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

**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.

**Types of Java Applications**There are mainly 4 types of applications that can be created using Java programming:  
1) Standalone Application  
2) Web Application  
3) Enterprise Application  
4) Mobile Application

A list of the most important features of the Java language is given below.  
1)[Simple](https://www.javatpoint.com/features-of-java#Simple)  
2)[Object-Oriented](https://www.javatpoint.com/features-of-java#Object-Oriented)  
 Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior. Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.  
Basic concepts of OOPs are:

1. [Object](https://www.javatpoint.com/object-and-class-in-java)
2. [Class](https://www.javatpoint.com/object-and-class-in-java#class)
3. [Inheritance](https://www.javatpoint.com/inheritance-in-java)
4. [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
5. [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
6. [Encapsulation](https://www.javatpoint.com/encapsulation)

3)[Portable](https://www.javatpoint.com/features-of-java#Portable)  
4)[Platform independent](https://www.javatpoint.com/features-of-java#Platform-independent)  
There are two types of platforms software-based and hardware-based. Java provides a software-based platform. The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API (Application Programming Interface)

Java code can be executed on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere (WORA).

5)[Secured](https://www.javatpoint.com/features-of-java#Secured)  
Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

* No explicit pointer
* Java Programs run inside a virtual machine sandbox

  
6)[Robust](https://www.javatpoint.com/features-of-java#Robust)  
7)[Architecture neutral](https://www.javatpoint.com/features-of-java#Architecture-neutral)  
8)[Interpreted](https://www.javatpoint.com/features-of-java#Interpreted)  
9)[High Performance](https://www.javatpoint.com/features-of-java#High-Performance)  
10)[Multithreaded](https://www.javatpoint.com/features-of-java#Multithreaded)  
A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.  
11)[Distributed](https://www.javatpoint.com/features-of-java#Distributed)  
12)[Dynamic](https://www.javatpoint.com/features-of-java#Dynamic)

|  |  |
| --- | --- |
| **To compile:** | javac Simple.java –File name |
| **To execute:** | java Simple –Class name(Can be different from file name if class is not public) |

**public** **class** Hello\_World {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-gen'Hello erated method stub

System.***out***.println("Hello World. This is Java training.");

}

* **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](https://www.javatpoint.com/command-line-argument). We will discuss it in coming section.
* **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.

What happens at compile time?  
At compile time, the Java file is compiled by Java Compiler (It does not interact with OS) and converts the Java code into bytecode.

What happens at runtime?  
At runtime, the following steps are performed as per below.

* Classloader: It is the subsystem of JVM that is used to load class files.
* Bytecode Verifier: Checks the code fragments for illegal code that can violate access rights to objects.
* Interpreter: Read bytecode stream then execute the instructions.

 

**JVM (Java Virtual Machine)** is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.

JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each [OS](https://www.javatpoint.com/os-tutorial) is different from each other. However, Java is platform independent. There are three notions of the JVM.   
 1. A specification: where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle and other companies.  
 2. An implementation: Its implementation is known as JRE (Java Runtime Environment).  
 3. Runtime Instance: Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

The JVM performs the following main tasks:  
Loads code  
Verifies code  
Executes code  
Provides runtime environment.

**JRE** **(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.

**JDK (Java Development Kit)** is a software development environment which is used to develop Java applications and [applets](https://www.javatpoint.com/java-applet). It physically exists. It contains JRE + development tools. The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.



--------------------------------------------------------------------------------------------------------------------------

You write your code. Javac(java compiler) converts the code into bytecode. JVM runs the bytecode. (JVM is platform dependent since it can be created only for a specific OS). JVM runs only one file(it calls/contains all other files). If there are 100 files in your project, you need to mention the first file that JVM needs to run and that file should contain a main method. Main method has got a signature: public static void main(string[] args)

A blackboard with writing on it

Description automatically generated

System.out.print 🡪 prints  
System.out.println 🡪 prints and adds new line

# **Java Variables**

A variable is a container which holds the value while the [Java program](https://www.javatpoint.com/simple-program-of-java) is executed. A variable is assigned with a data type. A variable is the name of a reserved area allocated in memory. In other words, it is the name of the memory location. It is a combination of "vary + able" which means its value can be changed. There are three types of variables in java:  
  
Local: - declared inside a method of a class and can be used only by that method. Other methods in the class cannot use it. It cannot be static   
  
Instance: - A variable declared inside the class but outside the body of the method, is called an instance. variable.It is not declared as [static](https://www.javatpoint.com/static-keyword-in-java).

