terminate the statement flow and execute the next statement. The break statement is optional.

Syntax:

```
switch(expression)

{

case value 1:
```

```
//block of code to be executed;

break; //Optional

case value 2:

//block of code to be executed;

break;//optional

.....

case value n:

//block of code to be executed;

break; // optional

default:

//block of code if all the above cases are not
matched;

}
```

Example:

```
public class SampleSwitch

{

public static void main(String[] args)

{

int n = 3;

switch(n)

{
```

```
Case 1:

System.out.println("A");

break;

Case 2:

System.out.println("B");

break;

Case 3:

System.out.println("C")

break;

default:

System.out.println("None of the cases matched");

}

}

}
```

Output:

C

Java Code Challenges for Beginners

Loop Statement in Java

Loop statements are used to execute the block of codes or the set of instructions repeatedly until the given condition becomes true. There are three types of loop statements in Java.

- For loop
- While loop
- Do-while loop

For Loop

The Java for Loop used to control the flow of statements to iterate a part of programs multiple times. It can be used when the number of iterations is known and fixed.

Syntax

```
for (init; condition;
increment/decrement)

{

//block of code to be executed;

}
```

Example:

```
public class SampleFor

{

public static void main(String[] args)

{

for(int = 1; i<=5; i++)

System.out.println(i);

}

}

}
```

Output

1

2

3

4

5

While Loop in Java

The while loop is used for iterating a block of code inside the program multiple times until the given condition hits false.

Syntax

```
while (condition)

{

//block of code;

}
```

Example

```
public class SampleWhile

{

public static void main(String[] args)

{

int I = 1;

while(i<=5)

{

System.out.println(i);
```

```
    }
}
}
```

Output

```
1
2
3
4
5
```

Do While Loop in Java

The “Do While Loop” of Java is used to iterate a part of code several times and it is used if the number of iterations is not given. It is executed at least once as the condition is checked after loop codes.

> **Syntax**
>
> ```
> do
> {
> //Block of code
> }
> while (condition)
> ```

Example

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

```
{
int n = 1;
do
{
System.out.println(n);
n++
}
while(n<=4);
}
}
```

Output

```
1
2
3
4
```

Upgrade your fundamental programming skills with our **advanced Java training** classes.

OOP Concepts in Java

Object-oriented programming is an important part of the Java programming language, as it provides numerous features for developing projects quickly and efficiently. It consists of object, class, abstraction, inheritance, polymorphism, and encapsulation.

Object: An object is defined as an instance of a class and it has address and memory space. It is

used to communicate without any background data of a class or method. The type of message and the returned response are the significant things of an object.

Example: *Book. A book has attributes like paper, pages, cover design, etc., as well as behavior such as message, author's mindset, and so on.*

Class: Class is a collection of objects and it consumes no space in memory. It can also be defined as a blueprint; there we can create individual objects.

Example:

> *public class peacock*
>
> *{*
>
> *int age;*
>
> *String color;*
>
> *String place;*
>
> *void dancing()*
>
> *{....}*
>
> *void roaming()*
>
> *{....}*
>
> *void sleeping()*
>
> *{....}*
>
> *}*

Creating an object for a class

A class provides a blueprint for an object and the

object is created from a class using a new keyword.

Following are the three steps used for creating an object for a class:

- **Declaration:** Variable declaration with name for an object type
- **Instantiation:** 'new' is the keyword used to create an object for a class.
- **Initialization:** 'new' is the keyword followed by a call to the constructor to initialize the object.

> > *public class Parrot*
> >
> > *{*
> >
> > *public Parrot(String color)*
> >
> > *{*
> >
> > *System.out.println("The color of the parrot is" + color);*
> >
> > *}*
> >
> > *public static void main(String[] args)*
> >
> > *{*
> >
> > *Parrot myParrot = new Parrot("Green");*
> >
> > *}*
> >
> > *}*

Output:

The color of the parrot is Green

Inheritance: If one class requires the objects and methods of another class, inheritance will be implemented. It provides a reusable feature and it is

used to achieve development quickly.

