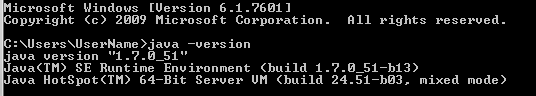
JAVA

# Is Java installed in your system?

Execute the command **java -version** in command window.

If Java is configured it will show the Java version details as below



If Java is not configured it will give message:

'java' is not recognized as an internal or external command,

operable program or batch file.

Then need to configure Java:

1. Install Java JDK
2. Set environment variables: steps to set java environment path is given in the below link.

<http://www.javatpoint.com/how-to-set-path-in-java>

# Java Examples

## Example\_01: *HelloWorld.java*

Open a text editor (Notepad or Notepad++) and type the below code, and save the file as **HelloWorld.java**

NOTE: the file name should be same as the name of Java class (case sensitive), in the below code the class name is HelloWorld and the file name has to be HelloWorld.java

/\*

This is a simple Java program.

Call this file "HelloWorld.java".

\*/

**class** HelloWorld{

// Your program begins with a call to main().

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

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

}

}

**Compiling the program:**

Open the command window in the folder where you have saved the HelloWorld.java file, and run the command javac HelloWorld.java.

Tips: Steps to open command window in the specified folder,

1. Go to the folder
2. SHIFT + Right Click
3. Select ‘Open command window here’

C:\>javac HelloWorld.java

Note: if you have installed JRE (Java runtime environment) in your system, you will not be able to compile the java code, since JRE can only run the java bytecode (.class files) but cannot compile the java code. To compile the java code you must have setup JDK into your system.

Once the file is compiled you can find the new file **HelloWorld.class** generated in the folder, this is the intermediate bytecode that can be run on any system (“*Compile once and run anywhere”*).

**Run the program:**

To run this file type the below command (java HelloWorld) in the command window.

C:\>java HelloWorld

## Example\_02: *Example2.java*

/\*

Call this file "Example2.java".

\*/

**class** Example2 {

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

**int** num; // this declares a variable called num

num = 100; // this assigns num the value 100

System.*out*.println("This is num: " + num);

num = num \* 2;

System.*out*.print("The value of num \* 2 is ");

System.*out*.println(num);

}

}

You can compile and run this code using below commands,

C:\>javac Example2.java

C:\>java Example2

Similarly you can compile any Java code using the specified file names.

As the complexity of the program increases you may require packaging and structuring the code, for this purpose we have IDEs (Integrated development environments), that simplifies some of the work.

There are many Editors and IDEs that can be used to run java program, one such widely used IDE for java is “Eclipse” IDE.

# Install Eclipse

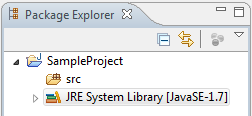
1. Download eclipse IDE: <https://eclipse.org/downloads/>
2. Start Eclipse by double clicking eclipse application file.

## Creating java Project

In eclipse,

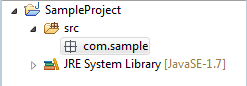
* Go to: File > New > Project…
* Select Java Project and Click Next.
* Give project name (any name), click finish.

New project will be created with “**src”** folder in it.

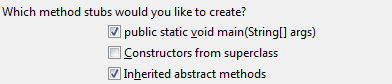


## Creating java class

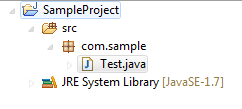
* Right click on “**src”** folder > New > Package
* Give package name(Ex: com.sample)



* Right click on package(com.sample) > New > Class
* Give class Name(Ex: Test)
* Select *public static void main(String[] arg)* check box, as shown below, to add main method to class.



* Click finish.
* It will create java class with main method.



You can run the above mentioned examples using eclipse as follow,

* Create new java class
* Edit the code (Ctrl + S to save the code).
* **Right Click** on the source code editor window and select **Run As**, then run as **Java Application.**

Output of the program will be displayed on the eclipse console window.

# Java Keywords

Here is a list of keywords in the Java programming language. You cannot use any of the following as identifiers in your programs. The keywords const and goto are reserved, even though they are not currently used. true, false, and null might seem like keywords, but they are actually literals; you cannot use them as identifiers in your programs.