Static: - A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

# **Data Types** Data types specify the different sizes and values that can be stored in the variable. There are two types:

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](https://www.javatpoint.com/object-and-class-in-java), [Interfaces](https://www.javatpoint.com/interface-in-java), and [Arrays](https://www.javatpoint.com/array-in-java).

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Type** | **Default Value** | **Default size** | **Example** |
| boolean | false | 1 bit | **boolean** a=**true**; |
| char | '\u0000' | 2 byte | **char** c='A'; |
| byte | 0 | 1 byte(-128 to 127) | **byte** b=127; |
| short | 0 | 2 byte | **short** sh=42; |
| int | 0 | 4 byte | **int** i=104; |
| long | 0L | 8 byte | **long** len=5567l; |
| float | 0.0f | 4 byte | **float** f=10.5f; |
| double | 0.0d | 8 byte | **double** d=10.578; |

* In Java, default value is double as it has more precision i.e more numbers after decimal(eg 10.56789).  
  When you assign float num=5.6, it throws error as it takes 5.6 as double. For float, we need to explicitly mention it by float num=5.6f. And same with long(long num=5584l).
* Java follows UNICODE, not ASCII code. For all character values, mention them in single quotes and for string values, mention them in double quotes.
* Boolean is true or false(not 0 or 1). And they do not need to be in quotes as they are itself reserved key words.
* In Java, we can use increment operator for char as well. For example, char c=’a’ and c++. Now c value becomes ‘b’

**Conversion and Type Casting**

Imagine a small box and a large box. Small boxes always fit into larger one. We noneed to do anything extra. But, to fit a large into small, we need to use some force or widen the shorter one.  
Small data types converting into larger datatypes is automatic by Java i.e implicitly conversion is done and i.e Type conversion. But larger datatypes converting into smaller datatypes requires some effort explicitly and i.e Type Casting.

**class** Simple

{

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

{

//Conversion

**byte** a=124;

**int** b=a; //Int is large compared to byte hence implicit conversion

//Type casting

**byte** c=(**byte**)b; //Byte is small compared to int hence, explicit conversion

//Convert into byte first and then assign

//Widening

**int** d=12;

**float** e=d;//Implicit conversion from int to float

//Narrowing

**float** f=12.65f;

**int** g=(**int**)f; //explicit conversion

//Prints only the numbers before decimal

//Overflow

**int** h=130;

**byte** i=(**byte**)h;

//Since byte range is only -128 to 127.

//It will do a modulus operator with 256 and gives the remainder

//Here 130%256 is -126. 256 is the entire range of byte

//Addition

**byte** j=10;

**byte** k=20;

**int** result=j+k;//Addition of two bytes is always int

**byte** product=(**byte**)(j\*k);//Any arithmetic operation of bytes gives always int

System.***out***.println("Hi");//Hi

System.***out***.println(a);//124

System.***out***.println(b);//124

System.***out***.println(c);//124

System.***out***.println(d);//12

System.***out***.println(e);//12.0

System.***out***.println(f);//12.65

System.***out***.println(g);//12

System.***out***.println(h);//130

System.***out***.println(i);//-126

System.***out***.println(result);//30

System.***out***.println(product);//-56

}

}

**Arithmetic and Assignment Operators**

Java arithmetic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.

**class** Simple

{

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

{

**int** num1=15,num2=9;

**int** sum=num1+num2;

**int** result=num1-num2;

**int** product=num1\*num2;

**int** quotient=num1/num2;

**int** remainder=num1%num2;

System.***out***.println(sum);//24

System.***out***.println(result);//6

System.***out***.println(product);//135

System.***out***.println(quotient);//1

System.***out***.println(remainder);//6

**int** a=num1;

a=a+5;//Increment by 5

a+=5;//short cut for increment by 5

a-=5;//short cut for decrement

a++;//Increment by 1

a--;//Decrement by 1

System.***out***.println(a);//20

a\*=2;

System.***out***.println(a);//40

a/=2;