Important terms used in inheritance

- **Base Class/Super Class/Parent Class:** This is the class from where other classes can inherit the features and properties.
- **Subclass, Child Class or Derived Class:** This is the class that inherits the properties from the base class.
- **Reusability:** This is the facility that the developer can reuse the fields and methods of an existing class to create a new project or class.

Syntax

> *Class SampleSub-name extends SampleSuperclass-name*
>
> *{*
>
> *//Methods and properties*
>
> *}*

Example

> *class Student*
>
> *{*
>
> *float marks = 472;*
>
> *}*
>
> *class Sports extends Student*
>
> *{*
>
> *int score = 8;*

```
public static void main(String[] args)

{

Sports s = new Sports();

System.out.println("The stundent's
mark is: "+s.marks);

System.out.println("The student's sport
score is:"+s.score "+out of 10");

}

}
```

Output

The student's mark is: 472

The student's sport score is 8 out of 10

Types of inheritance

There are five types of inheritance, such as single inheritance, multilevel inheritance, hierarchical inheritance, multiple inheritances, and hybrid inheritance. Except Multiple Inheritances, all four inheritances will support the Java programming language.

Single Inheritance

When one class inherits another class, it is known as a single inheritance. In this, there will be one base class and only one derived class.

```
class Bird

{

void food()

{
```

```
System.out.println("Grains");
}
class Parrot extends Bird
{
void fly()
{
System.out.println("Flying");
}
class SampleSingleInheritance
{
public static void main(String[] args)
{
Parrot p = new Parrot();
p.fly();
p.food();
}
}
```

Output

Grains

Flying

Multilevel Inheritance

“Multilevel inheritance” is the term used to describe a situation in which a derived class uses properties from its parent class as a base class for another derived class.

Example

```
class Bird

{

void food()

{

System.out.println(“Grains”);

}

class Parrot extends Bird

{

void fly()

{

System.out.println(“Flying”);

}

class Myna extends Parrot

{

void Sings()

{

System.out.println(“Singing”);
```

```
}

class SampleMultipleInheritance

{

public static void main(String[] args)

{

Myna m = new Myna();

m.sings()

m.fly();

m.food();

}

}
```

Output

Singing

Grains

Flying

Hierarchical Inheritance

Hierarchical inheritance has one base class and more derived classes that utilize the properties at the same level of authority.

Example:

```
class Bird

{

void food()
```

```
{
System.out.println(“Grains”);
}
class Parrot extends Bird
{
void fly()
{
System.out.println(“Flying”);
}
class Peacock extends Bird
{
void dance()
{
System.out.println(“Dancing”);
}
class SampleHierarchicalInheritance
{
public static void main(String[] args)
{
Parrot p = new Parrot();
```

```
p.fly();

p.food();

Peacock c = new Peacock();

c.dance();

c.food();

}

}
```

Output

Flying

Grains

Dancing

Grains

Polymorphism

If an object or method performs in various ways, it is referred to as polymorphism.

For example, a person is playing numerous roles, such as employee, family man, and student.

Example

public interface Human { }

public class Employee { }

public class Programmer extends Employee implements Human { }

The programmer class is considered to be polymorphic as it is has multiple inheritances, and it will be defined as follows:

A Programmer IS-A Employee

A Programmer IS-A Human

A Programmer IS-A Programmer

A Programmer IS-A Object

Example

```
class Bird

{

void food()

{

System.out.println("All Birds eat
Grains");

}

class Parrot extends Bird

{

void fly()

{

System.out.println("Parrot is
speaking");

}

class Peacock extends Bird

{

void dance()
```

```
{

System.out.println("Peacock is
Dancing");

}

class main

{

public static void main(String[] args)

{

Bird myBird = new Bird();

Bird myParrot = new Parrot();

Bird myPeacock = new Peacock();

}

}
```

Output

All Birds eat Grains

Parrot is speaking

Peacock is dancing

Abstraction

Abstraction is used to hide the background information and show the important functionality of a method or class. As we make a call to another number without knowing the internal processing completely.

