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");

}

}

Difference between JDK, JRE and JVM::::

Understanding the difference between JDK, JRE and JVM is important in Java. We are having brief

overview of JVM here.

JVM

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.

The JVM performs following main tasks:

Loads code

Verifies code

Executes code

Provides runtime environment

JRE

JRE is an acronym for Java Runtime Environment.It is used to provide runtime environment.It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.

Implementation of JVMs are also actively released by other companies besides Sun Micro Systems.

JDK

JDK is an acronym for Java Development Kit.It physically exists.It contains JRE + development tools.

JVM (Java Virtual Machine)

Java Virtual Machine

Internal Architecture of JVM

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

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

JVM Archtecture

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

1) Classloader

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

2) Class(Method) Area

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

3) Heap

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

4) Stack:::

Java Stack stores frames.It holds local variables and partial results, and plays a part in

method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

5) Program Counter Register

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

6) Native Method Stack

It contains all the native methods used in the application.

7) Execution Engine

It contains:

1) A virtual processor

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

3) Just-In-Time(JIT) compiler: It is used to improve the performance.JIT compiles parts of the

byte code that have similar functionality at the same time, and hence reduces the amount of time

needed for compilation.Here the term ?compiler? refers to a translator from the instruction set of

a Java virtual machine (JVM) to the instruction set of a specific CPU.

Variables and Data Types in Java

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

There are two types of data types in java: primitive and non-primitive.

Variable

**Variable** is name of *reserved area allocated in memory*. In other words, it is a *name of memory location*. It is a combination of "vary + able" that means its value can be changed.

**int** data=50;//Here data is variable .

#### 1) Local Variable

A variable which is declared inside the method is called local variable.

#### 2) Instance Variable

A variable which is declared inside the class but outside the method, is called instance variable . It is not declared as static.

#### 3) 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.

Data Types in Java

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

* Primitive data types
* Non-primitive data types

## Java Arrays

Arrays are objects that store multiple variables of the same type. However, an array itself is an object on the heap. We will look into how to declare, construct, and initialize in the upcoming chapters.

## Java Keywords

The following list shows the reserved words in Java. These reserved words may not be used as constant or variable or any other identifier names.

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

## Comments in Java

## Constructors

When discussing about classes, one of the most important sub topic would be constructors. Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.

Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

Following is an example of a constructor −

### Example

public class Puppy {

public Puppy() {

}

public Puppy(String name) {

// This constructor has one parameter, *name*.

}

}

Operators in java

**Operator** in java is a symbol that is used to perform operations. For example: +, -, \*, / etc.

There are many types of operators in java which are given below:

* Unary Operator,
* Arithmetic Operator,
* shift Operator,
* Relational Operator,
* Bitwise Operator,
* Logical Operator,
* Ternary Operator and
* Assignment Operator.

|  |  |  |
| --- | --- | --- |
| **Operator Type** | **Category** | **Precedence** |
| Unary | postfix | *expr*++ *expr*-- |
| prefix | ++*expr* --*expr* +*expr* -*expr* ~ ! |
| Arithmetic | multiplicative | \* / % |
| additive | + - |
| Shift | shift | << >> >>> |
| Relational | comparison | < > <= >= instanceof |
| equality | == != |
| Bitwise | bitwise AND | & |
| bitwise exclusive OR | ^ |
| bitwise inclusive OR | | |
| Logical | logical AND | && |
| logical OR | || |
| Ternary | ternary | ? : |
| Assignment | assignment | = += -= \*= /= %= &= ^= |= <<= >>= >>>= |

