# Java Programming for Complete Beginners

1. **Install Java on MacOS:**

* <https://www.oracle.com/in/java/technologies/javase-jdk16-downloads.html>
* Check the java and jshell versions.

Text, letter

Description automatically generated

1. **jShell and basics of Java:**

It first came in Java 9. The jshell tool allows you to execute Java code, getting immediate results. It makes Java easy to learn.

Text, letter, email

Description automatically generated

* Printing to console:

Graphical user interface, text

Description automatically generated with medium confidence

* Multiple lines of code: If complete statement is not typed and user presses enter, it brings next line. This allows to execute multiple lines of code.

Graphical user interface, text

Description automatically generated

* Printing dynamic values:

A picture containing chart

Description automatically generated

* Primitive Data types:

1. Integer numbers:

byte (8 bits)

short (16 bits)

int (32 bits)

long (64 bits)

1. Decimal numbers:

float (32 bits)

double (64 bits)

Ex:

double = 4.8; //correct as floating point literal by default is double.

float f2 = 4.8; //incorrect because of above reason.

float f1 = 4.8f; //correct. Explicitly mentioned as float ‘f’

1. Characters:

char (16 bits)

1. Boolean:

boolean (1 bit)

1. **The Java Platform:**

A Compiler is a program which understands the syntax of your programming language and converts it into binary code. However, different Operating systems have different instruction sets(binary code).

Then how Java is platform independent?

Java provides an interesting solution:

* All Java compilers translate source code to an intermediate representation (**bytecode**), which is stored as a .class file on the computer.
* To run Java programs (bytecode), you need a **JVM (Java Virtual Machine).**
* JVM understands bytecode and runs it.
* There are different JVM’s for different operating systems.

1. **Working out of jshell:**

We can create our own .java classes and separately compile and execute them.

1. **Create and Edit your file.**





Graphical user interface, text, application, email

Description automatically generated

1. **Compile your file.**



After successful compilation a .class file is created which contains bytecode.



1. **Execute**

Text

Description automatically generated

It’s giving an error. So some more things are required before we can execute or run the code defined within any class. We need to add a main method.

Graphical user interface, text, application

Description automatically generated

Now we need to compile again as we have changed the .java file. Then we execute it.



This time it’s giving blank. That’s because we did nothing in main. Let’s fix it.



Graphical user interface, text

Description automatically generated

Now we finally execute it and see the result.



1. **JDK vs JRE vs JVM:**

* **JVM** (**J**ava **V**irtual **M**achine) runs Java bytecode.
* **JRE** (**J**ava **R**un time **E**nvironment) = JVM + Libraries + Other components. JRE is required to run your .class files. Example if you give just the .class file to someone, then that person will just require the JRE to run the program.
* **JDK** (**J**ava **D**evelopment **K**it) = JRE + Compilers + Debuggers. JDK is required to compile and develop a program. Example if you give .java file to someone, then that person will require JDK to compile and execute the program.

1. **Setting up Eclipse IDE:**

Eclipse stores all your projects under a workspace. So a workspace will consist of multiple projects. Each project will contain multiple files.

Hierarchy:

Workspace

* Project1

>file1

>file2

* Project2

>file1

>file2

>file3

Graphical user interface, application

Description automatically generated

Steps to create a new Java Project:

1. File > New > Project.

Graphical user interface, application

Description automatically generated

1. Select Java Project from the wizard

Graphical user interface, application

Description automatically generated

1. Fill the Project name of your choice and click Finish.

Graphical user interface, text, application, email

Description automatically generated

1. Project perspective or view opens up with your newly created project.

Graphical user interface, application, Word

Description automatically generated

1. Add a new class to src folder.

Right click on src folder > New > class.

