**Java Tutorial**

**What is Java?**

Java is a **programming language** and a **platform**.

Java is a high level, robust, secured and object-oriented programming language.

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

**Java Example**

Let's have a quick look at java programming example. A detailed description of hello java example is given in next page.

**class** Simple{

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

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

    }

}

## Where it is used?

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

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

## Types of Java Applications

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

#### 1) Standalone Application

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

#### 2) Web Application

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

#### 3) Enterprise Application

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

#### 4) Mobile Application

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

**History of Java**

1. Brief history of Java
2. Java Version History

**Java history** is interesting to know. The history of java starts from Green Team. Java team members (also known as **Green Team**), initiated a revolutionary task to develop a language for digital devices such as set-top boxes, televisions etc.

1) **James Gosling**, **Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.

2) Originally designed for small, embedded systems in electronic appliances like set-top boxes.

3) Firstly, it was called **"Greentalk"** by James Gosling and file extension was .gt.

4) After that, it was called **Oak** and was developed as a part of the Green project.

Why sun choosed "Oak" name?

5) **Why Oak?** Oak is a symbol of strength and choosen as a national tree of many countries like U.S.A., France, Germany, Romania etc.

6) In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.

# Features of Java

1. Features of Java
   1. Simple
   2. Object-Oriented
   3. Platform Independent
   4. secured
   5. Robust
   6. Architecture Neutral
   7. Portable
   8. High Performance
   9. Distributed
   10. Multi-threaded

There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.

1. Simple
2. Object-Oriented
3. Platform independent
4. Secured
5. Robust
6. Architecture neutral
7. Portable
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed

### Object-oriented

|  |
| --- |
| Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour. |
| Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules. |
| **Basic concepts of OOPs are:** |
| 1. Object 2. Class 3. Inheritance 4. Polymorphism 5. Abstraction 6. Encapsulation |

### Platform Independent

|  |
| --- |
| A platform is the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based. Java provides software-based platform. The Java platform differs from most other platforms in the sense that it's 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 is platform independentJava code can be run on multiple platforms e.g.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). |

### Secured

|  |
| --- |
| Java is secured because: |
| * No explicit pointer * Programs run inside virtual machine sandbox. |

|  |  |
| --- | --- |
| how java is secured | how java is secured |

|  |
| --- |
| * **Classloader-** adds security by separating the package for the classes of the local file system from those that are imported from network sources. * **Bytecode Verifier-** checks the code fragments for illegal code that can violate access right to objects. * **Security Manager-** determines what resources a class can access such as reading and writing to the local disk. |
| These security are provided by java language. Some security can also be provided by application developer through SSL,JAAS,cryptography etc. |

### Robust

|  |
| --- |
| Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust. |

### Architecture-neutral

|  |
| --- |
| There is no implementation dependent features e.g. size of primitive types is set. |

### Portable

|  |
| --- |
| We may carry the java bytecode to any platform. |

### Multi-threaded

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 shares the same memory. Threads are important for multi-media, Web applications etc.

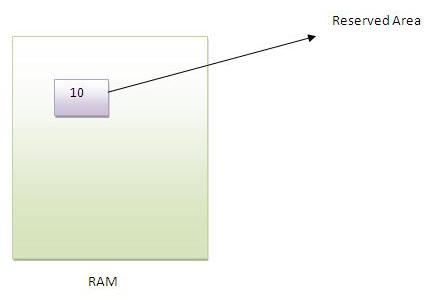
# Variable and Datatype in Java

1. Variable
2. Types of Variable
3. Data Types in Java

|  |
| --- |
| In this page, we will learn about the variable and java data types. Variable is a name of memory location. There are three types of variables: local, instance and static. There are two types of datatypes in java, primitive and non-primitive. |

### Variable

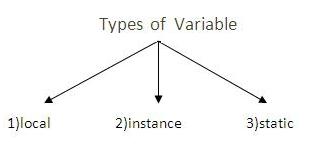
Variable is name of reserved area allocated in memory.



int data=50;//Here data is variable

### Types of Variable

|  |
| --- |
| There are three types of variables in java   * local variable * instance variable * static variable |



#### Local Variable

|  |
| --- |
| A variable that is declared inside the method is called local variable. |

#### Instance Variable

|  |
| --- |
| A variable that is declared inside the class but outside the method is called instance variable . It is not declared as static. |

#### Static variable

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

|  |
| --- |
| We will have detailed learning of these variables in next chapters. |