System.***out***.println(a);//20

**int** b=num2;//Assignment operator

System.***out***.println(b);//9

**int** c=num2++;//Post increment, assign to c and then increment the num2

System.***out***.println(c);//9

System.***out***.println(num2);//10

**int** d=++num2;//Pre increment, increment the num2 and then assign to d

System.***out***.println(d);//11

}

}

**Relational Operators**

**class** Simple

{

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

{

**int** a=7;

**float** b=7.8f;

**boolean** result=a>b;//greater than, false

System.***out***.println(result);

result=a<b;//less than, true

System.***out***.println(result);

result=a==b;//equal to, false

System.***out***.println(result);

result=a<=b;//less than or equal to, true

System.***out***.println(result);

result=a>=b;//greater than or equal to, false

System.***out***.println(result);

result=a!=b;//not equal to, true

System.***out***.println(result);

}

}

**Logical Operators**

AND, OR, NOT

Short circulating- (Use of two & or two |) It will check the first condition alone. Suppose in ‘And’ operator, if first condition is false, it will not check second and directly gives false and hence improves performance.

**class** Simple

{

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

{

**float** a=8.23f,b=9.96f;

**double** x=6.3,y=3.5;

**boolean** result;

result=a>b&x>y;

System.***out***.println(result);//and Operator bitwise, false

result=a>b&&x>y;

System.***out***.println(result);//Short circulating, false

result=a>b||x>y;

System.***out***.println(result);//Short circulating, true

**boolean** d=!result;

System.***out***.println(d);//Short circulating, false

System.***out***.println(!d);//true

}

}

**Control Statements**

Statements that can be used to control the flow of Java code.

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
   * for-each loop
3. Jump statements
   * break statement
   * continue statement

**If-Else Statement**

if-else statement tests the condition. It executes the if block if condition is true otherwise else block is executed.

**class** Simple

{

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

{

**i**f(**true**)

System.***out***.println("Welcome");

}

}

**----------------------------------------------------------**

//If only requires true or false either directly or through expression

**class** Simple

{

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

{

**int** num=10;

**if**(num%2==0)

System.***out***.println("Even");//One statement no need to put brackets

**else**

System.***out***.println("Odd");

}

}

**---------------------------------------------**

**class** Simple

{

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

{

**int** year=1900;

**if** ((year%4==0)&&(year%100!=0)||(year%400==0))

System.***out***.println("Leap");

**else**

System.***out***.println("Not Leap");

}

}

**If-Else-If Statement**

**class** Simple

{

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

{

**int** x=9,y=6,z=21;

**if**(x>y&&x>z)

System.***out***.println(x +" is greatest");

**else** **if**(y>z)

System.***out***.println(y +" is greatest");

**else**

System.***out***.println(z +" is greatest");

}

}

Output: 21 is greatest

**Ternary Operator**

Ternary operator is used as one line replacement for if-then-else statement. It is the only conditional operator which takes three operands.  
Condition?true statement:else-statement

**class** Simple

{

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

{

**int** a=10,b=7;

**int** min=a<b?a:b;

System.***out***.printf("%d is minimum of %d,%d",min,a,b);

}

}

**Switch Statement**

The Java *switch statement* executes one statement from multiple conditions. It tests the equality of a variable against multiple values.

### Points to Remember

* There can be *one or N number of case values* for a switch expression.
* The case value must be of switch expression type only. The case value must be *literal or constant*. It doesn't allow [variables](https://www.javatpoint.com/java-variables).
* The case values must be *unique*. In case of duplicate value, it renders compile-time error.
* The Java switch expression must be of *byte, short, int, long (with its Wrapper type),*[enums](https://www.javatpoint.com/java-switch)*and string*.
* Each case statement can have a *break statement* which is optional. When control reaches to the [break statement](https://www.javatpoint.com/java-break), it jumps the control after the switch expression. If a break statement is not found, it executes the next case.
* The case value can have a *default label* which is optional.

**Syntax:**switch(expression)

{      
 case value1:      
 //code to be executed;      
 break;  //optional  
 case value2:      
 //code to be executed;      
 break;  //optional      
--------  
default:  
code to be executed if all cases are not matched;

}

**class** Simple

{

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

{

String category="Regional", month="August";

**switch**(month)

{

**case** "January":

{ **switch**(category)

{

**case** "National":

System.***out***.println("Republic day");

**break**;

**case** "Regional":

System.***out***.println("Sankranthi");

**break**;

}

**break**;

}

**case** "August":

{ **switch**(category)

{

**case** "National":

System.***out***.println("Independence day");

**break**;

**case** "Regional":

System.***out***.println("Vinayak chavithi, Rakhi");

