**Java Tutorial**

**Java was developed by Sun Microsystems in 1995, and later acquired by Oracle Corporation.**

**Java is a highly popular, object-oriented programming language. This platform independent programming language is utilized for Android development, web development, artificial intelligence, cloud applications, and much more.**

**Java is intended to let application developers write once, and run anywhere (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java code is compiled to byte code that can run on any Java Virtual Machine.**

**Java is known for its simplicity, robustness, portability, security and high-performance features, making it a popular choice for enterprise-level applications.**

**Implementation of a Java application program involves a following step. They include:**

**1. Creating the program**

**2. Compiling the program**

**3. Running the program**

**Remember that, before we begin creating the program, the Java Development Kit (JDK) must be properly installed on our system and also path will be set.**

**• Creating Program**

**We can create a program using Text Editor (Notepad) or IDE (NetBeans)**

**class Test**

**{**

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

**{**

**System.out.println(“My First Java Program.”);**

**}**

**};**

**File -> Save -> d:\Test.java**

**• Compiling the program**

**To compile the program, we must run the Java compiler (javac), with the name of the source file on “command prompt” like as follows**

**Javac Test.java**

**If everything is OK, the “javac” compiler creates a file called “Test.class” containing byte code of the program.**

**• Running the program**

**We need to use the Java Interpreter to run a program.**

**Java terminology**

**1. Java Development Kit(JDK): It is a complete Java development kit that includes everything including compiler(javac), Java Runtime Environment (JRE), java debuggers, java docs, etc. For the program to execute in java, we need to install JDK on our computer in order to create, compile and run the java program.**

**2. Bytecode in the Development Process: The compilation is done by the JAVAC compiler which is a primary Java compiler included in the Java development kit (JDK). It takes the Java program as input and generates bytecode as output.It is saved as .class file by the compiler. The bytecode generated is a non-executable code and needs an interpreter like JVM to execute on a machine**

**3. Java Virtual Machine(JVM): It execute the bytecode produced by the compiler. Every Operating System has a different JVM but the output they produce after the execution of bytecode is the same across all the operating systems. This is why Java is known as a platform-independent language. It is also known as an**[**interpreter**](https://www.geeksforgeeks.org/compiler-vs-interpreter-2/)**.**

**4. Java Runtime Environment (JRE): JRE installation on our computers allows the java program to run, however, we cannot compile it. JRE includes a browser, JVM, applet support, and plugins. For running the java program, a computer needs JRE.**

**5. Garbage Collector: In Java, programmers can’t delete the objects. To delete or recollect that memory JVM has a program called Garbage Collector. Garbage Collectors can recollect the objects that are not referenced. So Java makes the life of a programmer easy by handling memory management.**

**6. ClassPath: The classpath is the file path where the java runtime and Java compiler look for .class files to load. By default, JDK provides many libraries. If you want to include external libraries they should be added to the classpath.**

**Primary/Main Features of Java**

**1. Platform Independent:**

**2. Object-Oriented Programming Language: Organizing the program in the terms of a collection of objects is a way of object-oriented programming, each of which represents an instance of the class.**

**The four main concepts of Object-Oriented programming are:**

1. **Abstraction**
2. **Encapsulation**
3. **Inheritance**
4. **Polymorphism**

**3. Simple: Java is one of the simple languages as it does not have complex features like pointers, operator overloading, multiple inheritances, and Explicit memory allocation.**

**4. Robust: Java language is robust which means reliable. It is developed in such a way that it puts a lot of effort into checking errors as early as possible, that is why the java compiler is able to detect even those errors that are not easy to detect by another programming language. The main features of java that make it robust are garbage collection, Exception Handling, and memory allocation.**

**5. Secure: In java, we don’t have pointers, so we cannot access out-of-bound arrays i.e it shows ArrayIndexOutOfBound Exception if we try to do so. That’s why several security flaws like stack corruption or buffer overflow are impossible to exploit in Java. Also, java programs run in an environment that is independent of the os(operating system) environment which makes java programs more secure.**

**6. Distributed: The java programs can be easily distributed on one or more systems that are connected to each other through an internet connection.**

**7. Multithreading: Java supports multithreading. It is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of the CPU.**

**8. Portable: As we know, java code written on one machine can be run on another machine. The platform-independent feature of java in which its platform-independent bytecode can be taken to any platform for execution makes java portable.**

**9. High Performance: Java architecture is defined in such a way that it reduces overhead during the runtime and at some times java uses Just In Time (JIT) compiler where the compiler compiles code on-demand basics where it only compiles those methods that are called making applications to execute faster.**

**10. Dynamic flexibility: Java being completely object-oriented gives us the flexibility to add classes, new methods to existing classes, and even create new classes through sub-classes. Java even supports functions written in other languages such as C, C++ which are referred to as native methods.**

**11. Sandbox Execution: Java programs run in a separate space that allows user to execute their applications without affecting the underlying system with help of a bytecode verifier. Bytecode verifier also provides additional security as its role is to check the code for any violation of access.**

**12. Write Once Run Anywhere: java application generates a ‘.class’ file that corresponds to our applications(program) but contains code in binary format. It provides ease to architecture-neutral ease as bytecode is not dependent on any machine architecture.**

**13. Power of compilation and interpretation: Most languages are designed with the purpose of either they are compiled language or they are interpreted language. But java integrates arising enormous power as Java compiler compiles the source code to bytecode and JVM executes this bytecode to machine OS-dependent executable code.**

**Sample.java**

**class Simple{**

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

**System.out.println("Hello Java");**

**}**

**}**

**Some Keywords:**

**class : class keyword is used to declare classes in Java.**

**public : It is an access specifier. Public means this function is visible to all.**

**static : static is again a keyword used to make a function static. To execute a static function you do not have to create an Object of the class. The main() method here is called by JVM, without creating any object for class.**

**void : It is the return type, meaning this function will not return anything.**

**main : main() method is the most important method in a Java program. This is the method which is executed, hence all the logic must be inside the main() method. If a java class is not having a main() method, it causes compilation error.**

**String[] args : This is used to signify that the user may opt to enter parameters to the Java Program at command line. We can use String[] args, String args[] or String... args. Java compiler would accept all three forms.**

**System.out.println : This is used to print anything on the console like “printf” in C language.**

**Note: Having a semicolon at the end of class is optional in Java.**

**Q) Can you save a Java source file by another name than the class name?**

**Yes, if the class is not public.**

**Java Class File**

* **A Java class file is a file containing Java bytecode and having .class extension that can be executed by**[**JVM**](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/)**.**
* **A Java class file is created by a Java compiler from .java files as a result of successful compilation. If a .java file has more than one class then each class will compile into a separate class files.**

***Elements of class file are as follows:***

1. **magic\_number:** The first 4 bytes of class file are termed as magic\_number. This is a predefined value which the JVM use to identify whether the .class file is generated by valid compiler or not. The predefined value will be in hexadecimal form i.e. 0xCAFEBABE.
2. **minor\_version & major\_version:** These both together represents .class file version. JVM will use these versions to identify which version of the compiler generates the current .class file. We denotes the version of class file as M.m where M stands for major\_version and m stands for minor\_version

Note: Lower version compiler generated .class file can be executed by high version JVM but higher version compiler generated .class file cannot be executed by lower version JVM. If we will try to execute we will get run time exception.

1. **constant\_pool\_count:** It represents the number of the constants present in the constant pool (When a Java file is compiled, all references to variables and methods are stored in the class's constant pool as a symbolic reference).
2. **constant\_pool[]:** It represents the information about constants present in constant pool file.
3. **access\_flags:** It provide the information about the modifiers which are declared to the class file.
4. **this\_class:** It represents fully qualified name of the class file.
5. **super\_class:** It represents fully qualified name of the immediate super class of current class. Consider above Sample.java file. When we will compile it, then we can say this\_class will be Sample class and super\_class will be Object class.
6. **interface\_count:** It returns the number of interfaces implemented by current class file.
7. **interface[]:** It returns interfaces information implemented by current class file.
8. fields\_count: It represents the number of fields (static variable) present in current class file.
9. **fields[]:** It represent fields (static variable) information present in current class file.
10. **method\_count**: It represents number of methods present in current class file.
11. **method[]:** It returns information about all methods present in current class file.
12. **attributes\_count:** It returns the number of attributes (instance variables) present in current class file.
13. **attributes[]:** It provides information about all attributes present in current class file.

**IMPORTANT POINTS OF JAVA:**

**1. Java is a high-level, object-oriented programming language that was first released in 1995.**

**2. Java is platform-independent, which means that code written in Java can run on any platform that has a Java Virtual Machine (JVM) installed.**

**3. Java code is compiled into bytecode, which can then be executed by the JVM.**

**4. Java is known for its “write once, run anywhere” philosophy, which makes it a popular choice for cross-platform development.**

**5. Java provides automatic memory management through garbage collection, which makes it easier to write and maintain code.**

**6. Java has a vast standard library that provides a wide range of tools for common programming tasks.**

**7. Java is widely used in enterprise applications, web development, and Android app development.**

**8. Java is a strongly typed language, which means that every variable and expression has a specific type that must be declared before use.**

**9. Java has a robust exception-handling mechanism that makes it easier to handle errors and unexpected behavior in code.**

**10. Java supports multithreading, which makes it possible to write programs that can perform multiple tasks simultaneously.**

**11. JVM, JRE, and JDK three are all platform-dependent because the configuration of each Operating System is different. But, Java is platform-independent.**

**Java Platforms / Editions**

**1) Java SE (Java Standard Edition): It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs,**[**String**](https://www.javatpoint.com/java-string)**, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking****, AWT, Swing, Reflection, Collection, etc.**

**2) Java EE (Java Enterprise Edition): It is an enterprise platform that is mainly used to develop web and enterprise applications. It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB,**[**JPA**](https://www.javatpoint.com/jpa-tutorial)**, etc.**