Example

```
abstract class Bird

{

public abstract void BirdSound();

public void Sleep()

{

System.out.println("Zzz");

}

}

class Dove extends Bird

{

public void BirdColor()

{

System.out.println("The dove is in white
and grey colors");

}

}

class Main

{

public static void main(String[] args)
```

```
{

Dove myD = new Dove();

myD.BirdColor();

myD.Sleep();

}

}
```

Output

The dove is in white and grey colors

Zzz

Encapsulation

Wrapping of data into a single class or method is referred to as encapsulation, as a capsule contains two or more different medicines.

Uses of Encapsulation in Java

- Control the attributes and methods of classes effectively.
- Attributes of class can be set read-only or write-only using get and set methods.
- Developers can change any part of the program without affecting other parts, as it provides flexibility.
- Data security for all the properties of classes and methods.

Sample

```
public class Bird

{
```

```
private String name;

public String getName()

{

return name;

}

public void setName(String newName)

{

this.name = newName;

}

}

public class Main

{

public static void main(String[] args)

{

Bird myBird = new Bird();

myBird.name = “American Bird”

System.out.println(myBird.name);

}

}
```

Output

Main.java:5:error: name has private access in Bird

myBird.name = "American Bird";

Main.java7:error: name has private access in Bird

System.out.println(myBird.name); 2 Errors

Java Online Training

Other Java Concepts

In addition to these OOP ideas, terminology like coupling, cohesion, association, composition, and aggregation are also employed to finish off an object-oriented architecture.

- **Coupling:** When classes are aware of one another, coupling, which refers to data from another class, can be used.

The modifiers private, protected, and public in Java indicate the degree of accessibility and visibility for a class, field, and method, respectively.

- **Cohesion:** It is the degree to which a component completes a clearly defined task; this approach divides the task into several sections.

The most cohesive packages for accessing classes and interfaces related to input and output are java.IO and java.util.

- **Association:** Relationships between items are referred to as associations, and one object might have more than one association.

There are four methods of association: many to one, many to many, one to many, and one to one.

- **Aggregation:** Aggregation is the process of combining the states of other things into one object in order to accomplish association.

Reusing an object is referred to as "having a" and "is a" relationship.

- **Composition:** It symbolizes the relationship that includes the state of another object and is also used to achieve association.

The child object will immediately be removed if its parent object is deleted; the state does not exist independently.

OOPs concepts make development and maintenance simpler, quicker, and easier. It offers data masking so users can access global data from any location. The OOPs philosophy makes it possible to simulate the real world.

Enroll in our **J2EE training course** to learn enterprise-level software development.

Networking in Java

Java networking is an important thing that connects two or more devices together for sharing the resources within connected machines through Java sockets. It has two main advantages, such as sharing of resources and centralized software management.

Networking Terminology in Java

The networking technology of Java is popular and it contains IP addresses, protocols, port numbers, MAC addresses, connection-oriented and connection-less protocols, and sockets.

- **IP Addresses:** An IP address is a unique value assigned to a node or computer device of a network. *192.168.1.1 and it ranges from 0 to 255. This logical address can be changed.*
- **Protocols:** A protocol is a collection of rules followed for communication through *TCP (Transmission Control Protocol), FTP (File Transfer Protocol), Telnet, SMTP (Simple Mail Transfer Protocol), and POP (Post Office*

Protocol), etc.

- **Port Number:** It is used to identify various applications and it acts as a communication endpoint between applications. *It is associated with an IP address for interacting between computer devices.*
- **MAC Address:** It is the Media Access Control that identifies the *NIC (Network Interface Controller) and it can have multiple NICs* with a unique MAC.
- **The connection-oriented protocol** will be reliable but slow, as it acknowledges that every communication is sent by the receiver to the sender. *Ex: TCP*
- **The connection-less protocol** will be fast but not reliable, as no acknowledgement is sent by the receiver. *(Ex: UDP: User Datagram Protocol)*
- **Sockets** are endpoints that establish and maintain two-way communications.

Java.Net is the package that contains numerous classes for establishing network connections between two computing devices in Java programming.

