Java

What is Java?

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

Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995. *James Gosling* is known as the father of Java. Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

## Types of Java Applications

There are mainly 4 types of applications that can be created using Java programming:

#### **1) Standalone Application**

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are 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 a dynamic page is called a web application. Currently, [Servlet](https://www.javatpoint.com/servlet-tutorial), [JSP](https://www.javatpoint.com/jsp-tutorial), [Struts](https://www.javatpoint.com/struts-2-tutorial), [Spring](https://www.javatpoint.com/spring-tutorial), [Hibernate](https://www.javatpoint.com/hibernate-tutorial), [JSF](https://www.javatpoint.com/jsf-tutorial), 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. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering. In Java, [EJB](https://www.javatpoint.com/ejb-tutorial) is used for creating enterprise applications.

#### **4) Mobile Application**

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

Features of Java:

### **Simple**

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language because:

* Java syntax is based on C++ (so easier for programmers to learn it after C++).
* Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
* There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

### **Object-oriented**

Java is an [object-oriented](https://www.javatpoint.com/java-oops-concepts) programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.

Play Videox[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

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)

### **Platform Independent**



Java is platform independent because it is different from other languages like [C](https://www.javatpoint.com/c-programming-language-tutorial), [C++](https://www.javatpoint.com/cpp-tutorial), etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. 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 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).

### **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**



* **Classloader:** Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
* **Bytecode Verifier:** It checks the code fragments for illegal code that can violate access rights to objects.
* **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.

### **Robust**

The English mining of Robust is strong. Java is robust because:

* It uses strong memory management.
* There is a lack of pointers that avoids security problems.
* Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
* There are exception handling and the type checking mechanism in Java. All these points make Java robust.

### **Architecture-neutral**

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

### **Portable**

Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

### **High-performance**

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

### **Distributed**

Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

### **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 doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

### **Dynamic**

Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

### **Difference between JDK and JRE**

**JRE** : The Java Runtime Environment (JRE) provides the libraries, the Java Virtual Machine, and other components to run applets and applications written in the Java programming language. JRE does not contain tools and utilities such as compilers or debuggers for developing applets and applications.



**JDK** : The JDK also called Java Development Kit is a superset of the JRE, and contains everything that is in the JRE, plus tools such as the compilers and debuggers necessary for developing applets and applications.



JVM:

Java virtual Machine(JVM) is a virtual Machine that provides runtime environment to execute java byte code. The JVM doesn't understand Java typo, that's why you compile your \*.java files to obtain \*.class files that contain the bytecodes understandable by the JVM.

JVM control execution of every Java program.

## JVM Architecture



**Class Loader :** Class loader loads the Class for execution.

**Method area :** Stores pre-class structure as constant pool.

**Heap :** Heap is a memory area in which objects are allocated.

**Stack :** Local variables and partial results are store here. Each thread has a private JVM stack created when the thread is created.

**Program register :** Program register holds the address of JVM instruction currently being executed.

**Native method stack :** It contains all native used in application.

**Executive Engine :** Execution engine controls the execute of instructions contained in the methods of the classes.

**Native Method Interface :** Native method interface gives an interface between java code and native code during execution.

**Native Method Libraries :** Native Libraries consist of files required for the execution of native code.

Example:

**public** **class** Hello {

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

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

}

}

## Steps to Compile and Run your first Java program

**Step 1:** Open a text editor and write the code as above.

**Step 2:** Save the file as Hello.java

**Step 3:** Open command prompt and go to the directory where you saved your first java program assuming it is saved in C drive.

**Step 4:** Type javac Hello.java and press Return**(Enter KEY)** to compile your code. This command will call the Java Compiler asking it to compile the specified file. If there are no errors in the code the command prompt will take you to the next line.