**3) Java ME (Java Micro Edition): It is a micro platform that is dedicated to mobile applications.**

**4) JavaFX: It is used to develop rich internet applications. It uses a lightweight user interface API.**

**C++ Vs Java:**

| **TOPIC** | **C++** | **Java** |
| --- | --- | --- |
| Memory Management | Use of pointers, structures, union | No use of pointers. Supports references, thread and interfaces. |
| Libraries | Comparatively available with low-level functionalities | Wide range of classes for various high-level services |
| Multiple Inheritance | Provide both single and multiple inheritance. | Multiple inheritances is partially done through interfaces |
| Operator Overloading | Supports operator overloading | It doesn’t support this feature |
| Program Handling | Functions and variables can reside outside classes. | Functions and variables reside only in classes, packages are used. |
| Portability | Platform dependent, must be recompiled for different platform | Platform independent, byte code generated works on every OS. |
| Thread Support | No built-in support for threads, depends on libraries. | It has built-in thread support. |
| Performance | faster | Slow, due to its use of a virtual machine and automatic memory management. |
| Call by Value and Call by reference | C++ supports both call by value and call by reference. | Java supports call by value only. |

**Python Vs Java:**

| **TOPIC** | **Python** | **Java** |
| --- | --- | --- |
| Compilation process | Python is an interpreted programming language | Java is both compiled and interpreted language, which is first compiled and then interpreted into a byte code. |
| Syntax Complexity | No need of semi colons and curly braces, uses indentation | Define particular block by curly braces, end statements by ; |
| Ease of typing | Dynamic, no need to define the exact datatype of variables. | Strongly typed, need to define the exact datatype of variables |
| Speed of execution | Expected to run slower than Java programs | Java is much faster than python in terms of speed. |
| Multiple Inheritance | Provide both single and multiple inheritance | Multiple inheritance is partially done through interfaces |

##### Java vs C#

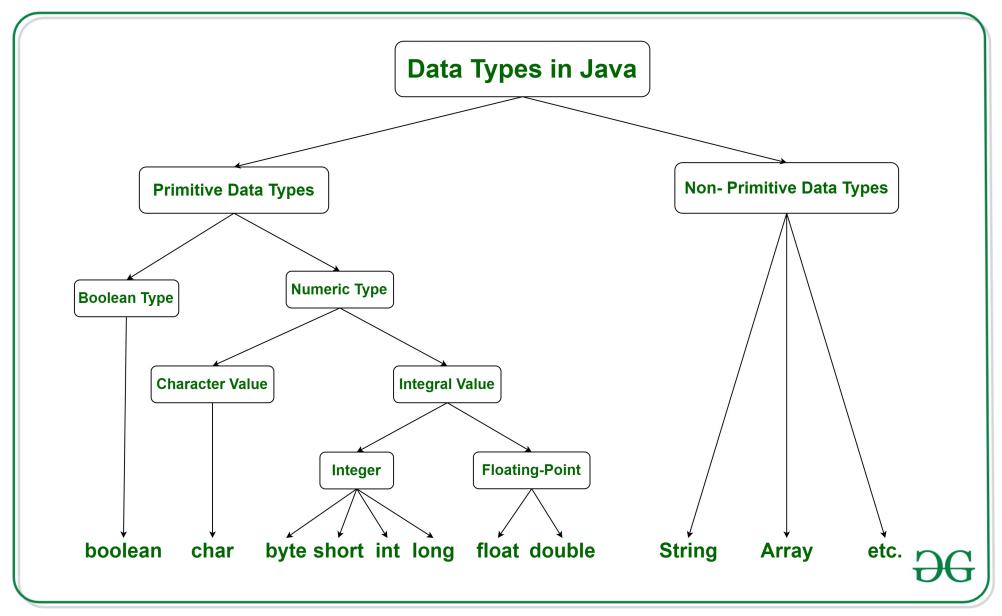
| **Feature** | **C#** | **Java** |
| --- | --- | --- |
| Operator Overloading | C# supports Operator Overloading. | Java does not support operator overloading. |
| Runtime Environment | C# supports [CLR](https://www.geeksforgeeks.org/common-language-runtime-clr-in-c/)(Common Language Runtime). | Java supports [JVM](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/)(Java Virtual Machine). |
| Pointers, Structure, Union | C# supports pointer, structures and unions. | Java does not support. |
| Checked Exceptions | C# does not support Unix-based checked exceptions. | Java supports both checked and unchecked exceptions. |
| Public Classes | In C#, there can be many public classes inside a code. | In Java there can be only one public class inside a source code |

**Data types in Java**

**Java is statically typed and a strongly typed language. It means, all** [**variables**](https://www.javatpoint.com/java-variables) **must be declared before its use.**

**Java has mostly two type of data types:**

1. **Primitive Data Type: such as boolean, char, int, short, byte, long, float, & double**
2. **Non-Primitive Data Type or Object Data type: such as String, Array, etc.**



1. **Types Of Primitive Data Types**

**Primitive data are only single values and have no special capabilities. There are 8 primitive data types. They are depicted below in tabular format below as follows:**



**Non-Primitive Data Type or Reference Data Types**

The Reference Data Types will contain a memory address of variable values because the reference types won’t store the variable value directly in memory. They are strings, objects, arrays, etc.

1. [**Strings**](https://www.geeksforgeeks.org/strings-in-java/)

Strings are defined as an array of characters. The difference between a character array and a string in Java is, that the string is designed to hold a sequence of characters in a single variable whereas, a character array is a collection of separate char type entities. Unlike C/C++, Java strings are not terminated with a null character.

Syntax:

// Declare String without using new operator

String var= "GeeksforGeeks";

// Declare String using new operator

String var1 = new String("GeeksforGeeks");

1. [**Class**](https://www.geeksforgeeks.org/classes-objects-java/)

A class is a user-defined blueprint or prototype from which objects are created.  It represents the set of properties or methods that are common to all objects of one type.

1. [**Object**](https://www.geeksforgeeks.org/classes-objects-java/)

It is a basic unit of Object-Oriented Programming and represents real-life entities.

1. [**Interface**](https://www.geeksforgeeks.org/interfaces-in-java/)

Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).

1. [**Array**](https://www.geeksforgeeks.org/arrays-in-java/)

An array is a group of like-typed variables that are referred to by a common name. Arrays in Java work differently than they do in C/C++.

**Variables in Java**

* Variable in Java is a data container that saves the data values during Java program execution. Every variable is assigned a data type that designates the type and quantity of value it can hold.
* A variable is a name to a memory location for the data.
* In Java, all variables must be declared before use.

Declaration: DataType dataName;

**Types of Variables in Java**

1. Local Variables :

A variable defined within a block or method or constructor is called a local variable.

* These variables are created when the block is entered, or the function is called and destroyed after exiting from the block or when the call returns from the function.
* The scope of these variables exists only within the block in which the variables are declared, i.e., we can access these variables only within that block.
* Initialization of the local variable is mandatory before using it in the defined scope.

2. Instance Variables:

Instance variables are non-static variables and are declared in a class outside of any method, constructor, or block.

* As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
* Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier, then the default access specifier will be used.
* Initialization of an instance variable is not mandatory. Its default value is 0.
* Instance variables can be accessed only by creating objects.

3. Static Variables:

Static variables are also known as class variables.

* These variables are declared similarly as instance variables. The difference is that static variables are declared using the static keyword within a class outside of any method, constructor or block.
* Unlike instance variables, we can only have one copy of a static variable per class, irrespective of how many objects we create.
* Static variables are created at the start of program execution and destroyed automatically when execution ends.
* Initialization of a static variable is not mandatory. Its default value is 0.
* If we access a static variable like an instance variable (through an object), the compiler will show a warning message, which won't halt the program. The compiler will replace the object name with the class name automatically.
* If we access a static variable without the class name, the compiler will automatically append the class name.
* Unlike C/C++, static local variables are not allowed in Java.

**Operators in Java**

1. **Arithmetic Operators:**

They are used to perform simple arithmetic operations on primitive data types.

* \* : Multiplication
* / : Division
* % : Modulo
* + : Addition
* - : Subtraction

1. **Unary Operators:**

Unary operators need only one operand. They are used to increment, decrement or negate a value.

* - : Unary minus, used for negating the values.
* + : Unary plus indicates the positive value . It performs an automatic conversion to int when the type of its operand is the byte, char, or short. This is called unary numeric promotion.
* ++ : Increment operator, used for incrementing the value by 1. There are two varieties of increment operators.
  + Post-Increment(x++): Value is first used for computing the result and then incremented.
  + Pre-Increment(++x): Value is incremented first, and then the result is computed.
* -- : Decrement operator, used for decrementing the value by 1. There are two varieties of decrement operators.
  + Post-decrement: Value is first used for computing the result and then decremented.
  + Pre-Decrement: Value is decremented first, and then the result is computed.
* ! : Logical not operator, used for inverting a boolean value.

1. **Assignment Operator:**

'=' Assignment operator is used to assigning a value to any variable. It has a right to left associativity, i.e. value given on the right-hand side of the operator is assigned to the variable on the left, and therefore right-hand side value must be declared before using it or should be a constant.

The general format of the assignment operator is:

variable = value;

In many cases, the assignment operator can be combined with other operators to build a shorter version of the statement called a Compound Statement. For example, instead of a = a+5, we can write a += 5.

* +=, for adding left operand with right operand and then assigning it to the variable on the left.
* -=, for subtracting right operand from left operand and then assigning it to the variable on the left.
* \*=, for multiplying left operand with right operand and then assigning it to the variable on the left.
* /=, for dividing left operand by right operand and then assigning it to the variable on the left.
* %=, for assigning modulo of left operand by right operand and then assigning it to the variable on the left.

1. **Relational Operators:**

These operators are used to check for relations like equality, greater than, and less than. They return boolean results after the comparison and are extensively used in looping statements as well as conditional if-else statements. The general format is,

variable relation\_operator value

Some of the relational operators are-

* ==, **Equal to:** returns true if the left-hand side is equal to the right-hand side.
* !=, **Not Equal to:** returns true if the left-hand side is not equal to the right-hand side.
* <, **less than**: returns true if the left-hand side is less than the right-hand side.
* <=, **less than or equal to:** returns true if the left-hand side is less than or equal to the right-hand side.
* >, **Greater than:** returns true if the left-hand side is greater than the right-hand side.
* >=, **Greater than or equal to:** returns true if the left-hand side is greater than or equal to the right-hand side.