Graphical user interface, application

Description automatically generated

1. Fill the package name and class name. Also check mark next to public static void main stub.

Graphical user interface, application

Description automatically generated

1. Newly created class opens up. Also check the folder structure.

Graphical user interface, text, application

Description automatically generated

1. Edit your class file. Then Right click > Run As > Java Application

Graphical user interface, application, Word

Description automatically generated

1. This executes your code. And Displays the output in console window.

Graphical user interface, application

Description automatically generated

1. Alternatively you can press CMD + f11 to execute your code.
2. **Introduction to Object Oriented Programming:**

//Template or blue print

class Planet {

name, location, distanceFromSun //data/state/fields

revolve(), rotate() //actions/methods/behaviour

}

Planet earth = new Planet(); //Object/Instance

Planet venus = new Planet();

**Encapsulation**: Member variables of a class should not be directly accessible outside its class. They must be accessible only through methods (getters and setters).

We can manually create getters and setters for member variables. But Eclipse provides a shortcut to generate them. Right click > source > Generate Getters and Setters. This creates required methods for us.

Graphical user interface, application

Description automatically generated

**Abstraction:** It’s the way of providing(or exposing) only the necessary details to consumer to consume the functionality, but abstracting away the business logic.

**Constructors:** A special member function which is used to assign initial values to the member variables. It has no return type and has name as class name.

If we have not provided our own constructor, then Java by default provides a default constructor. But if we provide our own constructor, then compiler will not add a default constructor. So we cannot create objects with empty parameters. In this case we need to explicitly provide a default constructor.

Constructors cannot be directly called. A constructor is always invoked when a class object is created, using the **new** keyword

1. **Primitive Data types:**
2. **Integer types:**

byte (8 bits)

short (16 bits)

int (32 bits)

long (64 bits)

* **Wrapper class**

Java has wrapper class corresponding to each of the integer types.

Byte: for byte

Short: for short

Integer: for int

Long: for long

These classes have some useful information like SIZE, BYTES, MIN\_VALUE, MAX\_VALUE etc..

Byte.SIZE // 8

Byte.BYTES // 1

Byte.MAX\_VALUE // 127

Byte.MIN\_VALUE // -128

Short.BYTES // 2

Integer.BYTES // 4

Long.BYTES // 8

* **Type conversions (or casting):**

int i = 1000000;

long l = 50000000000l;

i = l //Error. Cannot store large data into smaller

i = (int) l; //Correct. Using **explicit cast**

l = i; //Correct. Can store small data into large. Compiler does **implicit cast**.

But the compiler is not responsible for the type-safety of the statements. The onus is on the programmer to be aware of type casts, its risks and behaviors.

* **Storing octal and hexadecimal:**

int eight = 010; //Octal

int sixteen = 0x10; //Hexadecimal

* **Built-in operators (+,-,\*,/,%,++,--):**

//Post increment

int i = 10;

int j = i++; // j = 10 and i = 11

//Pre increment

int i = 10;

int j = ++i; // j = 11 and i = 11

1. **Floating-Point types:**

float (32 bits)

double (64 bits)

Default floating point type in Java is double. A float literal must be accompanied with a trailing **f** or **F.**

float f = 34.5; //Error

float f = 34.5f; //Correct

* **BigDecimal**

**double** and **float** are not very precise representations of floating point numbers. They are not used in computations that require high degrees of accuracy.



Actual result should be:

Calendar

Description automatically generated

The **BigDecimal** class was introduced in Java to solve these problems.

Accuracy of BigDecimal representation is retained only when string literals are used to build it.

A screenshot of a computer

Description automatically generated with low confidence

A BigDecimal type can be used to create only **immutable** objects.

All BigDecimal operations supports only BigDecimal operands.

.add

.multiply

.divide

1. **Boolean types:**

Boolean data type is one that holds only one of two values: true or false. Both labels are case-sensitive.

1. **Character types:**
2. **Reference Types:**
3. **Primitive variable v/s Reference variable:**

class Animal {

int id;

Animal(int id) {

this.id = id;

}

}

Animal dog = new Animal(10); //Reference variable

Animal cat = new Animal(20); //Reference variable

int i = 5; //Primitive variable

Here Animal is the **Reference Type** and dog and cat are the **Reference Variable**.

There are two types of memory:

**Stack**: Each method has a separate stack

**Heap**: Globally shared.

Whenever a new object is created, it is stored on **Heap**.