**Step 5:** Now type java Hello on command prompt to run your program.

**Step 6:** You will be able to see **Hello world program** printed on your command prompt.

## Hello World Program using Eclipse

Eclipse is an IDE (Integrated Development Environment) which is used to develop applications. It is design and developed by Eclipse foundation, if you don’t have eclipse download, then download it from its official site by following this download link [Download Eclipse from here](https://www.eclipse.org/downloads/) Here we will see how to create and run **hello world** program using eclipse IDE. It require following steps that consists of **creating project, class file, writing code, running code etc**.

#### **Run Eclipse and Create Project**

Open eclipse startup and then create new project. To create project click on **File** menu and select **Java project** option. It will open a window that ask for project name. Provide the project name and click on the finish button. See the below screenshot.



After creating project, we can see our new created project in the left side bar that looks like below.



### **Create Java Class**

Now create Java class file by **right click** on the **project** and **select class** file option. It will open a window to ask for class name, provide the class name and click on finish button.



### **Write Hello World**

The above created class file includes some line of codes including main method as well. Now we need to write just print statement to print Hello World message.



### **Run The Program**

Now run the program by selecting **Run** menu from the menu bar or use **Ctrl+F11** button combination. After running, it will print Hello World to the console which is just bottom to the program window.



This is a simple program that we run here while using IDE we can create and build large scale of applications. If you are a beginner and not familiar to the Eclipse then don’t worry it is very easy to operate just follow the above steps to create the program.

### **Creating Hello World Example**

Let's create the hello java program:

1. **class** Simple{
2. **public** **static** **void** main(String args[]){
3. System.out.println("Hello Java");
4. }
5. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=Simple)

Save the above file as Simple.java.

|  |  |
| --- | --- |
| **To compile:** | javac Simple.java |
| **To execute:** | java Simple |

**Output:**

Hello Java

## Parameters used in First Java Program

Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().

* **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. We will discuss the internal working of [System.out.println()](https://www.javatpoint.com/system-out-println-in-java) statement in the coming section.

## 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:



**Classloader:** It is the subsystem of JVM that is used to load class files.

Play Videox

**Bytecode Verifier:** Checks the code fragments for illegal code that can violate access rights to objects.

**Interpreter:** Read bytecode stream then execute the instructions.

Example1:

**package** com.Employee;

**public** **class** Student {

**int** rollno=20;

String address="hyd";

**int** age=28;

**void** display() {

System.***out***.println(rollno+" "+address+" "+age);

}

**static** **void** display1() {

System.***out***.println("this is static method");

}

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

Student st=**new** Student();

st.display();

Student.*display1*();

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

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

}

}

Access Modifiers:

There are four types of Java access modifiers:

* 1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
* Example:

|  |
| --- |
| * // Java program to illustrate error while * // using class from different package with * // private modifier * **package** p1; * **class** A * { * **private** **void** display() * { * System.out.println("GeeksforGeeks"); * } * } * **class** B * { * **public** **static** **void** main(String args[]) * { * A obj = **new** A(); * // Trying to access private method * // of another class * obj.display(); * } * } |

**Output:**

error: display() has private access in A

obj.display();

**2.Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Example:

|  |
| --- |
| // Java program to illustrate  // public modifier  **package** p1;  **public** **class** A  {  **public** **void** display()      {          System.out.println("GeeksforGeeks");      }  } |

* Java

|  |
| --- |
| **package** p2;  **import** p1.\*;  **class** B {  **public** **static** **void** main(String args[])      {          A obj = **new** A();          obj.display();      }  } |

**Output:**

GeeksforGeeks

**3.Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package.

Example:

|  |
| --- |
| // Java program to illustrate default modifier  **package** p1;    // Class Geeks is having Default access modifier  **class** Geek  {  **void** display()      {          System.out.println("Hello World!");      }  } |

* Java

|  |
| --- |
| // Java program to illustrate error while  // using class from different package with  // default modifier  **package** p2;  **import** p1.\*;    // This class is having default access modifier  **class** GeekNew  {  **public** **static** **void** main(String args[])      {          // Accessing class Geek from package p1          Geeks obj = **new** Geek();            obj.display();      }  } |