**break**;

}

**break**;

}

**case** "October":

{ **switch**(category)

{

**case** "National":

System.***out***.println("Gandhi Jayanthi");

**break**;

**case** "Regional":

System.***out***.println("Dussehra");

**break**;

}

**break**;

}

**default**:

System.***out***.println("Enter valid month");

}

}

}

**OVERFLOW WITHOUT BREAK**

**class** Simple

{

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

{

**char** month='T';

**switch** (month)

{

**case** 'M':

System.***out***.println("Monday");

**case** 'T':

System.***out***.println("Tuesday");

**case** 'W':

System.***out***.println("Wednesday");

**case** 'H':

System.***out***.println("Thursday");

**case** 'F':

System.***out***.println("Friday");

**case** 'R':

System.***out***.println("Saturday");

**case** 'S':

System.***out***.println("Sunday");

}

}

}

Output:  
Tuesday

Wednesday

Thursday

Friday

Saturday

Sunday

//The Java switch statement is fall-through. It means it executes all statements after the first match if a break statement is not present

**Updated Switch Without BREAK**We can use arrow instead of colon to avoid writing break after each case.

**class** Simple

{

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

{

**char** month='T';

**switch** (month)

{

**case** 'M'-> System.***out***.println("Monday");

**case** 'T'-> System.***out***.println("Tuesday");

**case** 'W'->System.***out***.println("Wednesday");

**case** 'H'->System.***out***.println("Thursday");

**case** 'F'->System.***out***.println("Friday");

**case** 'R'->System.***out***.println("Saturday");

**case** 'S'->System.***out***.println("Sunday");

}

}

}//Output:Tuesday

**Switch In Expression (Using Arrow)**

**class** Simple

{

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

{

**char** month='T';

String result;

result = **switch**(month)

{

**case** 'M'-> "Monday";

**case** 'T'-> "Tuesday";

**case** 'W'->"Wednesday";

**case** 'H'->"Thursday";

**case** 'F'->"Friday";

**case** 'R'->"Saturday";

**case** 'S'->"Sunday";

**default** ->"Enter valid";

};

System.***out***.println("Today is "+result );

}

}

Output: Today is Tuesday. When you are using expression, don’t forget to put semicolon after switch.

**Switch In Expression(Using colon)**

**class** Simple

{

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

{

**char** month='T';

String result;

result = **switch**(month)

{

**case** 'M': **yield** "Monday";

**case** 'T': **yield** "Tuesday";

**case** 'W':**yield** "Wednesday";

**case** 'H':**yield** "Thursday";

**case** 'F':**yield** "Friday";

**case** 'R':**yield** "Saturday";

**case** 'S': **yield**"Sunday";

**default** : **yield** "Enter valid";

};

System.***out***.println("Today is "+result );

}

}

### **Loop Statements**

In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true. However, loop statements are used to execute the set of instructions in a repeated order. The execution of the set of instructions depends upon a particular condition.

It consists of four parts: Initialization, Condition, Increment/decrement, statement.

**While Loop**

The while loop is considered as a repeating if statement. If the number of iterations is not fixed, it is recommended to use the while [loop](https://www.javatpoint.com/java-for-loop).

**class** Simple

{

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

{

**int** i=1;

**while** (i<=4)

{

**int** j=1;

System.***out***.println("Hi "+i);

**while**(j<=3)

{

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

j++;

}

i++;

}

}

}

Output:  
Hi 1

Hello

Hello

Hello

Hi 2

Hello

Hello

Hello

Hi 3

Hello

Hello

Hello

Hi 4

Hello

Hello

Hello

**Do While Loop**

Java do-while loop is called an **exit control loop**. Therefore, unlike while loop and for loop, the do-while check the condition at the end of loop body. The Java *do-while loop* is executed at least once because condition is checked after loop body.

**class** Simple

{

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

{

**int** i=5;

**do**

{

System.***out***.println("Hi "+i);

i++;

}**while** (i<=4);

}

}

Output:  
Hi 5

**For Loop**

The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is **fixed**, it is recommended to use for loop.