1. **Logical Operators:**

These operators are used to perform "logical AND" and "logical OR" operations, i.e., a function similar to AND gate and OR gate in digital electronics. One thing to keep in mind is the second condition is not evaluated if the first one is false, i.e., it has a short-circuiting effect. Used extensively to test for several conditions for making a decision. Java also has "Logical NOT", which returns true when the condition is false and vice-versa

Conditional operators are:

* &&, **Logical AND**: returns true when both conditions are true.
* ||, **Logical OR:** returns true if at least one condition is true.
* !, **Logical NOT:** returns true when a condition is false and vice-versa

1. **Ternary operator:**

Ternary operator is a shorthand version of the if-else statement. It has three operands and hence the name ternary.

The general format is:

condition ? if true : if false

1. **Bitwise Operators:**

These operators are used to perform the manipulation of individual bits of a number. They can be used with any of the integer types. They are used when performing update and query operations of the Binary indexed trees.

* &, Bitwise AND operator: returns bit by bit AND of input values.
* |, Bitwise OR operator: returns bit by bit OR of input values.
* ^, Bitwise XOR operator: returns bit-by-bit XOR of input values.
* ~, Bitwise Complement Operator: This is a unary operator which returns the one's complement representation of the input value, i.e., with all bits inverted.

1. **Shift Operators:** These operators are used to shift the bits of a number left or right, thereby multiplying or dividing the number by two, respectively. They can be used when we have to multiply or divide a number by two.

General format-  number shift\_op number\_of\_places\_to\_shift;

* <<, **Signed Left shift operator:**shifts the bits of the number to the left and fills 0 on voids left as a result. Similar effect as multiplying the number with same power of two. E.g. 10<< 1 = 10\*2 = 20 (as 1010<<1 = 10100 = 20)

Also, 1<<n = 2n

* >>, **Signed Right shift operator:** shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit depends on the sign of the initial number. Similar effect as dividing the number with same power of two.

E.g. 20 >>1 = 20/2 = 10 (as 10100>>1 = 01010 = 10)

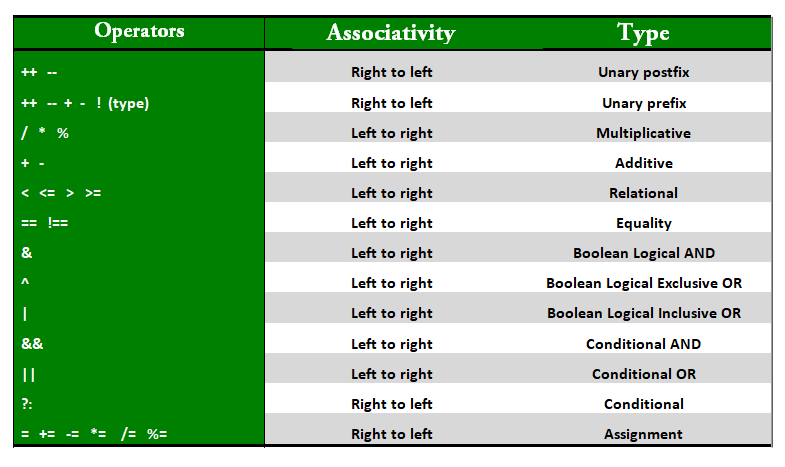
* >>>, **Unsigned Right shift operator:** shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit is set to 0.

1. **instanceof operator:** The instance of the operator is used for type checking. It can be used to test if an object is an instance of a class, a subclass, or an interface. It is also known as a comparison operator where the instance is getting compared to type returning boolean true or false.

General format-  object instanceof class/subclass/interface\_name

**Precedence and Associativity of Operators**

Precedence and associative rules are used when dealing with hybrid equations involving more than one type of operator. In such cases, these rules determine which part of the equation to consider first, as there can be many different valuations for the same equation. The below table depicts the precedence of operators in decreasing order as magnitude, with the top representing the highest precedence and the bottom showing the lowest precedence.



Example: int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;

Then (a + b \* d - e / f) = 219

As per rule: e/f -> b\*d -> a+(b\*d) -> a+(b\*d)-(e/f)

If a = b++ + c;

Then a = 10, b = 11, c =0

As first assigning b+c to a and then incrementing b.

**Comparison of Autoboxed Integer objects in Java**

Following are some interesting output questions based on comparison of Autoboxed Integer objects.

**public class Main {**

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

**Integer x = 400, y = 400;**

**if (x == y)**

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

**else**

**System.out.println("Not Same");**

**}**

**}**

/\*Output:

Not Same

\*/

//Since x and y refer to different objects, we get the output as "Not Same"

**public class Main {**

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

**Integer x = 40, y = 40;**

**if (x == y)**

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

**else**

**System.out.println("Not Same");**

**}**

**}**

/\*Output:

Same

\*/

In Java, values from -128 to 127 are cached, so the same objects are returned. The implementation of valueOf() uses cached objects if the value is between -128 to 127.

If we explicitly create Integer objects using new operator, we get the output as "Not Same". See the following Java program. In the following program, valueOf() is not used.

Integer x = new Integer(40), y = new Integer(40);

if (x == y)

System.out.println("Same");

else

System.out.println("Not Same");

Output:

Not Same

**Addition and Concatenation Using + Operator in Java**

public class GFG {

public static void main(String[] args)

{

// Print statements to illustrate addition and Concatenation

// using + operators over string and integer combination

System.out.println(2 + 0 + 1 + 6 + "GeeksforGeeks");

System.out.println("GeeksforGeeks" + 2 + 0 + 1 + 6);

System.out.println(2 + 0 + 1 + 5 + "GeeksforGeeks" + 2 + 0 + 1 + 6);

System.out.println(2 + 0 + 1 + 5 + "GeeksforGeeks" + (2 + 0 + 1 + 6));

}

}

/\*Output:

9GeeksforGeeks

GeeksforGeeks2016

8GeeksforGeeks2016

8GeeksforGeeks9

\*/

Explanation: This unpredictable output is due to the fact that the compiler evaluates the given expression from left to right given that the operators have the same precedence. Once it encounters the String, it considers the rest of the expression as of a String (again based on the precedence order of the expression).

**Basic Terminologies in Java**

**1. Class:  It can be defined as a logical template that share common properties and methods.**

**2. Object: an instance of a class. It has behavior and state.**

**3. Method: The behavior of an object is the method.**

**4. Instance variables: Every object has its own unique set of instance variables. The state of an object is generally created by the values that are assigned to these instance variables.**

**Note:**When the class is public, the name of the file has to be the class name.

**Syntax:**

**1. Comments in Java**

**There are three types of comments in Java.**

**i. Single line Comment**

**// System.out.println("This is an comment.");**

**ii. Multi-line Comment**

**/\*  
 System.out.println("This is the first line comment.");  
 System.out.println("This is the second line comment.");  
\*/**

**iii. Documentation Comment. Also called a doc comment.**

**/\*\* documentation \*/**

**2. Source File Name**

**The name of a source file should exactly match the public class name with the extension of .java.**

**3. Case Sensitivity**

**Java is a case-sensitive language, which means that the identifiers AB, Ab, aB, and ab are different in Java.**

**4. Class Names**

**i. The first letter of the class should be in Uppercase (lowercase is allowed but discouraged).**

**ii. If several words are used to form the name of the class, each inner word’s first letter should be in Uppercase. Underscores are allowed, but not recommended.**

**iii. If you are naming any class then it should be a noun.**

**5. public static void main(String [] args)**

**The method main() is the main entry point into a Java program; this is where the processing starts. Without main method, program can compile but it will not run.**

**6. Method Names**

**i. All the method names should be verb and should start with a lowercase letter (uppercase is also allowed but lowercase is recommended).**

**ii. If several words are used to form the name of the method, then each first letter of the inner word should be in Uppercase.**

**7. Identifiers in java**

**Identifiers are the names of local variables, instance and class variables, and labels, but also the names for classes, packages, modules and methods.**

**The only allowed characters for identifiers are all alphanumeric characters([A-Z],[a-z],[0-9]), '$'(dollar) and '\_' (underscore), but should not start with digits([0-9]).**

1. **Identifiers can begin with a letter, a currency symbol or an underscore (\_). According to the convention, a letter should be lower case for variables.**
2. **Constants (static, final, attributes and enums) should be in all Uppercase letters.**
3. **Identifiers are case-sensitive.**
4. **A keyword cannot be used as an identifier since it is a reserved word and has some special meaning.**

**8. Java Keywords**

**Keywords or Reserved words** are the words in a language that are used for some internal process or represent some predefined actions. These words are therefore not allowed to use as variable names or objects.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |  |  |  |  |  |  |

1. **White spaces in Java**

**A line containing only white spaces, possibly with the comment, is known as a blank line, and the Java compiler totally ignores it.**

1. **Access Modifiers:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Modifier | Within Class | Within Package | Outside Package by subclass only | Outside Package |
| Private | Yes | No | No | No |
| Default | Yes | Yes | No | No |
| Protected | Yes | Yes | Yes | No |
| Public | Yes | Yes | Yes | Yes |

**Note: Java uses CamelCase as a practice for writing names of methods, variables, classes.**

**Methods in Java:**

**A method is a collection of statements that perform some specific task and return result to the caller. A method can perform some specific task without returning anything. In Java, every method must be part of some class which is different from languages like C, C++ and Python.**



**Method Declaration:**

**Method signature: It consists of method name and parameter list (number of parameters, type of the parameters and order of the parameters). Return type and exceptions are not considered as part of it.  
Method Signature of above function:**

**max(int x, int y)**

**Object Oriented Programming (OOPs) Concept in Java**

* **Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects.**
* **The main aim of OOPs is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.**

**OOPS concepts include:**

* [**Class**](https://www.geeksforgeeks.org/classes-objects-java)
* [**Object**](https://www.geeksforgeeks.org/classes-objects-java)
* **4 Pillars of OOPs**