**4.Protected**: The access level of a protected modifier is within the package and outside the package through child class

Example:

|  |
| --- |
| // Java program to illustrate  // protected modifier  **package** p1;    // Class A  **public** **class** A  {  **protected** **void** display()      {          System.out.println("GeeksforGeeks");      }  } |

* Java

|  |
| --- |
| // Java program to illustrate  // protected modifier  **package** p2;  **import** p1.\*; // importing all classes in package p1    // Class B is subclass of A  **class** B **extends** A  {  **public** **static** **void** main(String args[])  {      B obj = **new** B();      obj.display();  }    } |

**Output:**

GeeksforGeeks

DataTypes:

There are two types of data types in Java:

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).

## 8 Primitive Data Types

### 1. boolean type

* The boolean data type has two possible values, either true or false.
* Default value: false.
* They are usually used for **true/false** conditions.

### Example 1: Java boolean data type

class Main {

public static void main(String[] args) {

boolean flag = true;

System.out.println(flag); // prints true

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

### 2. byte type

* The byte data type can have values from **-128** to **127** (8-bit signed two's complement integer).
* If it's certain that the value of a variable will be within -128 to 127, then it is used instead of int to save memory.
* Default value: 0

### Example 2: Java byte data type

class Main {

public static void main(String[] args) {

byte range;

range = 124;

System.out.println(range); // prints 124

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

### 3. short type

* The short data type in Java can have values from **-32768** to **32767** (16-bit signed two's complement integer).
* If it's certain that the value of a variable will be within -32768 and 32767, then it is used instead of other integer data types (int, long).
* Default value: 0

### Example 3: Java short data type

class Main {

public static void main(String[] args) {

short temperature;

temperature = -200;

System.out.println(temperature); // prints -200

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

### 4. int type

* The int data type can have values from **-231** to **231-1** (32-bit signed two's complement integer).
* If you are using Java 8 or later, you can use an unsigned 32-bit integer. This will have a minimum value of 0 and a maximum value of 232-1. To learn more, visit [How to use the unsigned integer in java 8?](http://stackoverflow.com/questions/25556017/how-to-use-the-unsigned-integer-in-java-8)
* Default value: 0

### Example 4: Java int data type

class Main {

public static void main(String[] args) {

int range = -4250000;

System.out.println(range); // print -4250000

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

### 5. long type

* The long data type can have values from **-263** to **263-1** (64-bit signed two's complement integer).
* If you are using Java 8 or later, you can use an unsigned 64-bit integer with a minimum value of **0** and a maximum value of **264-1**.
* Default value: 0

### Example 5: Java long data type

class LongExample {

public static void main(String[] args) {

long range = -42332200000L;

System.out.println(range); // prints -42332200000

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

Notice, the use of L at the end of -42332200000. This represents that it's an integer of the long type.

### 6. double type

* The double data type is a double-precision 64-bit floating-point.
* It should never be used for precise values such as currency.
* Default value: 0.0 (0.0d)

### Example 6: Java double data type

class Main {

public static void main(String[] args) {

double number = -42.3;

System.out.println(number); // prints -42.3

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

### 7. float type

* The float data type is a single-precision 32-bit floating-point. Learn more about [single-precision and double-precision floating-point](http://stackoverflow.com/questions/801117/whats-the-difference-between-a-single-precision-and-double-precision-floating-p) if you are interested.
* It should never be used for precise values such as currency.
* Default value: 0.0 (0.0f)

### Example 7: Java float data type

class Main {

public static void main(String[] args) {

float number = -42.3f;

System.out.println(number); // prints -42.3

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

Notice that we have used -42.3f instead of -42.3in the above program. It's because -42.3 is a double literal.

To tell the compiler to treat -42.3 as float rather than double, you need to use f or F.