Some of the popular used packages are Authenticator, CacheRequest, CacheResponse, CacheHandler, CacheManager, CookieManager, Datagram Pocket, Datagram Socket, InterfaceAddress, JavaURL Connection, Multicast Socket, InetAddress, Network Interface, Socket, ResponseCache, SecureCacheResponse, SocketPermission, URI, URL, URLEncoder, URLDecoder, URLStreamHandler, StandardSocketOptions, and URLConnection, and so on.

AWT in Java

Abstract Window Toolkit is an API that helps to develop GUI or window-based applications in Java Programming Language.

These components will be platform-independent and they will display as per the view of the operating system. AWT will be implemented in Java programming through the *java.awt* package that contains classes such as RadioButton, Choice, Label, TextField, List, TextArea, etc.

It has the following components:

- **Container** that contains components like text fields, buttons, labels, and so on.
- **Window** has no borders and menu bars. We can use this for dialog, frames, and so on.
- **Panel** is the place where we get title and menu bars.
- **Frame** is the container that has menu and title bars.

Methods for implementing AWT

public void add(Component c)

public void setSize(int width, int height)

public void setLayout(LayoutManager m)

public void setVisible(boolean status)

Example

```
import java.awt.*;

class Sample extends Frame

{

Sample()

Button b = new Button ("Click Here");

b.setBounds(30,100,70,30);

add(b);
```

```
setSize(300,300);

setLayout(null);

setVisible(true);

}

public static void main(String[] args)

{

Sample s = new Sample();

}

}
```

Output

Click Here

Swing in Java

Swing is one of the important concepts of Java Foundation Classes (JFC) used for developing window-based applications and it is built on top of the AWT API. But it provides lightweight components and platform-independent features for developing Java applications.

java.swing is the package that consists of classes for java swing with JButton, JTextField, JTextArea, JCheckbox, JRadioButton, JColorChooser, and JMenu, etc.

Java Swing follows MVC architecture and supports pluggable look and feel through powerful components such as lists, tables, scollpanes, tabbed panes, colorchoosers, etc.

Following are the commonly used methods of the

component class in the Swing package:

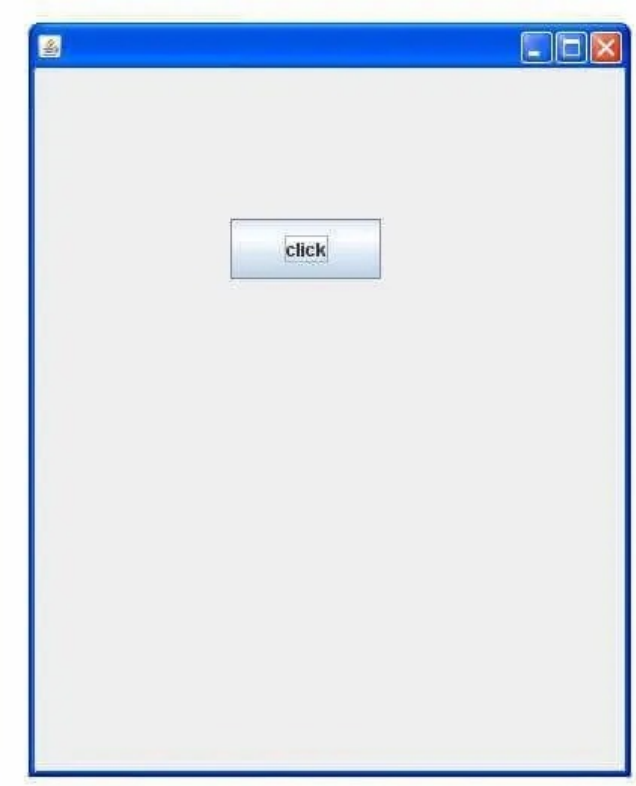
- public void add(Component c) to add a component on another component
- public void setSize(int width, int height) to set a size of the component
- public void setLayout(LayoutManager m) to set the layout manager for the component
- public void setVisible(boolean b) to set the visibility of the component with default as false.

Java Swing is used for drawing a frame in two ways, such as by creating the object of the Frame class (association) and by extending the Frame class (inheritance).