1. [**Abstraction**](https://www.geeksforgeeks.org/abstraction-in-java-2)
2. [**Encapsulation**](https://www.geeksforgeeks.org/encapsulation-in-java)
3. [**Inheritance**](https://www.geeksforgeeks.org/inheritance-in-java)
4. [**Polymorphism**](https://www.geeksforgeeks.org/polymorphism-in-java)

* **Compile-time polymorphism**
* **Runtime polymorphism**

1. **Abstraction:**

[**Abstraction**](https://www.geeksforgeeks.org/abstraction-in-java-2)**is the property by virtue of which only the essential details are displayed to the user. The non-essential units are not displayed to the user.**

**Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the car speed or applying brakes will stop the car, but he does not know how on pressing the accelerator, the speed is actually increasing. He does not know about the inner mechanism of the car or the implementation of the accelerators, brakes etc. in the car. This is what abstraction is.**

* **In Java, abstraction is achieved by**[**interfaces**](https://www.geeksforgeeks.org/interfaces-in-java)**and**[**abstract classes**](https://www.geeksforgeeks.org/abstract-classes-in-java)**. We can achieve 100% abstraction using interfaces.**
* **An abstract class and methods are declared with an**[**abstract keyword.**](https://www.geeksforgeeks.org/abstract-keyword-in-java/)
* **Abstract method contains only method declaration but not implementation.**
* **An abstract class may or may not have all abstract methods. Some of them can be concrete methods.**
* **A method-defined abstract must always be redefined in the subclass, thus making**[**overriding**](https://www.geeksforgeeks.org/overriding-in-java/)**compulsory or making the subclass itself abstract.**
* **The class should be abstract if a class has one or many abstract methods.**
* **Abstract class can’t be instantiated directly with the new operator. That is, there can be no object of an abstract class.**
* **An abstract class can have constructors, concrete methods, static method, and final method.**

1. **Encapsulation:**

* **Encapsulation is the process of wrapping data and functions into a single unit called class.**
* **In Encapsulation, the data is not accessed directly; it is accessed through the functions present inside the class.**
* **In encapsulation, the data in a class is hidden from other classes, which is similar to what data-hiding does. So, the terms “encapsulation” and “data-hiding” are used interchangeably.**
* **Encapsulation can be achieved by declaring all the variables in a class as private and writing public methods in the class to set and get the values of the variables.**

1. **Inheritance:**

* **Inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically. In such a way, you can reuse, extend or modify the attributes and behaviors which are defined in other classes.**
* **We can achieve inheritance by using extends keyword.**
* **Inheritance is also known as “is-a” relationship.**
* **Superclass: The class whose features are inherited is known as superclass (also known as base or parent class).**
* **Subclass: The class that inherits the other class is known as subclass (also known as derived or extended or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.**

**Java Inheritance Types:**

1. **Single Inheritance: In single inheritance, a sub-class is derived from only one super class.**
2. **Multilevel Inheritance: A derived class will be inheriting a base class, and that derived class also acts as the base class for other classes.**
3. **Hierarchical Inheritance: one class serves as a superclass (base class) for more than one subclass.**
4. **Multiple Inheritance (Through Interfaces only): one class inherits features from more than one interfaces.**
5. **Hybrid Inheritance: It is a mix of two or more of the above types of inheritance.**
6. **Polymorphism:**

* **The ability to appear in many forms is called**[**polymorphism**](https://www.geeksforgeeks.org/polymorphism-in-java)**.**
* **It refers to the ability of object-oriented programming languages to differentiate between entities with the same name efficiently.**
* **Polymorphism allows us to define one interface and have multiple implementations.**
* **This is done by Java with the help of the signature and declaration of these entities.**

**Polymorphism in Java is mainly of 2 types:**

1. **Compile-time Polymorphism (static polymorphism):**

**The polymorphism which is implemented at the compile time is known as compile-time polymorphism. This type of polymorphism is achieved by method overloading or operator overloading. But Java doesn’t support the Operator Overloading.**

**Method Overloading: Java can distinguish the methods with same name but with different parameters list in a class.**

* + **Overloaded methods are differentiated based on the number and type of the parameters passed as an arguments to the methods.**
  + **You can not define more than one method with the same name and the same number and the type of the arguments. It would be compiler error.**
  + **The compiler does not consider the return type while differentiating the overloaded method.**
  + **A method can be declared abstract using the abstract keyword and final using the final keyword but can not be both simultaneously.**

1. **Runtime Polymorphism ([Dynamic):](https://www.geeksforgeeks.org/overriding-in-java)**

**[It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding.](https://www.geeksforgeeks.org/overriding-in-java)**

**[Method Overriding: is a feature that allows a subclass or child class to](https://www.geeksforgeeks.org/overriding-in-java)**

**[provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, the same parameters or signature, and the same return type as a method in its super-class, then the method in the subclass is said to override the method in the super-class.](https://www.geeksforgeeks.org/overriding-in-java)**

**[Advantages of OOPs in Java:](https://www.geeksforgeeks.org/overriding-in-java)**

* **OOPs helps in organizing and structuring code in a more manageable way, making it easier to maintain and scale Java applications.**
* **It also promotes code reusability, modularity, and flexibility, leading to efficient and robust software development.**

**Why Java is not a purely Object-Oriented Language?**

**Pure Object Oriented Language are Fully Object Oriented Language that have features that treats everything inside the program as objects. It doesn’t support primitive datatype(like int, char, float, bool, etc.). There are seven qualities to be satisfied for a programming language to be pure object-oriented. They are:**

1. **Encapsulation/Data Hiding**
2. **Inheritance**
3. **Polymorphism**
4. **Abstraction**
5. **All predefined types are objects**
6. **All user defined types are objects**
7. **All operations performed on objects must be only through methods exposed at the objects.**

**Since Java does not support 5 and 6. So, Java is not.**

**Class**

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. Class does not occupy memory till the time an object is instantiated.

In general, class declarations can include these components, in order:

**Modifiers**: A class can be public or has default access (Refer [this](https://www.cdn.geeksforgeeks.org/access-specifiers-for-classes-or-interfaces-in-java/) for details).

**Class keyword:** class keyword is used to create a class.

**Class name:**The name should begin with an initial letter (capitalized by convention).

**Superclass**(if any): The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.

**Interfaces**(if any): A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements.

**Body**: The class body is surrounded by braces, { }. It can contain: Data member, Method, Constructor, Nested Class and Interface.

There are various types of classes that are used in real time applications such as [**nested classes**](https://www.cdn.geeksforgeeks.org/inner-class-java/)**,**[**anonymous classes**](https://www.cdn.geeksforgeeks.org/anonymous-inner-class-java/)**,**[**lambda expressions**](https://www.cdn.geeksforgeeks.org/lambda-expressions-java-8/).

**Objects (instance of a class)**

It is an instance of the class. All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.

**Ways to create an object of a class**

There are **four ways** to create objects in java. Strictly speaking there is only one way(by using new keyword), and the rest internally use new keyword.

1. **Using new keyword:** It is the most common and general way to create an object in java. Example:

// creating object of class Test

Test t = new Test();

1. **Using Class.forName(String className) method:** There is a pre-defined class in java.lang package with name Class. The forName(String className) method returns the Class object associated with the class with the given string name. We have to give a fully qualified name for a class. On calling new Instance() method on this Class object returns new instance of the class with the given string name.

// creating object of public class Test

// consider class Test present in com.p1 package

Test obj = (Test)Class.forName("com.p1.Test").newInstance();

1. **Using clone() method:** clone() method is present in Object class. It creates and returns a copy of the object.

// creating object of class Test

Test t1 = new Test();

// creating clone of above object

Test t2 = (Test)t1.clone();

1. **Deserialization**: De-serialization is a technique of reading an object from the saved state in a file.

FileInputStream file = new FileInputStream(filename);

ObjectInputStream in = new ObjectInputStream(file);

Object obj = in.readObject();

**Note:** Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object.The byte stream created is platform independent. To make a Java object serializable we implement the java.io.Serializable interface. The ObjectOutputStream class contains writeObject() method for serializing an Object. Similarly, The ObjectInputStream class contains readObject() method for deserializing an object.

* When an object is created using a new keyword, then space is allocated for the variable in a heap, and the starting address is stored in the stack memory.

**Anonymous objects:**

Anonymous objects are objects that are instantiated but are not stored in a reference variable.

* They are used for immediate method calling.
* They will be destroyed after method calling.
* They are widely used in different libraries. For example, in AWT libraries, they are used to perform some action on capturing an event(eg a key press).
* In the example below, when a key is button(referred by the btn) is pressed, we are simply creating anonymous object of EventHandler class for just calling handle method.

btn.setOnAction(new EventHandler()

{

public void handle(ActionEvent event)

{

System.out.println("Hello World!");

}

});

**‘this’ keyword :**  ‘this’ keyword in Java that refers to the current instance of the class. In OOPS it is used to:

* pass the current object as a parameter to another method
* refer to the current class instance variable

**Constructor :** Constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new objects generally.

* Constructors have the same name as class or structure.
* Constructors don’t have a return type. (Not even void)
* Constructors are only called once, at object creation.

**Java is Strictly Pass by Value?**

**For primitive data types:** Like C/C++, Java creates a copy of the variable being passed in the method and then do the manipulations -> Pass by value

**For non-primitive data types(or objects of any class):** -> Pass by reference

Example:

class Test

{

int x;

Test(int i) { x = i; }

Test() { x = 0; }

}

class Main

{

public static void main(String[] args)

{

// t is a reference

Test t = new Test(5);

// Reference is passed and a copy of reference is created in change()

change(t);

// New value of x is printed

System.out.println(t.x); // t.x = 10

}

// This change() doesn't change the reference, it only

// changes member of object referred by reference

public static void change(Test t)

{

t.x = 10;

}

}

**Note**: if we set some other reference to object passed inside method, the object from calling method as well its reference will remain unaffected. The changes are not reflected back if we change the object itself to refer some other location or object.