If you want to know about single-precision and double-precision,

### 8. char type

* It's a 16-bit Unicode character.
* The minimum value of the char data type is '\u0000' (0) and the maximum value of the is '\uffff'.
* Default value: '\u0000'

### Example 8: Java char data type

class Main {

public static void main(String[] args) {

char letter = '\u0051';

System.out.println(letter); // prints Q

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

Here, the Unicode value of Q is **\u0051**. Hence, we get Q as the output.

Here is another example:

class Main {

public static void main(String[] args) {

char letter1 = '9';

System.out.println(letter1); // prints 9

char letter2 = 65;

System.out.println(letter2); // prints A

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

Here, we have assigned 9 as a character (specified by single quotes) to the letter1 variable. However, the letter2 variable is assigned 65 as an integer number (no single quotes).

Hence, A is printed to the output. It is because Java treats characters as an integer and the ASCII value of A is 65.

### String type

Java also provides support for character strings via java.lang.String class. Strings in Java are not primitive types. Instead, they are objects. For example,

String myString = "Java Programming";

Here, myString is an object of the String class.

Methods:

Example:

**package** com.Employee;

**public** **class** Student {

**int** x=10;

**int** y=20;

**public** **void** display()

{

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

}

**public** **int** add() {

**int** z=x+y;

**return** z;

}

**public** **int** sum(**int** z,**int** c)

{

**int** s=z+c;

**return** s;

}

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

Student st=**new** Student();

st.display();

**int** m= st.add();

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

**int** d=st.sum(50,70);

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

}

}

In Java, there are two types of methods:

* **User-defined Methods**: We can create our own method based on our requirements.
* **Standard Library Methods**: These are built-in methods in Java that are available to use.

Let's first learn about user-defined methods.

## Declaring a Java Method

The syntax to declare a method is:

returnType methodName() {

// method body

}

Here,

* **returnType** - It specifies what type of value a method returns For example if a method has an int return type then it returns an integer value.  
    
  If the method does not return a value, its return type is void.
* **methodName** - It is an [identifier](https://www.programiz.com/java-programming/keywords-identifiers#identifiers) that is used to refer to the particular method in a program.
* **method body** - It includes the programming statements that are used to perform some tasks. The method body is enclosed inside the curly braces { }.

For example,

int addNumbers() {

// code

}

In the above example, the name of the method is adddNumbers(). And, the return type is int.

This is the simple syntax of declaring a method. However, the complete syntax of declaring a method is

modifier static returnType nameOfMethod (parameter1, parameter2, ...) {

// method body

}

Here,

* **modifier** - It defines access types whether the method is public, private, and so on. To learn more, visit [Java Access Specifier](https://www.programiz.com/java-programming/access-modifiers).
* **static** - If we use the static keyword, it can be accessed without creating objects.  
    
  For example, the sqrt() method of standard [Math class](https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html) is static. Hence, we can directly call Math.sqrt() without creating an instance of Math class.
* **parameter1/parameter2** - These are values passed to a method. We can pass any number of arguments to a method.

## Calling a Method in Java

In the above example, we have declared a method named addNumbers(). Now, to use the method, we need to call it.

Here's is how we can call the addNumbers() method.

// calls the method

addNumbers();

Working of Java Method Call

## Example 1: Java Methods

class Main {

// create a method

public int addNumbers(int a, int b) {

int sum = a + b;

// return value

return sum;

}

public static void main(String[] args) {

int num1 = 25;

int num2 = 15;

// create an object of Main

Main obj = new Main();

// calling method

int result = obj.addNumbers(num1, num2);

System.out.println("Sum is: " + result);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

Sum is: 40

In the above example, we have created a method named addNumbers(). The method takes two parameters a and b. Notice the line,

int result = obj.addNumbers(num1, num2);

Here, we have called the method by passing two arguments num1 and num2. Since the method is returning some value, we have stored the value in the result variable.