**class** Simple

{

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

{

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

{

System.***out***.println("Day "+i);

**for** (**int** j=9;j<=17;j++)

{

System.***out***.println(" "+j+"-"+(j+1));

}

}

}

}

Output:  
Day 1

9-10

10-11

11-12

12-13

13-14

14-15

15-16

16-17

17-18

Day 2

9-10

10-11

11-12

12-13

13-14

14-15

15-16

16-17

17-18

-----and so on upto Day 5

**Pyramid example**

**class** Simple

{

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

{

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

{

**for** (**int** j=1;j<=i;j++)

{

System.***out***.print("\*");

}

System.***out***.println();

}

}

}

Output:  
\*  
\*\*  
\*\*\*  
\*\*\*\*  
\*\*\*\*\*

**Reverse Pyramid**  
**class** Simple

{

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

{

**for** (**int** i=5;i>=1;i--)

{

**for** (**int** j=1;j<=i;j++)

{

System.***out***.print("\*");

}

System.***out***.println();

}

}

}

Output:

\*\*\*\*\*

\*\*\*\*

\*\*\*

\*\*

\*

**Infinite Loop**  
**class** Simple

{

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

{

**for**(;;)

{

System.***out***.println("infinitive loop");

}

}

}

Output:  
infinitive loop

infinitive loop

infinitive loop

-----------------

**Labelled for loop**

We can have a name of each Java for loop. To do so, we use label before the for loop. It is useful while using the nested for loop as we can break/continue specific for loop.

**class** Simple

{

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

{

aa:

**for**(**int** i=1;i<=3;i++)

{

bb:

**for**(**int** j=1;j<=3;j++)

{

**if** ((i==2)&&(j==2))

**break** bb;

System.***out***.println(i+" "+j);

}

}

}

}

Output:  
1 1

1 2

1 3

2 1

3 1

3 2

3 3

**Object Oriented Programming System**

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. An object has both properties and behavior. Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.  
Basic concepts of OOPs are:

1. [Object](https://www.javatpoint.com/object-and-class-in-java)
2. [Class](https://www.javatpoint.com/object-and-class-in-java#class)
3. [Inheritance](https://www.javatpoint.com/inheritance-in-java)
4. [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
5. [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
6. [Encapsulation](https://www.javatpoint.com/encapsulation)

We design a blueprint (class file) which is compiled into byte code and then using bytecode JVM creates object.

**Object:** Any entity that has a state and behavior is known as an object. An Object can be defined as an instance(result) of a class. An object contains an address and takes up some space in memory.

An object has three characteristics:

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

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

**Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical. Class doesn't consume any space.

A class in Java can contain:

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

**Naming Convention in Java**  
By using standard Java naming conventions, you make your code easier to read for yourself and other programmers. Readability of Java program is very important. It indicates that less time is spent to figure out what the code does.

|  |  |  |
| --- | --- | --- |
| **Identifiers Type** | **Naming Rules** | **Examples** |
| Class | It should start with the uppercase letter. It should be a noun such as Color, Button, System, Thread, etc. Use appropriate words, instead of acronyms. | public class **Employee** { //code snippet } |
| Interface | It should start with the uppercase letter. It should be an adjective such as Runnable, Remote, ActionListener. Use appropriate words, instead of acronyms. | interface **Printable** { //code snippet } |
| Method | It should start with lowercase letter. It should be a verb such as main(), print(), println(). If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed(). | class Employee { // method void **draw()** { //code snippet } } |
| Variable | It should start with a lowercase letter such as id, name. It should not start with the special characters like & (ampersand), $ (dollar), \_ (underscore). If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName. Avoid using one-character variables such as x, y, z. | class Employee { // variable int **id**; //code snippet } |
| Package | It should be a lowercase letter such as java, lang. If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang. | //package package **com.javatpoint;** class Employee { //code snippet } |
| Constant | It should be in uppercase letters such as RED, YELLOW. If the name contains multiple words, it should be separated by an underscore(\_) such as MAX\_PRIORITY. It may contain digits but not as the first letter. | class Employee { //constant static final int **MIN\_AGE** = 18; //code snippet } |

### 

### **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 Java, a method is like a function which is used to expose the behavior of an object. 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.  
Advantage of Method  
Code Reusability  
Code Optimization

**new keyword in Java**

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

### **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. members cannot be accessed (or viewed) from outside the class
* Protected: When we use protected access specifier, the method is accessible within the same package or subclasses in a different package. members cannot be accessed from outside the class, however, they can be accessed in inherited classes.
* 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. It may have a primitive data type, object, collection, void, etc. 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. Suppose, if we are creating a method for subtraction of two numbers, the method name must be **subtraction().** A method is invoked by its name.