Example:

class Main {

// swap() doesn't swap i and j

public static void swap(Integer i, Integer j)

{

Integer temp = new Integer(i); // creating new reference

i = j;

j = temp; // i =20, j = 10

}

public static void main(String[] args)

{

Integer i = new Integer(10);

Integer j = new Integer(20);

swap(i, j);

System.out.println("i = " + i + ", j = " + j); // i = 10, j = 20

}

}

**Packages In Java:**

**Package** in [Java](https://www.geeksforgeeks.org/java/) is a mechanism to encapsulate a group of classes, sub packages and interfaces. Packages are used for:

* Preventing naming conflicts.
* Making searching/locating and usage of classes, interfaces, enumerations and annotations easier
* Providing controlled access: protected and default have package level access control.
* Packages can be considered as data encapsulation (or data-hiding).

**Package Naming:** Packages are named in reverse order of domain names, i.e., root\_directory.parent\_package.package\_name. For example, in a college, the recommended convention is college.tech.cse, college.tech.ee, etc.

**Adding a class to a Package :**We can add more classes to a created package by using package name at the top of the program and saving it in the package directory.

**Subpackages:**Packages that are inside another package are the **subpackages**. These are not imported by default, they have to imported explicitly. Also, members of a subpackage have no access privileges, i.e., they are considered as different package for protected and default access specifiers.

**Built-in Packages:**

1. **java.lang:**Contains language support classes(e.g classes which defines primitive data types, math operations). This package is automatically imported.
2. **java.io:**Contains classes for supporting input / output operations.
3. **java.util:**Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.
4. **java.applet:**Contains classes for creating Applets.
5. **java.awt:**Contain classes for implementing the components for graphical user interfaces (like button , ;menus etc). 6)
6. **java.net:**Contain classes for supporting networking operations.

**Wrapper Classes in Java**

A Wrapper class is a class whose object wraps or contains primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store primitive data types.

In other words, we can wrap a primitive value into a wrapper class object.

**Need of Wrapper Classes:**

1. They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).
2. The classes in java.util package handles only objects and hence wrapper classes help in this case also.
3. Data structures in the Collection framework, such as [ArrayList](https://www.cdn.geeksforgeeks.org/arraylist-in-java/) and [Vector](https://www.cdn.geeksforgeeks.org/vector-vs-arraylist-java/), store only objects (reference types) and not primitive types.
4. An object is needed to support synchronization in multithreading.



**Autoboxing**: Automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example – conversion of int to Integer, long to Long, double to Double etc.

char ch = 'a';

*// Autoboxing- primitive to Character object conversion* Character a = ch;

**Unboxing**: It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing. For example – conversion of Integer to int, Long to long, Double to double, etc.

Character ch = 'a';

*// unboxing - Character object to primitive conversion*

char a = ch;

**enum in Java**

* Enumerations serve the purpose of representing a group of named constants in a programming language. For example, the 4 suits in a deck of playing cards may be 4 enumerators named Club, Diamond, Heart, and Spade, belonging to an enumerated type named Suit.    
  Enums are used when we know all possible values at compile time, such as choices on a menu, rounding modes, command-line flags, etc. It is not necessary that the set of constants in an enum type stay fixed for all time.
* A Java enumeration is a class type. Although we don't need to instantiate an enum using new, it has the same capabilities as other classes. Just like classes, you can give them constructor, add instance variables and methods, and even implement interfaces.
* One thing to keep in mind is that, unlike classes, enumerations neither inherit other classes nor can get extended(i.e become superclass).
* In Java (from 1.5), enums are represented using enum data type.
* The main objective of enum is to define our own data types(Enumerated Data Types).
* Declaration of enum in Java can be done outside a Class or inside a Class but not inside a Method.

// A simple enum example where enum is declared

// outside any class (Note enum keyword instead of class keyword)

enum Color {

RED,

GREEN,

BLUE;

}

public class Test {

// Driver method

public static void main(String[] args)

{

Color c1 = Color.RED;

System.out.println(c1); //output= RED

}

}

**Important Points of enum:**

* Every enum is internally implemented by using Class.

/\* internally above enum Color is converted to

class Color

{

public static final Color RED = new Color();

public static final Color BLUE = new Color();

public static final Color GREEN = new Color();

}\*/

* Every enum constant represents an object of type enum.
* enum type can be passed as an argument to switch statements.
* Every enum constant is always implicitly public static final. Since it is static, we can access it by using the enum Name. Since it is final, we can't create child enums.
* We can declare the main() method inside the enum. Hence we can invoke enum directly from the Command Prompt.
* All enums implicitly extend **java.lang.Enum** class.
* Methods:
  + values() method -> return an array of all values present inside the enum.
  + ordinal() method -> return each enum constant index, just like an array index.
  + valueOf(String) method -> returns the enum constant of the specified string value if exists.

**enum and constructor:**

* enum can contain a constructor and it is executed separately for each enum constant at the time of enum class loading.
* We can't create enum objects explicitly and hence we can't invoke enum constructor directly.

**enum and methods:**

* enum can contain both concrete methods and abstract methods. If an enum class has an abstract method, then each instance of the enum class must implement it

// Java program to demonstrate working of values(), ordinal() and valueOf()

enum Color {

RED,

GREEN,

BLUE;

}

public class Test {

public static void main(String[] args)

{

// Calling values()

Color arr[] = Color.values();

// enum with loop

for (Color col : arr) {

// Calling ordinal() to find index of color.

System.out.println(col + " at index "

+ col.ordinal());

}

// Using valueOf(). Returns an object of Color with given constant.

// Uncommenting second line causes exception IllegalArgumentException

System.out.println(Color.valueOf("RED"));

// System.out.println(Color.valueOf("WHITE"));

}

}

/\* Output:

RED at index 0

GREEN at index 1

BLUE at index 2

RED

\*/

**Enum with Customized Value in Java**

By default enums have their own string values, we can also assign some custom values to enums. Consider below example for that.

enum Fruits

{

APPLE(“RED”), BANANA(“YELLOW”), GRAPES(“GREEN”);

}

In above example we can see that the Fruits enum have three members i.e APPLE, BANANA and GRAPES with have their own different custom values RED, YELLOW and GREEN respectively.  
  
Now to use this enum in code, there are some points we have to follow:-

* We have to create parameterized constructor for this enum class. Why? Because as we know that enum class’s object can’t be create explicitly so for initializing we use parameterized constructor. And the constructor cannot be the public or protected it must have private or default modifiers. Why? if we create public or protected, it will allow initializing more than one objects. This is totally against enum concept.
* We have to create one getter method to get the value of enums.

// Java program to demonstrate how values can be assigned to enums.

enum TrafficSignal

{

// This will call enum constructor with one String argument

RED("STOP"), GREEN("GO"), ORANGE("SLOW DOWN");

// declaring private variable for getting values

private String action;

// getter method

public String getAction()

{

return this.action;

}

// enum constructor - cannot be public or protected

private TrafficSignal(String action)

{

this.action = action;

}

}

// Driver code

public class EnumConstructorExample

{

public static void main(String args[])

{

// let's print name of each enum and there action

TrafficSignal[] signals = TrafficSignal.values();

for (TrafficSignal signal : signals)

{

// use getter method to get the value

System.out.println("name : " + signal.name() +

" action: " + signal.getAction() );

}

}

}

/\* Output:

name : RED action: STOP

name : GREEN action: GO

name : ORANGE action: SLOW DOWN

\*/

###### Ways to read input from console in Java

In Java, there are four different ways for reading input from the user in the command line environment(console).

1. **Using Scanner Class:** (easy, less typing, but not recommended very slow)

* Scanner is a class in java.util package used for obtaining the input of the primitive types like int, double, etc. and strings.
* It is the easiest way to read input in a Java program, though not very efficient if you want an input method for scenarios where time is a constraint like in competitive programming.
* To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream. We may pass an object of class File if we want to read input from a file.
* To read numerical values of a certain data type XYZ, the function to use is nextXYZ().

// Java program to read data of various types using Scanner class.

import java.util.Scanner;

public class ScannerDemo1

{

public static void main(String[] args)

{

// Declare the object and initialize with predefined standard input object

Scanner sc = new Scanner(System.in);

// String input

String name = sc.nextLine();

// Character input

char gender = sc.next().charAt(0);

// Numerical data input

byte b1 = sc.nextByte();

short s1 = sc.nextShort();

int age = sc.nextInt();

long mobileNo = sc.nextLong();

Float totalMarks = sc.nextFloat();

double cgpa = sc.nextDouble();

}

}

/\* Input :

Himanshu Gupta

M

10

1998

26

8561859674

755.5

7.5

\*/

For example, to read a value of type short, we can use nextShort()

To read strings, we use nextLine().

To read a single character, we use next().charAt(0). next() function returns the next token/word in the input as a string and charAt(0) function returns the first character in that string.

* The Scanner class reads an entire line and divides the line into tokens. Tokens are small elements that have some meaning to the Java compiler.

For example,

Suppose there is an input string: “How are you”  
In this case, the scanner object will read the entire line and divides the string into tokens: "How", "are" and "you". The object then iterates over each token and reads each token using its different methods.

* Sometimes, we have to check if the next value we read is of a certain type or if the input has ended (EOF marker encountered).
* Then, we check if the scanner's input is of the type we want with the help of hasNextXYZ() functions where XYZ is the type we are interested in. The function returns true if the scanner has a token of that type, otherwise false. For example, in the below code, we have used hasNextInt(). To check for a string, we use hasNextLine(). Similarly, to check for a single character, we use hasNext().charAt(0).

1. **Using Buffered Reader Class:** (fast, but requires a lot of typing)

* The Java.io.BufferedReader class reads text from a character-input stream, buffering characters to provide for the efficient reading of characters, arrays, and lines.
* With this method, we will have to parse the value every time for the desired type. Reading multiple words from a single line adds to its complexity because of the use of Stringtokenizer.

//Java program to demonstrate BufferedReader

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.IOException;

public class BufferedReaderExample {

public static void main (String[] args) throws IOException

{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String name = br.readLine();

char gender = br.readLine().charAt(0);

int age = Integer.parseInt(br.readLine());

float cgpa = Float.parseFloat(br.readLine());

// To read multiple value

String[] books = br.readLine().split(" ");

}

}

**3. Using Console Class:**

* It has been becoming a preferred way for reading user’s input from the command line.
* In addition, it can be used for reading password-like input without echoing the characters entered by the user; the format string syntax can also be used (like System.out.printf()).
* Does not work in non-interactive environment (such as in an IDE).