**Note**: The method is not static. Hence, we are calling the method using the object of the class.

## Java Method Return Type

A Java method may or may not return a value to the function call. We use the **return statement** to return any value. For example,

int addNumbers() {

...

return sum;

}

Here, we are returning the variable sum. Since the return type of the function is int. The sum variable should be of int type. Otherwise, it will generate an error.

### Example 2: Method Return Type

class Main {

// create a method

public static int square(int num) {

// return statement

return num \* num;

}

public static void main(String[] args) {

int result;

// call the method

// store returned value to result

result = square(10);

System.out.println("Squared value of 10 is: " + result);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

Squared value of 10 is: 100

In the above program, we have created a method named square(). The method takes a number as its parameter and returns the square of the number.

Here, we have mentioned the return type of the method as int. Hence, the method should always return an integer value.

Representation of the Java method returning a value

**Note**: If the method does not return any value, we use the void keyword as the return type of the method. For example,

public void square(int a) {

int square = a \* a;

System.out.println("Square is: " + square);

}

## Method Parameters in Java

A method parameter is a value accepted by the method. As mentioned earlier, a method can also have any number of parameters. For example,

// method with two parameters

int addNumbers(int a, int b) {

// code

}

// method with no parameter

int addNumbers(){

// code

}

If a method is created with parameters, we need to pass the corresponding values while calling the method. For example,

// calling the method with two parameters

addNumbers(25, 15);

// calling the method with no parameters

addNumbers()

### Example 3: Method Parameters

class Main {

// method with no parameter

public void display1() {

System.out.println("Method without parameter");

}

// method with single parameter

public void display2(int a) {

System.out.println("Method with a single parameter: " + a);

}

public static void main(String[] args) {

// create an object of Main

Main obj = new Main();

// calling method with no parameter

obj.display1();

// calling method with the single parameter

obj.display2(24);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

Method without parameter

Method with a single parameter: 24

Here, the parameter of the method is int. Hence, if we pass any other data type instead of int, the compiler will throw an error. It is because Java is a strongly typed language.

**Note**: The argument 24 passed to the display2() method during the method call is called the actual argument.

The parameter num accepted by the method definition is known as a formal argument. We need to specify the type of formal arguments. And, the type of actual arguments and formal arguments should always match.

## Standard Library Methods

The standard library methods are built-in methods in Java that are readily available for use. These standard libraries come along with the Java Class Library (JCL) in a Java archive (\*.jar) file with JVM and JRE.

For example,

* print() is a method of java.io.PrintSteam. The print("...") method prints the string inside quotation marks.
* sqrt() is a method of Math class. It returns the square root of a number.

Here's a working example:

### Example 4: Java Standard Library Method

public class Main {

public static void main(String[] args) {

// using the sqrt() method

System.out.print("Square root of 4 is: " + Math.sqrt(4));

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

Square root of 4 is: 2.0

To learn more about standard library methods, visit [Java Library Methods](https://www.programiz.com/java-programming/library).

## What are the advantages of using methods?

**1.** The main advantage is **code reusability**. We can write a method once, and use it multiple times. We do not have to rewrite the entire code each time. Think of it as, "write once, reuse multiple times".

### Example 5: Java Method for Code Reusability

public class Main {

// method defined

private static int getSquare(int x){

return x \* x;

}

public static void main(String[] args) {

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

// method call

int result = getSquare(i);

System.out.println("Square of " + i + " is: " + result);

}

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

Square of 1 is: 1

Square of 2 is: 4

Square of 3 is: 9

Square of 4 is: 16

Square of 5 is: 25

In the above program, we have created the method named getSquare() to calculate the square of a number. Here, the method is used to calculate the square of numbers less than **6**.

Hence, the same method is used again and again.

**2.** Methods make code more **readable and easier** to debug. Here, the getSquare() method keeps the code to compute the square in a block. Hence, makes it more readable.