**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 Method:**  
Single word always start with lower case. In multi words, lowercase first word, followed by second word with uppercase.  
Single-word method name: sum(), area() Multi-word method name: areaOfCircle(), stringComparision()

**Object and class example: Main within class**

**class** Simple

{

**int** id;

String name;

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

{

Simple s=**new** Simple();

System.***out***.println(s.id);

System.***out***.println(s.name);

}

}

Output:  
0

Null

**Object and class example: Main outside class //File name class should have main method**

**class** student

{

**int** id;

String name;

}

**class** Simple

{

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

{

student s=**new** student();

System.***out***.println(s.id);

System.***out***.println(s.name);

}

}

Output:  
0

Null

There are 3 ways to initialize an object. They are:  
1. By reference variable  
2. By method  
3. By constructor

**Initialization by reference**

**class** student

{

**int** id;

String name;

}

**class** Simple

{

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

{

student s1=**new** student();

student s2=**new** student();

s1.id=01;

s1.name="Advika";

s2.id=02;

s2.name="Aadhya";

System.***out***.println(s1.id+" "+s1.name);

System.***out***.println(s2.id+" "+s2.name);

}

}

Output:  
1 Advika

2 Aadhya

**Initialization by method**

**class** student

{

**int** id;

String name;

**void** insert(**int** id\_val,String name\_val)

{

id=id\_val;

name=name\_val;

}

**void** display()

{

System.***out***.println(id+" "+name);

}

}

**class** Simple

{

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

{

student s1=**new** student();

student s2=**new** student();

s1.insert(01,"Advika");

s2.insert(02,"Aadhya");

s1.display();

s2.display();

}

}

**Creating new methods**

**class** subtract//New class

{

int a;//instance variable

**public** **int** sub (**int** n1,**int** n2) //method is sub and returns a value🡪 int

{

**int** n3=n1-n2;

**return** n3;

}

}

**class** Simple

{

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

{

**int** num1=15,num2=4;

subtract s=**new** subtract();//create object

**int** result=s.sub(num1, num2);//using method

System.***out***.println(result);

}

}

//Java Program to demonstrate the working of a banking-system

//where we deposit and withdraw amount from our account.

//Creating an Account class which has deposit() and withdraw() methods

**class** Account{

**int** acc\_no;

String name;

**float** amount;

//Method to initialize object

**void** insert(**int** a,String n,**float** amt){

acc\_no=a;

name=n;

amount=amt;

}

//deposit method

**void** deposit(**float** amt){

amount=amount+amt;

System.***out***.println(amt+" deposited");

}

//withdraw method

**void** withdraw(**float** amt){

**if**(amount<amt){

System.***out***.println("Insufficient Balance");

}**else**{

amount=amount-amt;

System.***out***.println(amt+" withdrawn");

}

}

//method to check the balance of the account

**void** checkBalance(){System.***out***.println("Balance is: "+amount);}

//method to display the values of an object

**void** display(){System.***out***.println(acc\_no+" "+name+" "+amount);}

}

//Creating a test class to deposit and withdraw amount

**class** Simple{

**public** **static** **void** main(String[] args){

Account a1=**new** Account();

a1.insert(832345,"Ankit",1000);

a1.display();

a1.checkBalance();

a1.deposit(40000);

a1.checkBalance();

a1.withdraw(15000);

a1.checkBalance();

}}

**Using Scanner method**

//Addition program using two different classes

**import** java.util.\*;

**class** Addition

{

**public** **int** add(**int** num1, **int** num2)

{

**int** result=num1+num2;

**return** result;

}

}

**class** Simple

{

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

{

**int** a,b;

Scanner s= **new** Scanner(System.***in***);

Addition m=**new** Addition();

System.***out***.println("Enter the two numbers:");

a=s.nextInt();

b=s.nextInt();

**int** sum=m.add(a, b);

System.***out***.println(sum);

}

}

//Addition program using two different classes

**import** java.util.\*;

**class** Simple

{

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

{

**int** a;

Scanner s= **new** Scanner(System.***in***);

System.***out***.println("Enter the number:");

a=s.nextInt();

String result=*evenOdd*(a);

System.***out***.println(result);

}

**public** **static** String evenOdd(**int** num)

{

String r;

**if** (num%2==0)

r="Even";

**else**

r="Odd";

**return** r;

}

}