#### Example to understand the types of variables

class A{

int data=50;//instance variable

static int m=100;//static variable

void method(){

int n=90;//local variable

}

}//end of class

## Data Types in Java

|  |
| --- |
| In java, there are two types of data types   * primitive data types * non-primitive data types |



|  |  |  |
| --- | --- | --- |
| **Data Type** | **Default Value** | **Default size** |
| Boolean | false | 1 bit |
| Char | '\u0000' | 2 byte |
| Byte | 0 | 1 byte |
| Short | 0 | 2 byte |
| Int | 0 | 4 byte |
| Long | 0L | 8 byte |
| Float | 0.0f | 4 byte |
| Double | 0.0d | 8 byte |

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

because java uses unicode system rather than ASCII code system. \u0000 is the lowest range of unicode system.To get detail about Unicode see below.

# Operators in java

**Operator** in java is a symbol that is used to perform operations. There are many types of operators in java such as unary operator, arithmetic operator, relational operator, shift operator, bitwise operator, ternary operator and assignment operator.

|  |  |
| --- | --- |
| **Operators** | **Precedence** |
| Postfix | expr++ expr-- |
| Unary | ++expr --expr +expr -expr ~ ! |
| Multiplicative | \* / % |
| Additive | + - |
| Shift | << >> >>> |
| Relational | < > <= >= instanceof |
| Equality | == != |
| bitwise AND | & |
| bitwise exclusive OR | ^ |
| bitwise inclusive OR | | |
| logical AND | && |
| logical OR | || |
| Ternary | ? : |
| Assignment | = += -= \*= /= %= &= ^= |= <<= >>= >>>= |

**Java If-else Statement**

The Java *if statement* is used to test the condition. It returns *true* or *false*. There are various types of if statement in java.

* if statement
* if-else statement
* nested if statement
* if-else-if ladder

Java IF Statement

The if statement tests the condition. It executes the if statement if condition is true.

**Syntax:**

**if**(condition){

//code to be executed

}

**Example:**

**public** **class** IfExample {

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

**int** age=20;

**if**(age>18){

        System.out.print("Age is greater than 18");

    }

}

}

Output:

Age is greater than 18

**Java IF-else Statement**

The if-else statement also tests the condition. It executes the *if block* if condition is true otherwise *else block*.

**Syntax:**

**if**(condition){

//code if condition is true

}**else**{

//code if condition is false

}

**Example:**

**public** **class** IfElseExample {

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

**int** number=13;

**if**(number%2==0){

        System.out.println("even number");

    }**else**{

        System.out.println("odd number");

    }

}

}

Output:

odd number

**Java IF-else-if ladder Statement**

The if-else-if ladder statement executes one condition from multiple statements.

**Syntax:**

**if**(condition1){

//code to be executed if condition1 is true

}**else** **if**(condition2){

//code to be executed if condition2 is true

}

**else** **if**(condition3){

//code to be executed if condition3 is true

}

**else**{

//code to be executed if all the conditions are false

}

**Example:**

**public** **class** IfElseIfExample {

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

**int** marks=65;

**if**(marks<50){

        System.out.println("fail");

    }

**else** **if**(marks>=50 && marks<60){

        System.out.println("D grade");

    }

**else** **if**(marks>=60 && marks<70){

        System.out.println("C grade");

    }

**else** **if**(marks>=70 && marks<80){

        System.out.println("B grade");

    }

**else** **if**(marks>=80 && marks<90){

        System.out.println("A grade");

    }**else** **if**(marks>=90 && marks<100){

        System.out.println("A+ grade");

    }**else**{

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

    }

}

}

Output:

C grade

**Java Switch Statement**

The Java *switch statement* is executes one statement from multiple conditions. It is like if-else-if ladder statement.

**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;

}

**Example:**

**public** **class** SwitchExample {

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

**int** number=20;

**switch**(number){

**case** 10: System.out.println("10");**break**;

**case** 20: System.out.println("20");**break**;

**case** 30: System.out.println("30");**break**;

**default**:System.out.println("Not in 10, 20 or 30");

    }

}

}

Output:

20

**Java Switch Statement is fall-through**

The java switch statement is fall-through. It means it executes all statement after first match if break statement is not used with switch cases.