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

\* Not used

\*\* added in 1.2

\*\*\* added in 1.4

\*\*\*\* added in 5.0

# Datatypes

Java defines eight *primitive* types of data: **byte**, **short**, **int**, **long**, **char**, **float**, **double**, and **boolean**. The primitive types are also commonly referred to as *simple* types.

|  |  |  |
| --- | --- | --- |
| Type | Explanation | Range |
| byte | An 8-bit (1-byte) integer value | –128 to 127 |
| short | A 16-bit (2-byte) integer value | –32,768 to 32,767 |
| int | A 32-bit (4-byte) integer value | –2,147,483,648 to 2,147,483,647 |
| long | A 64-bit (8-byte) integer value | –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| float | A 32-bit (4-byte) floating-point value | 1.4e–045 to 3.4e+038 |
| double | A 64-bit (8-byte) floating-point value | 4.9e–324 to 1.8e+308 |
| char | A 16-bit character using the Unicode encoding scheme | 0 to 65535 |
| boolean | A true or false value |  |

String in java is an object type (java.lang.String class).

|  |  |  |
| --- | --- | --- |
| String | A sequence of characters |  |

# Control Statements

## Selection Statements

### if

Syntax:

if (*condition*) *statement1*;

else *statement2*;

Here, each *statement* may be a single statement or a compound statement enclosed in curly braces (that is, a *block*). *Else* is optional

EX:

boolean *fileExists*;

//..

if(*fileExists*){

//update file

} else {

// create new file

}

### if-else-if Ladder

Syntax:

if(*condition*)

*statement*;

else if(*condition*)

*statement*;

else if(*condition*)

*statement*;

.

.

.

else

*statement*;

EX:

**public** **class** IfElseExample {

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

**float** score = 59.9f;

String grade;

**if** (score >= 90) {

grade = "A";

} **else** **if** (score < 90 && score >= 75) {

grade = "B";

} **else** **if**(score < 75 && score >= 60){

grade = "C";

} **else** **if**(score < 60 && score >= 40){

grade = "D";

} **else**{

grade = "E";

}

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

}

}

### switch

Switch statements are used when comparing for equality.

Syntax:

switch (*expression*) {

case *value1*:

// statement sequence

break;

case *value2*:

// statement sequence

break;

.

.

.

case *valueN* :

// statement sequence

break;

default:

// default statement sequence

}

NOTE: Beginning with JDK 7, expression can also be of type String, to run below example you need JDK 1.7 or above.

EX:

**public** **class** SwitchExample {

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

String grade = "E";

String result = **null**;

**switch**(grade){

**case** "A":

**case** "B":

**case** "C":

**case** "D":

result = "Pass";

**break**;

**case** "E":

result = "Fail";

**break**;

}

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

}

}

You can observe that break statement is skipped for the first four cases, to result in same output.

## Iteration Statements

### while

Syntax:

while(*condition*) {

// body of loop

}

EX:

**public** **class** WhileExample {

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

**int** i = 0;

**while** (i < 10) {

System.***out***.println("value of i is: " + i);

i++;

}

}

}

### do-while

Syntax:

do {

// body of loop

} while (*condition*);

Ex:

**public** **class** DoWhileExample {

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

String line = "aaaaa\n";

StringBuffer sb = **new** StringBuffer();

**int** i = 0;

**char** c;

// read characters in a line;

**do** {

c = line.charAt(i);

sb.append(c);

i++;

} **while** (c != '\n');

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

}

}

### for

Syntax:

for(*initialization*; *condition*; *iteration*) {

// body

}

Ex:

**public** **class** ForLoopExample {

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

/\*

\* Below sequence shows different ways a character can be declared in Java

\* 'A' - character 'A'

\* 66 - decimal value for character 'B'

\* 0103 - octal value for character 'C'

\* 0X44 - hexadecimal value for character 'D'

\* \u0045 - Unicode value for character 'E'

\*/

**char**[] sequence = { 'A', 66, 0103, 0X44, '\u0045' };

// loop until the last character in the array

// output will be ABCDE

**for** (**int** i = 0; i < sequence.length; i++) {

System.***out***.print(sequence[i]);

}

}

}

Classes, Objects and Methods

# Class

The class is at the core of Java, and the most important thing to understand about a class is that it defines a new data type.

An object is an *instance* of a class.