Whenever a primitive variable is created, it is stored on **Stack**.

|  |  |  |
| --- | --- | --- |
| **Stack** | | |
| **Location** | **Value** | **Variable name** |
| A | 5 | i |
| B |  |  |
| C | 1A | dog |
| D | 1C | cat |
| E |  |  |

|  |  |
| --- | --- |
| **Heap** | |
| **Location** | **Object** |
| 1A | new Animal(10) //Animal object |
| 1B |  |
| 1C | new Animal(20) //Animal object |
|  |  |

Here dog and cat stores the reference to the memory locations stored on Heap.

1. **String class:**

String s = “This is Test”;

s.charAt(0); //T

s.substring(1,3); //hi

s.length(); //12

s.indexOf(“is”); //2

s.lastIndexOf(“s”); //10

s.endsWith(“Test”); //true

s.endsWith(“est”); //true

s.endsWith(“Best”); //false

s.startsWith(“This”); //true

s.isEmpty(); //false

s.equals(“This is Test”); //true

s.equalsIgnoreCase(“this is Test”); //true

s.concat(“String”); //”This is Test String” //Creates a new string, **String is immutable**

s; //”This is Test”

s.toUpperCase(); //THIS IS TEST

s.toLowerCase(); //this is test

1. **StringBuffer and StringBuilder**

“123”+”123”+”1234”+”123456”

“1231231234123456”

Just for this basic concatenation operation, it requires creating multiple objects.

One for each of the strings.....4 objects

Additional objects for each concatenation....3 objects more.

Total 7 objects are required. This is costly operation. That’s why **StringBuffer** and **StringBuilder** classes came. They are **mutable** unlike normal strings.

StringBuffer sb = new StringBuffer(“Test”);

StringBuilder sb = new StringBuilder(“Test”);

1. **Wrapper Classes**

A Wrapper class wraps (encloses) around a data type and gives it an object appearance. Wrapper classes are final and immutable.

|  |  |
| --- | --- |
| **Wrapper** | **Primitive** |
| Boolean | boolean |
| Byte | byte |
| Character | char |
| Double | double |
| Float | float |
| Integer | int |
| Long | long |
| Short | short |

Wrapper classes provides lot of utility methods and constants which can be used in various situations.

Integer i1 = new Integer(5); //**First way**

Integer i2 = new Integer(5);

Integer i3 = Integer.valueOf(5); //**Second way**. It does not creates a new Object, but it reuses old values if it exists.

Integer i4 = Integer.valueOf(5);

i1 == i2; //false. Since these reference variables are holding references of two different memory locations.

i3 == i4; //true. Since these reference variables are holding references to the same memory location.

Integer i5 = 5;//**Third way**; **AutoBoxing**; new Java concept

Integer i6 = 5;

i5 == i6; //true. Since AutoBoxing works like the Second way

1. **Working with Date and Time:**

**import java.time.\***

****

****

****

1. **Introduction to Array and ArrayList:**
2. **Arrays:**

int[] marks1 = {10,20,30}; //[10,20,30] **First way**

int[] marks2 = new int[5]; //[0,0,0,0,0] **Second way**

marks1.length; //3 Here length is not the method, it’s a property

System.out.println(marks1); //some memory location

System.out.println(Arrays.toString(marks1)); //[10,20,30]

int[] marks3 = new int[5]; [0,0,0,0,0]

Arrays.fill(marks3, 100); [100,100,100,100,100]

Arrays.equal(marks1, marks2); //false

Arrays.sort(marks1); //[10,20,30]

1. **Functions with variable arguments:**

void print(int... values) {

System.out.println(Arrays.toString(values));

}

print(1); //[1]

print(1,2); //[1,2]

print(1,2,3,4,5); //[1,2,3,4,5]

Note: variable argument should be the last in the argument list in the function.

1. **ArrayList:**

ArrayList<String> arrayList = new ArrayList<String>();

arrayList.add(“Apple”);

arrayList.add(“Bat”);

arrayList.add(“Cat”);

arrayList; //[“Apple”, “Bat”, “Cat”]

arrayList.remove(“Cat”);

arrayList; //[“Apple”, “Bat”]

arrayList.remove(1);

arrayList; //[“Apple”]

1. **Object Oriented Programming:**

**Inheritance:**

Java doesn’t support multiple inheritance. It can have multi-level inheritance.