# Creting object

## String Methods

Here is the list of methods supported by String class −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | [**char charAt(int index)**](https://www.tutorialspoint.com/java/java_string_charat.htm)  Returns the character at the specified index. |
| 2 | [**int compareTo(Object o)**](https://www.tutorialspoint.com/java/java_string_compareto.htm)  Compares this String to another Object. |
| 3 | [**int compareTo(String anotherString)**](https://www.tutorialspoint.com/java/java_string_compareto_anotherstring.htm)  Compares two strings lexicographically. |
| 4 | [**int compareToIgnoreCase(String str)**](https://www.tutorialspoint.com/java/java_string_comparetoignorecase.htm)  Compares two strings lexicographically, ignoring case differences. |
| 5 | [**String concat(String str)**](https://www.tutorialspoint.com/java/java_string_concat.htm)  Concatenates the specified string to the end of this string. |
| 6 | [**boolean contentEquals(StringBuffer sb)**](https://www.tutorialspoint.com/java/java_string_contentequals.htm)  Returns true if and only if this String represents the same sequence of characters as the specified StringBuffer. |
| 7 | [**static String copyValueOf(char[] data)**](https://www.tutorialspoint.com/java/java_string_copyvalueof.htm)  Returns a String that represents the character sequence in the array specified. |
| 8 | [**static String copyValueOf(char[] data, int offset, int count)**](https://www.tutorialspoint.com/java/java_string_copyvalueof_dataoffsetcount.htm)  Returns a String that represents the character sequence in the array specified. |
| 9 | [**boolean endsWith(String suffix)**](https://www.tutorialspoint.com/java/java_string_endswith.htm)  Tests if this string ends with the specified suffix. |
| 10 | [**boolean equals(Object anObject)**](https://www.tutorialspoint.com/java/java_string_equals.htm)  Compares this string to the specified object. |
| 11 | [**boolean equalsIgnoreCase(String anotherString)**](https://www.tutorialspoint.com/java/java_string_equalsignorecase.htm)  Compares this String to another String, ignoring case considerations. |
| 12 | [**byte getBytes()**](https://www.tutorialspoint.com/java/java_string_getbytes.htm)  Encodes this String into a sequence of bytes using the platform's default charset, storing the result into a new byte array. |
| 13 | [**byte[] getBytes(String charsetName)**](https://www.tutorialspoint.com/java/java_string_getbytes_charsetname.htm)  Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array. |
| 14 | [**void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)**](https://www.tutorialspoint.com/java/java_string_getchars.htm)  Copies characters from this string into the destination character array. |
| 15 | [**int hashCode()**](https://www.tutorialspoint.com/java/java_string_hashcode.htm)  Returns a hash code for this string. |
| 16 | [**int indexOf(int ch)**](https://www.tutorialspoint.com/java/java_string_indexof.htm)  Returns the index within this string of the first occurrence of the specified character. |
| 17 | [**int indexOf(int ch, int fromIndex)**](https://www.tutorialspoint.com/java/java_string_indexof_fromindex.htm)  Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index. |
| 18 | [**int indexOf(String str)**](https://www.tutorialspoint.com/java/java_string_indexof_str.htm)  Returns the index within this string of the first occurrence of the specified substring. |
| 19 | [**int indexOf(String str, int fromIndex)**](https://www.tutorialspoint.com/java/java_string_indexof_strfromindex.htm)  Returns the index within this string of the first occurrence of the specified substring, starting at the specified index. |
| 20 | [**String intern()**](https://www.tutorialspoint.com/java/java_string_intern.htm)  Returns a canonical representation for the string object. |
| 21 | [**int lastIndexOf(int ch)**](https://www.tutorialspoint.com/java/java_string_lastindexof.htm)  Returns the index within this string of the last occurrence of the specified character. |
| 22 | [**int lastIndexOf(int ch, int fromIndex)**](https://www.tutorialspoint.com/java/java_string_lastindexof_chfromindex.htm)  Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index. |
| 23 | [**int lastIndexOf(String str)**](https://www.tutorialspoint.com/java/java_string_lastindexof_str.htm)  Returns the index within this string of the rightmost occurrence of the specified substring. |
| 24 | [**int lastIndexOf(String str, int fromIndex)**](https://www.tutorialspoint.com/java/java_string_lastindexof_fromindex.htm)  Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index. |
| 25 | [**int length()**](https://www.tutorialspoint.com/java/java_string_length.htm)  Returns the length of this string. |
| 26 | [**boolean matches(String regex)**](https://www.tutorialspoint.com/java/java_string_matches.htm)  Tells whether or not this string matches the given regular expression. |