Example

```
import javax.swing.*;

public class SampleSwing

{

public static void main(String[] args)

{

  JFrame fr = new JFrame();

JButton bt = new JButton("Click Here");

bt.setBounds(200,150,150,50);

fr.add(bt);

fr.setSize(300,500);

fr.setLayout(null);

fr.setVisible(true);
```

```
}

}
```



Java Tutorial

Applet in Java

Applet of Java is used to create a program that is embedded in the web pages with dynamic content and it runs inside the browser at the client-side.

It has many advantages, such as less response time, security, and being platform-independent.

Hierarchy of Applet

Object → Component → Container → Panel → Applet → JApplet

Lifecycle of Applet

- Initialization of Applet
- Begin of Applet
- Applet is painted
- Stopping an Applet
- Destroy the Applet.

Java.applet.Applet Class

public void init() to initialize the applet that is invoked once.

public void start() to invoke after the init() method to start the applet.

public void stop() to stop the applet

public void destroy() to destroy the applet, which is invoked once.

Applet Java is executed in two ways, such as by HTML file and appletViewer tool.

Example

```
import java.applet.Applet;

import java.awt.Graphics;

public class SampleFirst extends Applet

{

public void paint(Graphics gr)

{

gr.drawString("Welcome", 100, 100);

}

}

myapplet.html

<html>

<applet code ="SampleFirst.class"
width = "250" height ="250">
```

```
</applet>

</body>

</html>
```

Output

c:\>javac Sample.java

c:\>appletviewer SampleFirst.java

Java plug-in software is required for managing the lifecycle of an Applet. This is considered the limitation of the Java Applet.

Reshape your app development skills with our **full-stack developer course** in Chennai.

Database Connection in Java

JDBC is the acronym for Java Database Connectivity. It connects a Java API to connect the database to execute a Java application. It is a part of JavaSE (Java Standard Edition) and JDBC drivers to connect with databases with four types of JDBC drivers as follows:

- JDBC-ODBC Bridge Driver
- Native Driver
- Network Protocol Driver, and
- The Driver

JDBC is based on an open SQL Call Level Interface. Java.sql is the package that contains interfaces and classes for JDBC API (Application Programming Interface).

The following is the list of popularly used interfaces of the JDBC API.

- Driver Interface
- Connection Interface
- Statement Interface
- PreparedStatement Interface

- CallableStatement Interface
- ResultSet Interface
- ResultSetMetaData Interface
- DatabaseMetadata Interface
- RowSet Interface

Following are the popularly used classes of the JDBC API:

- DriverManager Class
- Blob Class
- Clob class
- Types Class

JBDC is used to connect, update, and execute the database queries along with retrieving the result from the database.

Secure your career with in-demand IT skills through our **placement training institute** in Chennai.

Conclusion

We hope this Java tutorial will be helpful for beginners to understand the core and advanced concepts of Java. Gain expertise with our **Java training in Chennai** and start a career in software development.

Softlogic Systems

KK Nagar [Corporate Office]

No.10, PT Rajan Salai, K.K. Nagar, Chennai – 600 078.
Landmark: Karnataka Bank Building
Phone: +91 86818 84318
Email: enquiry@softlogicsys.in
Map: Google Maps Link

OMR

No. E1-A10, RTS Food Street
92, Rajiv Gandhi Salai (OMR),
Navalur, Chennai - 600 130.
Landmark: Adj. to AGS Cinemas
Phone: +91 89256 88858
Email: info@softlogicsys.in
Map: Google Maps Link

Courses

- Python
- Software Testing
- Full Stack Developer
- Java
- Power BI
- Clinical SAS
- Data Science
- Embedded
- Cloud Computing
- Hardware and Networking
- VBA Macros
- Mobile App Development
- DevOps

- Corporate Training
- Hire With Us
- Job Seekers
- SLA's Recently Placed Students
- Reviews
- Sitemap

Important Links

- Disclaimer
- Privacy Policy
- Terms and Conditions

Social Media Links

Review Sources

- Google
- Trustpilot
- Glassdoor
- Mouthshut
- Sulekha
- Justdial
- Ambitionbox
- Indeed
- Software Suggest
- Sitejabber