Class can be represented as below,

class classname {

*type instance-variable1;*

*type instance-variable2;*

// ...

*type instance-variableN;*

*type methodname1(parameter-list) {*

// body of method

*}*

// ...

*type methodnameN(parameter-list) {*

// body of method

*}*

*}*

Ex: class without constructor.

**public** **class** Rectangle {

**float** length;

**float** breadth;

// method to return Area of rectangle

**public** **float** getAreaOfRectangle() {

**return** length \* breadth;

}

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

// instantiating Rectangle class and setting instance variable values.

Rectangle rectangle = **new** Rectangle();

rectangle.length =10;

rectangle.breadth = 10;

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

}

}

Ex: class with a constructor.

**public** **class** Rectangle {

**float** length;

**float** breadth;

// constructor

Rectangle(**float** length, **float** breadth) {

**this**.length = length;

**this**.breadth = breadth;

}

// method to return Area of rectangle

**public** **float** getAreaOfRectangle() {

**return** length \* breadth;

}

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

Rectangle rectangle = **new** Rectangle(10, 10);

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

}

}

## Access Modifier:

default : Visible to the package. No modifiers are needed. (in the above example access type to constructor is a default type)

private : Visible to the class only.

public : Visible to the world.

protected : Visible to the package and all subclasses.

## Overloading:

The process to define **two or more methods** within the same class that share the **same name**, as long as their parameter declarations are different. (Polymorphism)

Ex:

**public** **class** Calculate {

**public** **int** sum(**int** a, **int** b) {

**return** a + b;

}

// Overload sum for parameter type float

**public** **float** sum(**float** a, **float** b) {

**return** a + b;

}

// Overload sum for 3 int parameters.

**public** **int** sum(**int** a, **int** b, **int** c) {

**return** a + b + c;

}

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

Calculate calculate = **new** Calculate();

System.***out***.println("sum of 2 int: " + calculate.sum(1, 1));

System.***out***.println("sum of 2 float: " + calculate.sum(1f, 1f));

System.***out***.println("sum of 3 int: " + calculate.sum(1, 1, 1));

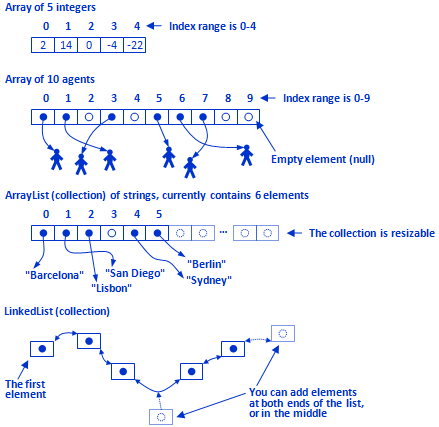
}

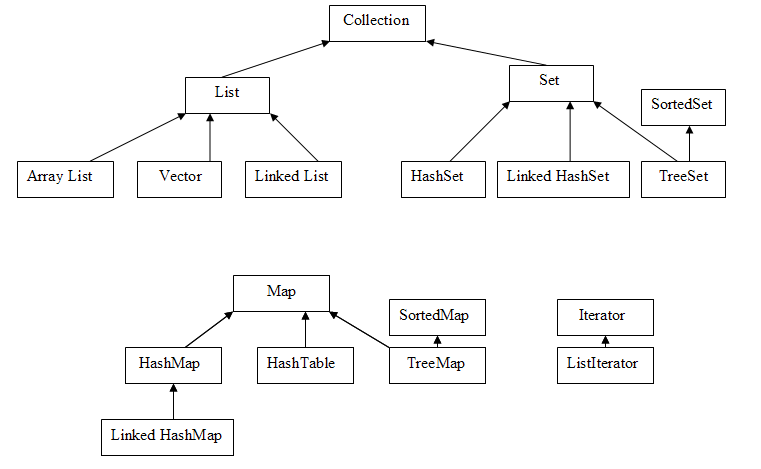
}

Similarly you can overload constructor.

## Arrays and Collections

* Arrays are fixed size and of same type of objects/primitive types.
* Collections are resizable and contain only object type data.
* Collection cannot store primitive types





# Further reading references:

* <https://docs.oracle.com/javase/tutorial/java/>
* <http://www.java-examples.com/>
* *Java The Complete Reference, 9th Edition*.