// Java program to demonstrate working of System.console()

// Note that this program does not work on IDEs as

// System.console() may require console

public class Sample {

public static void main(String[] args)

{

// Using Console to input data from user

String name = System.console().readLine();

System.out.println("You entered string " + name);

}

}

**4.**[**Using Command line argument**](https://www.geeksforgeeks.org/command-line-arguments-in-java/)**:**

Most used user input for competitive coding. The command-line arguments are stored in the String format. The parseInt method of the Integer class converts string argument into Integer. Similarly, for float and others during execution. The usage of args[] comes into existence in this input form. The passing of information takes place during the program run. The command line is given to args[]. These programs have to be run on cmd.

// Program to check for command line arguments

class Hello {

public static void main(String[] args)

{

// check if length of args array is greater than 0

if (args.length > 0) {

System.out.println("The command line arguments are:");

// iterating the args array and printing the command line arguments

for (String val : args)

System.out.println(val);

}

else

System.out.println("No command line arguments found.");

}

}

cmd: java Main Hello World

**Character Stream Vs Byte Stream in Java**

A stream is a sequence of data. [I/O Stream](https://www.geeksforgeeks.org/java-io-input-output-in-java-with-examples/) refers to a stream that is unlikely a method to sequentially access a file. I/O Stream means an input source or output destination representing different types of sources e.g. disk files. The java.io package provides classes that allow you to convert between Unicode character streams and byte streams of non-Unicode text.

**When to use Character Stream over Byte Stream?**

In Java, characters are stored using Unicode conventions. Character stream is useful when we want to process text files. These text files can be processed character by character. Character size is typically 16 bits.

**When to use Byte Stream over Character Stream?**

Byte oriented reads byte by byte.  A byte stream is suitable for processing raw data like binary files.

**Key points while using and dealing with any of the above streams are as follows:**

1. Names of character streams typically end with Reader/Writer and names of byte streams end with InputStream/OutputStream
2. The streams used in example codes are unbuffered streams and less efficient. We typically use them with buffered readers/writers for efficiency. We will soon be discussing use BufferedReader/BufferedWriter (for character stream) and BufferedInputStream/BufferedOutputStream (for byte stream) classes.
3. It is always recommended to close the stream if it is no longer in use. This ensures that the streams won't be affected if any error occurs.
4. The above codes may not run in online compilers as files may not exist.

### ****Character Stream****

// Java Program illustrate Reading

// a File in Human Readable

// Format Using FileReader Class

// Importing required classes

import java.io.\*;

// Main class

public class GFG {

// Main driver method

public static void main(String[] args)

throws IOException

{

// Initially assigning null as we have not read

// anything

FileReader sourceStream = null;

// Try block to check for exceptions

try {

// Reading from file

sourceStream = new FileReader(

"/Users/mayanksolanki/Desktop/demo.rtf");

// Reading sourcefile and writing content to

// target file character by character.

int temp;

// If there is content inside file

// than read

while ((temp = sourceStream.read()) != -1)

System.out.println((char)temp);

// Display message for successful execution of program

System.out.print("Program successfully executed");

}

// finally block that executes for sure

// where we are closing file connections

// to avoid memory leakage

finally {

// Closing stream as no longer in use

if (sourceStream != null)

sourceStream.close();

}

}

}

### ****Byte Stream****

// Java Program Illustrate ByteStream Class to

// Copy Contents of One File to Another File

// Importing required classes

import java.io.\*;

// Main class

public class GFG {

// Main driver method

public static void main(String[] args)

throws IOException

{

// Initially assigning null ot objects for

// reading and writing to file

FileInputStream sourceStream = null;

FileOutputStream targetStream = null;

// Try block to check for exceptions

try {

// Passing the files via local directory

sourceStream = new FileInputStream(

"/Users/mayanksolanki/Desktop/demo.rtf");

targetStream = new FileOutputStream(

"/Users/mayanksolanki/Desktop/democopy.rtf");

// Reading source file and writing content to

// target file byte by byte

int temp;

// If there is content inside file

// than read

while ((temp = sourceStream.read()) != -1)

targetStream.write((byte)temp);

// Display message for successful execution of program

System.out.print("Program successfully executed");

}

// finally block that executes for sure

// where we are closing file connections

// to avoid memory leakage

finally {

if (sourceStream != null)

sourceStream.close();

if (targetStream != null)

targetStream.close();

}

}

}

###### Formatted output in Java

There are different ways in which we can format output in Java. Some of them are given below.

1. Using System.out.printf()
2. Using DecimalFormat class
3. Using SimpleDateFormat class (for formatting Dates)

**1. Formatting output using System.out.printf()**

This is the easiest of all methods as this is similar to printf in C. Note that System.out.print() and System.out.println() take a single argument, but printf() may take multiple arguments.

class JavaFormatter1 {

public static void main(String args[])

{

int x = 100;

System.out.printf("Printing simple integer: x = %d\n", x);

// this will print it upto 2 decimal places

System.out.printf("Formatted with precision: PI = %.2f\n", Math.PI);

float n = 5.2f;

// automatically appends zero to the rightmost part of decimal

System.out.printf( "Formatted to specific width: n = %.4f\n", n);

n = 2324435.3f;

// here number is formatted from right margin and occupies a width of 20 characters

System.out.printf("Formatted to right margin: n = %20.4f\n", n);

}

}

/\* Output:

Printing simple integer: x = 100

Formatted with precision: PI = 3.14

Formatted to specific width: n = 5.2000

Formatted to right margin: n = 2324435.2500

\*/

**Note**: System.out.format() is equivalent to printf() and can also be used.

1. **Formatting using DecimalFormat class:**

// Java program to demonstrate working of DecimalFormat

import java.text.DecimalFormat;

class JavaFormatter2 {

public static void main(String args[])

{

double num = 123.4567;

// prints only numeric part of a floating number

DecimalFormat ft = new DecimalFormat("####");

System.out.println("Without fraction part: num = " + ft.format(num));

// this will print it upto 2 decimal places

ft = new DecimalFormat("#.##");

System.out.println("Formatted to Give precision: num = "+ ft.format(num));

// automatically appends zero to the rightmost part of decimal instead of #,we use digit 0

ft = new DecimalFormat("#.000000");

System.out.println("appended zeroes to right: num = "+ ft.format(num));

// automatically appends zero to the leftmost of decimal number instead of #,we use digit 0

ft = new DecimalFormat("00000.00");

System.out.println("formatting Numeric part : num = "+ ft.format(num));

// formatting money in dollars

double income = 23456.789;

ft = new DecimalFormat("$###,###.##");

System.out.println("your Formatted Dream Income : "+ ft.format(income));

}

}

/\**Output:*

Without fraction part: num = 123

Formatted to Give precision: num = 123.46

appended zeroes to right: num = 123.456700

formatting Numeric part : num = 00123.46

your Formatted Dream Income : $23,456.79

\*/

**3. Formatting dates and parsing using SimpleDateFormat class**

// Java program to demonstrate working of SimpleDateFormat

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

class Formatter3 {

public static void main(String args[])

throws ParseException

{

// Formatting as per given pattern in the argument

SimpleDateFormat ft

= new SimpleDateFormat("dd-MM-yyyy");

String str = ft.format(new Date());

System.out.println("Formatted Date : " + str);

// parsing a given String

str = "02/18/1995";

ft = new SimpleDateFormat("MM/dd/yyyy");

Date date = ft.parse(str);

// this will print the date as per parsed string

System.out.println("Parsed Date : " + date);

}

}

/\*Output:

Formatted Date : 24-01-2022

Parsed Date : Sat Feb 18 00:00:00 UTC 1995

\*/

**Loops in Java**

Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true.

Java provides three ways for executing the loops.

1. [**while loop:**](https://www.geeksforgeeks.org/java-while-loop-with-examples/)

A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

Syntax :

while (boolean condition)

{

loop statements...

}

* While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called **Entry control loop.**
* Once the condition is evaluated to true, the statements in the loop body are executed. Normally the statements contain an update value for the variable being processed for the next iteration.
* When the condition becomes false, the loop terminates which marks the end of its life cycle.
* While loop in Java comes into use when we need to repeatedly execute a block of statements, or when the number of iterations are not fixed.

1. [**do while:**](https://www.geeksforgeeks.org/java-do-while-loop-with-examples/)

do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of **Exit Control Loop**.

Syntax:

do

{

statements..

}

while (condition);

* It is important to note that the do-while loop will execute its statements atleast once before any condition is checked.

1. [**for loop:**](https://www.geeksforgeeks.org/java-for-loop-with-examples/)

*for* loop provides a concise way of writing the loop structure. Unlike a *while* loop, a *for* statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

Syntax:

for (initialization condition; testing condition;

increment/decrement)

{

statement(s)

}

* 1. **For-each loop in Java:**
* It starts with the keyword for like a normal for-loop.
* Instead of declaring and initializing a loop counter variable, you declare a variable that is the same type as the base type of the array, followed by a colon, which is then followed by the array name.
* In the loop body, you can use the loop variable you created rather than using an indexed array element.
* It's commonly used to iterate over an array or a Collections class (eg, ArrayList).

Syntax:

*for (type var : array)*

*{*

*statements using var;*

*}*

**Limitations of for-each loop :**

1. For-each loops are not appropriate when you want to modify the array.
2. For-each loops do not keep track of index. So we can not obtain array index using For-Each loop.
3. For-each only iterates forward over the array in single steps.
4. For-each cannot process two decision making statements at once.
5. For-each also has some performance overhead over simple iteration:

**Decision Making in Java (if, if-else, switch, break, continue, jump)**

1. **if:**

if statement is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

Syntax:

if(condition)

{

// Statements to execute if

// condition is true

}

* If we do not provide the curly braces '{' and '}' after if( condition ) then bydefault if statement will consider the immediate one statement to be inside its block.