**Example:**

**public** **class** SwitchExample2 {

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

**int** number=20;

**switch**(number){

**case** 10: System.out.println("10");

**case** 20: System.out.println("20");

**case** 30: System.out.println("30");

**default**:System.out.println("Not in 10, 20 or 30");

    }

}

}

Output:

20

30

Not in 10, 20 or 30

**Java 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.

There are three types of for loop in java.

* Simple For Loop
* For-each or Enhanced For Loop
* Labeled For Loop

**Java Simple For Loop**

The simple for loop is same as C/C++. We can initialize variable, check condition and increment/decrement value.

**Syntax:**

**for**(initialization;condition;incr/decr){

//code to be executed

}

**Example:**

**public** **class** ForExample {

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

**for**(**int** i=1;i<=10;i++){

        System.out.println(i);

    }

}

}

Output:

1

2

3

4

5

6

7

8

9

10

**Java For-each Loop**

The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.

It works on elements basis not index. It returns element one by one in the defined variable.

**Syntax:**

**for**(Type var:array){

//code to be executed

}

**Example:**

**public** **class** ForEachExample {

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

**int** arr[]={12,23,44,56,78};

**for**(**int** i:arr){

        System.out.println(i);

    }

}

}

Output:

12

23

44

56

78

**Java While Loop**

The Java *while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

**Syntax:**

**while**(condition){

//code to be executed

}

**Example:**

**public** **class** WhileExample {

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

**int** i=1;

**while**(i<=10){

        System.out.println(i);

    i++;

    }

}

}

Output:

1

2

3

4

5

6

7

8

9

10

**Java Infinitive While Loop**

If you pass **true** in the while loop, it will be infinitive while loop.

**Syntax:**

**while**(**true**){

//code to be executed

}

**Example:**

**public** **class** WhileExample2 {

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

**while**(**true**){

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

    }

}

}

Output:

infinitive while loop

infinitive while loop

infinitive while loop

infinitive while loop

infinitive while loop

ctrl+c

**Java do-while Loop**

The Java *do-while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use while loop.

It is executed at least once because condition is checked after loop body.

**Syntax:**

**do**{

//code to be executed

}**while**(condition);

**Example:**

**public** **class** DoWhileExample {

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

**int** i=1;

**do**{

        System.out.println(i);

    i++;

    }**while**(i<=10);

}

}

Output:

1

2

3

4

5

6

7

8

9

10

**Java Infinitive do-while Loop**

If you pass **true** in the do-while loop, it will be infinitive do-while loop.

**Syntax:**

**while**(**true**){

//code to be executed

}

**Example:**

**public** **class** DoWhileExample2 {

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

**do**{

        System.out.println("infinitive do while loop");

    }**while**(**true**);

}

}

Output:

infinitive do while loop

infinitive do while loop

infinitive do while loop

ctrl+c

**Java Break Statement**

The Java *break* is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

**Syntax:**

1. jump-statement;
2. **break**;

**Java Break Statement with Loop**

**Example:**

**public** **class** BreakExample {

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

**for**(**int** i=1;i<=10;i++){

**if**(i==5){

**break**;

        }

        System.out.println(i);

    }

}

}

Output:

1

2

3

4

**Java Break Statement with Inner Loop**

It breaks inner loop only if you use break statement inside the inner loop.

**Example:**

**public** **class** BreakExample2 {

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

**for**(**int** i=1;i<=3;i++){

**for**(**int** j=1;j<=3;j++){

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

**break**;

                        }

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

                    }

            }

}

}

Output:

1 1

1 2

1 3

2 1

3 1

3 2

3 3

**Java Break Statement with Switch**

To understand the example of break with switch statement, please visit here: Java Switch Statement.

**Java Continue Statement**

The Java *continue statement* is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop.

**Syntax:**

jump-statement;

**continue**;

**Java Continue Statement Example**

**Example:**

**public** **class** ContinueExample {

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

**for**(**int** i=1;i<=10;i++){

**if**(i==5){

**continue**;

        }

        System.out.println(i);

    }

}

}

Output:

1

2

3

4

6

7

8

9

10

**Java Continue Statement with Inner Loop**

It continues inner loop only if you use continue statement inside the inner loop.

**Example:**

**public** **class** ContinueExample2 {

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

**for**(**int** i=1;i<=3;i++){

**for**(**int** j=1;j<=3;j++){

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

**continue**;

                        }

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

                    }

            }

}

}

Output:

1 1

1 2

1 3

2 1

2 3

3 1

3 2

3 3