| 27 | [**boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)**](https://www.tutorialspoint.com/java/java_string_regionmatches_ignorecase.htm)  Tests if two string regions are equal. |
| 28 | [**boolean regionMatches(int toffset, String other, int ooffset, int len)**](https://www.tutorialspoint.com/java/java_string_regionmatches.htm)  Tests if two string regions are equal. |
| 29 | [**String replace(char oldChar, char newChar)**](https://www.tutorialspoint.com/java/java_string_replace.htm)  Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar. |
| 30 | [**String replaceAll(String regex, String replacement**](https://www.tutorialspoint.com/java/java_string_replaceall.htm)  Replaces each substring of this string that matches the given regular expression with the given replacement. |
| 31 | [**String replaceFirst(String regex, String replacement)**](https://www.tutorialspoint.com/java/java_string_replacefirst.htm)  Replaces the first substring of this string that matches the given regular expression with the given replacement. |
| 32 | [**String[] split(String regex)**](https://www.tutorialspoint.com/java/java_string_split.htm)  Splits this string around matches of the given regular expression. |
| 33 | [**String[] split(String regex, int limit)**](https://www.tutorialspoint.com/java/java_string_split_regexlimit.htm)  Splits this string around matches of the given regular expression. |
| 34 | [**boolean startsWith(String prefix)**](https://www.tutorialspoint.com/java/java_string_startswith.htm)  Tests if this string starts with the specified prefix. |
| 35 | [**boolean startsWith(String prefix, int toffset)**](https://www.tutorialspoint.com/java/java_string_startswith_prefixtoffset.htm)  Tests if this string starts with the specified prefix beginning a specified index. |
| 36 | [**CharSequence subSequence(int beginIndex, int endIndex)**](https://www.tutorialspoint.com/java/java_string_subsequence.htm)  Returns a new character sequence that is a subsequence of this sequence. |
| 37 | [**String substring(int beginIndex)**](https://www.tutorialspoint.com/java/java_string_substring.htm)  Returns a new string that is a substring of this string. |
| 38 | [**String substring(int beginIndex, int endIndex)**](https://www.tutorialspoint.com/java/java_string_substring_beginendindex.htm)  Returns a new string that is a substring of this string. |
| 39 | [**char[] toCharArray()**](https://www.tutorialspoint.com/java/java_string_tochararray.htm)  Converts this string to a new character array. |
| 40 | [**String toLowerCase()**](https://www.tutorialspoint.com/java/java_string_tolowercase.htm)  Converts all of the characters in this String to lower case using the rules of the default locale. |
| 41 | [**String toLowerCase(Locale locale)**](https://www.tutorialspoint.com/java/java_string_tolowercase_locale.htm)  Converts all of the characters in this String to lower case using the rules of the given Locale. |
| 42 | [**String toString()**](https://www.tutorialspoint.com/java/java_string_tostring.htm)  This object (which is already a string!) is itself returned. |
| 43 | [**String toUpperCase()**](https://www.tutorialspoint.com/java/java_string_touppercase.htm)  Converts all of the characters in this String to upper case using the rules of the default locale. |
| 44 | [**String toUpperCase(Locale locale)**](https://www.tutorialspoint.com/java/java_string_touppercase_locale.htm)  Converts all of the characters in this String to upper case using the rules of the given Locale. |
| 45 | [**String trim()**](https://www.tutorialspoint.com/java/java_string_trim.htm)  Returns a copy of the string, with leading and trailing whitespace omitted. |
| 46 | [**static String valueOf(primitive data type x)**](https://www.tutorialspoint.com/java/java_string_valueof.htm)  Returns the string representation of the passed data type argument. |

# Java Switch Statement

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

**Syntax:**

1. **switch**(expression){
2. **case** value1:
3. //code to be executed;
4. **break**;  //optional
5. **case** value2:
6. //code to be executed;
7. **break**;  //optional
8. ......
10. **default**:
11. code to be executed **if** all cases are not matched;
12. }

Java IF Statement

The Java if statement tests the condition. It executes the *if block* if condition is true.

**Syntax:**

1. **if**(condition){
2. //code to be executed
3. }

Example:

public class IfExample {

public static void main(String[] args) {

int age=20;

if(age>18){

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

}

}

}

## Java IF-else Statement

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

**Syntax:**

1. **if**(condition){
2. //code if condition is true
3. }**else**{
4. //code if condition is false
5. }