1. **if-else:**

We can use the else statement with if statement to execute a block of code when the condition is false.

Syntax:

if (condition)

{

// Executes this block if

// condition is true

}

else

{

// Executes this block if

// condition is false

}

1. **if-else-if ladder:**

Here, a user can decide among multiple options.The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

Syntax:

if (condition)

statement;

else if (condition)

statement;

.

.

else

statement;

1. **switch-case:**

The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.

Syntax:

switch (expression)

{

case value1:

statement1;

break;

case value2:

statement2;

break;

.

.

case valueN:

statementN;

break;

default:

statementDefault;

}

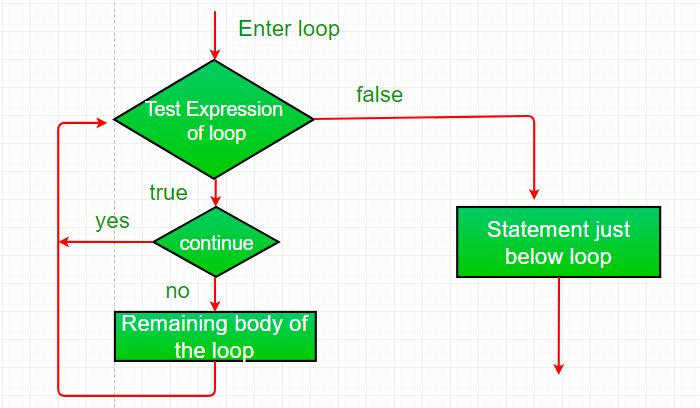
* The expression can be of type byte, short, int, char, String or an enumeration.
* Duplicate case values are not allowed.
* The default statement is optional.
* The break statement is optional. If omitted, execution will continue on into the next case.

1. **jump:**Java supports three jump statements: break, continue and return.

**Break**: In Java, a break is majorly used for:

* Terminate a sequence in a switch statement (discussed above).
* To exit a loop.
* Used as a "civilized" form of goto.

**Continue**: Sometimes it is useful to force an early iteration of a loop. That is, you might want to continue running the loop but stop processing the remainder of the code in its body for this particular iteration.



[**Return**:](https://www.geeksforgeeks.org/return-keyword-java/) The return statement is used to explicitly return from a method. That is, it causes program control to transfer back to the caller of the method.

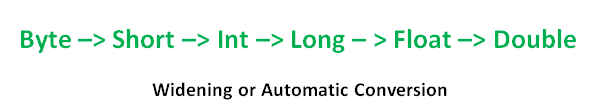
**Type conversion in Java**

When you assign a value of one data type to another, the two types might not be compatible with each other. If the data types are compatible, then Java will perform the conversion automatically known as Automatic Type Conversion, and if not then they need to be cast or converted explicitly.

1. **Widening or Automatic Type Conversion:**

* When the two data types are compatible.
* When we assign a value of a smaller data type to a bigger data type.

For Example, in java, the numeric data types are compatible with each other but no automatic conversion is supported from numeric type to char or boolean. Also, char and boolean are not compatible with each other.



Example:

int i = 100;

// Automatic type conversion Integer to long type

long l = i;

// Automatic type conversion long to float type

float f = l;

// Automatic type conversion char to int type

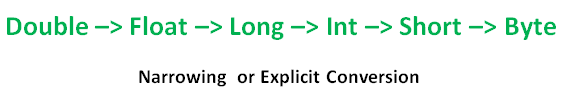
Char c = ‘a’;

Int x = c; // ASCII value of ‘a’ will get

1. **Narrowing or Explicit Conversion:**

* If we want to assign a value of a larger data type to a smaller data type we perform explicit type casting or narrowing.
* This is useful for incompatible data types where automatic conversion cannot be done.

Here, the target type specifies the desired type to convert the specified value to.



Example:

// Double datatype

double d = 100.04;

// Explicit type casting by forcefully getting data from double datatype to long type long l = (long)d;

// Explicit type casting

int i = (int)l;

**Note**: While assigning value to byte type the fractional part is lost and is reduced to modulo 256(range of byte).

Example:

int i = 257;

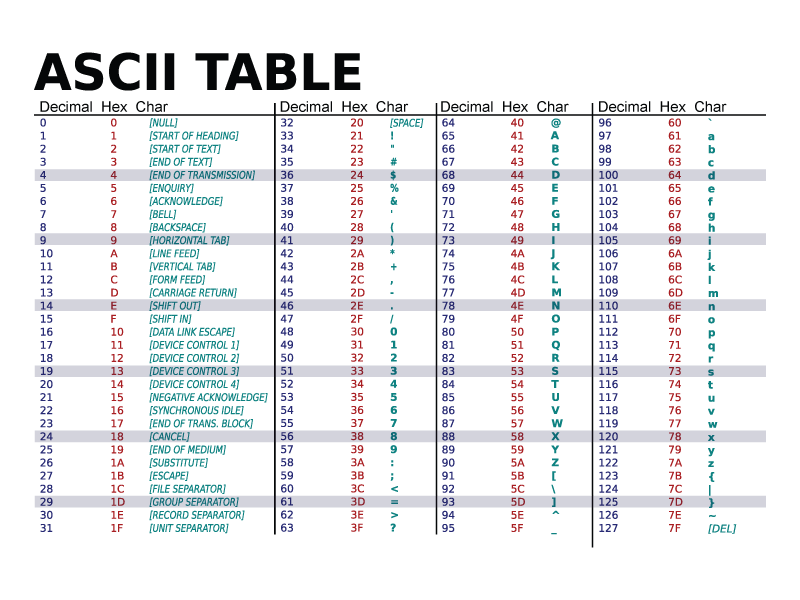
//Explicit byte conversion

byte b = (byte) i; // b = 257%256 = 1

**ASCII Values**

ASCII (American Standard Code for Information Interchange) is a standard character encoding used in telecommunication. The ASCII pronounced ‘ask-ee’, is strictly a seven-bit code based on the English alphabet. ASCII codes are used to represent alphanumeric data.

Since it is a seven-bit code, it can at the most represent 128 characters. it currently defines 95 printable characters including 26 upper case letters (A to Z), 26 lower case letters, 10 numerals (0 to 9), and 33 special characters including mathematical symbols, punctuation marks, and space characters.



**Array Data Structure**

* An array is a collection of items of same data type stored at contiguous memory locations.
* This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array). The base value is index 0 and the difference between the two indexes is the offset.
* the array has a fixed size meaning once the size is given to it, it cannot be changed.
* In java, arrays are first class objects. So, arrays are alloted on heap.
* In java, it is not allowed to put the size of array in declaration.

i. e. int arr[5] will error.

**Initialization of an Array:**

By default the array is uninitialized, and no elements of the array are set to any value. However, for the proper working of the array, array initialization becomes important. Array initialization can be done by the following methods:

**1. Passing no value within the initializer:** One can initialize the array by defining the size of the array and passing no values within the initializer.

Syntax:

int arr[ 5 ] = {  };

**2. By passing specific values within the initializer:** One can initialize the array by defining the size of the array and passing specific values within the initializer.

Syntax:

int arr[ 5 ] = { 1 , 2 , 3 , 4 , 5 };

Note:   
If the count of elements within the "{ }" is less than the size of the array, the remaining positions are considered to be '0'.

Syntax:

int arr[ 5 ] = { 1 , 2 , 3 } ;

**3. By passing specific values within the initializer but not declaring the size:** One can initialize the array by passing specific values within the initializer and not particularly mentioning the size, the size is interpreted by the compiler.

Syntax:

int arr[  ] = { 1 , 2 , 3 , 4 , 5 };

**4. Universal Initialization:**After the adoption of universal initialization in C++, one can avoid using the equals sign between the declaration and the initializer.

Syntax:

int arr[ ]  { 1 , 2 , 3 , 4 , 5 };

**Different operations on the array:**

Arrays allow random access to elements. This makes accessing elements by position faster. Hence operation like searching, insertion, and access becomes really efficient. Array elements can be accessed using the loops.

**1. Insertion in Array:**

We try to insert a value to a particular array index position, as the array provides random access it can be done easily using the assignment operator.

Pseudo Code:

// to insert a value= 10 at index position 2;

arr[ 2 ] = 10;

**2. Access elements in Array:**

Pseudo Code:

// to access array element at index position 2, we simply can write

return arr[ 2 ] ;

**3. Searching in Array:**

We try to find a particular value in the array, in order to do that we need to access all the array elements and look for the particular value.

Pseudo Code:

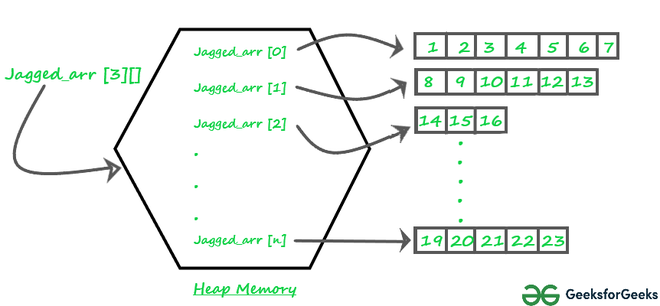
// searching for value 2 in the array;

Loop from i = 0 to arr.length:  
 check if  arr[i] = 2:  
        return true;

**Jagged Array in Java:**

A jagged[array](https://en.wikipedia.org/wiki/Jagged_array) is an array of arrays such that member arrays can be of different sizes, i.e., we can create a 2-D array but with a variable number of columns in each row. These types of arrays are also known as Jagged arrays.

Pictorial representation of Jagged array in Memory:



Syntax: data\_type array\_name[][] = new data\_type[n][]; //n: no. of rows

array\_name[] = new data\_type[n1] //n1= no. of columns in row-1

array\_name[] = new data\_type[n2] //n2= no. of columns in row-2

array\_name[] = new data\_type[n3] //n3= no. of columns in row-3

.

.

.

array\_name[] = new data\_type[nk] //nk=no. of columns in row-n

**Alternative, ways to Initialize a Jagged array :**

int arr\_name[][] = new int[][] {

new int[] {10, 20, 30 ,40},

new int[] {50, 60, 70, 80, 90, 100},

new int[] {110, 120}

};

OR

int[][] arr\_name = {

new int[] {10, 20, 30 ,40},

new int[] {50, 60, 70, 80, 90, 100},

new int[] {110, 120}

};

OR

int[][] arr\_name = {

{10, 20, 30 ,40},

{50, 60, 70, 80, 90, 100},

{110, 120}

};

**Multidimensional Arrays in Java**

* Multidimensional Arrays can be defined in simple words as array of arrays.
* Data in multidimensional arrays are stored in tabular form (in row major order).

Syntax:



Examples:

Two dimensional array:

int[][] twoD\_arr = new int[10][20];

Three dimensional array:

int[][][] threeD\_arr = new int[10][20][30];

**Size of multidimensional arrays:** The total number of elements that can be stored in a multidimensional array can be calculated by multiplying the size of all the dimensions.  
For example:  
The array int[][] x = new int[10][20] can store a total of (10\*20) = 200 elements.

1. **Two - dimensional Array (2D-Array)**

Syntax:

data\_type[][] array\_name = new data\_type[x][y];

For example: int[][] arr = new int[10][20];

OR

data\_type[][] array\_name = {

{valueR1C1, valueR1C2, ....}, {valueR2C1, valueR2C2, ....}

};

For example: int[][] arr = {{1, 2}, {3, 4}};

**Accessing Elements of Two-Dimensional Arrays**

Elements in two-dimensional arrays are commonly referred by x[i][j] where 'i' is the row number and 'j' is the column number.

Syntax:

x[row\_index][column\_index]

**Representation of 2D array in Tabular Format:**

A two - dimensional array can be seen as a table with 'x' rows and 'y' columns where the row number ranges from 0 to (x-1) and column number ranges from 0 to (y-1). A two - dimensional array 'x' with 3 rows and 3 columns is shown below:

[](https://media.geeksforgeeks.org/wp-content/uploads/two-d.png)

**Print 2D array in tabular format:**

To output all the elements of a Two-Dimensional array, use nested for loops. For this two *for* loops are required, One to traverse the rows and another to traverse columns.

public static void main(String[] args) {

int[][] arr = { { 1, 2 }, { 3, 4 } };

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

for (int j = 0; j < arr[i].length; j++)

{

System.out.print(arr[i][j] + " ");

}

System.out.println();

}

}

**String class in Java**

String is a class in Java that represents a immutable sequence of characters, which means a constant and cannot be changed once created.

**Creating a String:**

1. **String literal**

String s = “GeeksforGeeks”;

1. **Using new keyword**

String s = new String (“GeeksforGeeks”);

**Constructors**

1. **String(byte[] byte\_arr)** - Construct a new String by decoding the byte array. It uses the platform's default character set for decoding.

Example:

byte[] b\_arr = {71, 101, 101, 107, 115};

String s\_byte =new String(b\_arr); //Geeks

1. **String(byte[] byte\_arr, Charset char\_set)** - Construct a new String by decoding the byte array. It uses the char\_set for decoding.

Example:

byte[] b\_arr = {71, 101, 101, 107, 115};

Charset cs = Charset.defaultCharset();

String s\_byte\_char = new String(b\_arr, cs); //Geeks

1. **String(byte[] byte\_arr, String char\_set\_name)** - Construct a new String by decoding the byte array. It uses the char\_set\_name for decoding. It looks similar to the above constructs and they appear before similar functions but it takes the String(which contains char\_set\_name) as parameter while the above constructor takes CharSet.

Example:

byte[] b\_arr = {71, 101, 101, 107, 115};

String s = new String(b\_arr, "US-ASCII"); //Geeks

1. **String(byte[] byte\_arr, int start\_index, int length)** - Construct a new string from the bytes array depending on the start\_index(Starting location) and length(number of characters from starting location).

Example:

byte[] b\_arr = {71, 101, 101, 107, 115};

String s = new String(b\_arr, 1, 3); // eek

1. **String(byte[] byte\_arr, int start\_index, int length, Charset char\_set)**- Construct a new string from the bytes array depending on the start\_index(Starting location) and length(number of characters from starting location).Uses char\_set for decoding.

Example:

byte[] b\_arr = {71, 101, 101, 107, 115};

Charset cs = Charset.defaultCharset();

String s = new String(b\_arr, 1, 3, cs); // eek

1. **String(byte[] byte\_arr, int start\_index, int length, String char\_set\_name) -** Construct a new string from the bytes array depending on the start\_index(Starting location) and length(number of characters from starting location).Uses char\_set\_name for decoding.

Example:

byte[] b\_arr = {71, 101, 101, 107, 115};

String s = new String(b\_arr, 1, 4, "US-ASCII"); // eeks

1. **String(char[] char\_arr)** - Allocates a new String from the given Character array.

Example:

char char\_arr[] = {'G', 'e', 'e', 'k', 's'};

String s = new String(char\_arr); //Geeks

1. **String(char[] char\_array, int start\_index, int count)**- Allocates a String from a given character array but choose count characters from the start\_index.

Example:

char char\_arr[] = {'G', 'e', 'e', 'k', 's'};

String s = new String(char\_arr , 1, 3); //eek

1. **String(int[] uni\_code\_points, int offset, int count)**- Allocates a String from a uni\_code\_array but choose count characters from the start\_index.

Example:

int[] uni\_code = {71, 101, 101, 107, 115};

String s = new String(uni\_code, 1, 3); //eek

1. **String(StringBuffer s\_buffer)**- Allocates a new string from the string in s\_buffer.

Example:

StringBuffer s\_buffer = new StringBuffer("Geeks");

String s = new String(s\_buffer); //Geeks

1. **String(StringBuilder s\_builder)**- Allocates a new string from the string in s\_builder.

Example:

StringBuilder s\_builder = new StringBuilder("Geeks");

String s = new String(s\_builder); //Geeks

**String Methods**

1. **int length():** Returns the number of characters in the String.

"GeeksforGeeks".length();  // returns 13

1. [**Char charAt(int i)**](https://www.geeksforgeeks.org/java-string-charat-method-example/)**:**Returns the character at ith index.

"GeeksforGeeks".charAt(3); // returns  ‘k’

1. [**String concat( String str)**](https://www.geeksforgeeks.org/java-string-concat-examples/)**:** Concatenates specified string to the end of this string.

String s1 = ”Geeks”;

String s2 = ”forGeeks”;

String output = s1.concat(s2); // returns “GeeksforGeeks”

1. **boolean equals( Object otherObj):** Compares this string to the specified object.

boolean out = “Geeks”.equals(“Geeks”); // returns true

boolean out = “Geeks”.equals(“geeks”); // returns false

1. [**boolean  equalsIgnoreCase (String anotherString)**](https://www.geeksforgeeks.org/equalsignorecase-in-java/)**:** Compares string to another string, ignoring case considerations.

boolean out= “Geeks”.equalsIgnoreCase(“Geeks”); // returns true

boolean out = “Geeks”.equalsIgnoreCase(“geeks”); // returns true

1. [**int compareTo( String anotherString)**](https://www.geeksforgeeks.org/java-lang-string-compareto/)**:** Compares two string lexicographically.

int out = s1.compareTo(s2);  // where s1 ans s2 are strings to be compared.

This returns difference s1-s2. If :

out < 0 // s1 comes before s2 or s1 < s2

out = 0 // s1 and s2 are equal. Or s1 = s2

out > 0 // s1 comes after s2. or s1>s2

1. **int compareToIgnoreCase( String anotherString):**Compares two string lexicographically, ignoring case considerations.

int out = s1.compareToIgnoreCase(s2);

Note- In this case, it will not consider case of a letter (it will ignore whether it is uppercase or lowercase).

1. **boolean contains(**[**CharSequence**](https://docs.oracle.com/javase/8/docs/api/java/lang/CharSequence.html)**s):** Returns true if and only if this string contains the specified sequence of char values.

“himanshu”.contains(“ansh”); // return true

1. **static**[**String**](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html)**copyValueOf(char[] data):** Returns a String that contains the characters of the character array. Equivalent to [valueOf(char[])](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#valueOf-char:A-).

char[] cArray = {'a', 'b','c','d','e'};

System.out.println(String.copyValueOf(cArray)); // returns “abcde”

1. **boolean endsWith(**[**String**](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html)**suffix):** Tests if this string ends with the specified suffix.

System.out.println("guptaji".endsWith("ji")); // true

1. **byte[] getBytes():** Encodes this String into a sequence of bytes using the platform's default charset, and returning the result into a new byte array.

for(byte b: "abcABC".getBytes()) {

System.out.print(b+" "); } //returns {97 98 99 65 66 67}

1. **void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin):** Copies characters from this string into the destination character array. The first character to be copied is at index srcBegin; the last character to be copied is at index srcEnd-1 (thus the total number of characters to be copied is srcEnd-srcBegin). The characters are copied into the subarray of dst starting at index dstBegin and ending at index: dstBegin + (srcEnd-srcBegin) - 1

char[] dst = new char[3];

"himanshu".getChars(1, 4, dst, 0); // dst = {i,m,a}

1. **int hashCode():** Returns a hash code for this string. The hash code for a String object is computed as

s[0]\*31^(n-1) + s[1]\*31^(n-2) + ... + s[n-1]

using int arithmetic, where s[i] is the ith character of the string, n is the length of the string, and ^ indicates exponentiation. (The hash value of the empty string is zero.)

1. **boolean isEmpty():** Returns true if, and only if, [length()](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#length--) is 0.
2. **int indexOf(int ch):** Returns the index within this string of the first occurrence of the specified character. If a character with value ch occurs in the character sequence represented by this String object, then the index (in Unicode code units) of the first such occurrence is returned. If no such character occurs in this string, then -1 is returned.

System.out.println("hima".indexOf(97)); // returns index of ‘a’ = 3

1. **int indexOf(int ch, int fromIndex):** Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
2. [**int indexOf (String s)**](https://www.geeksforgeeks.org/java-string-indexof/)**:** Returns the index within the string of the first occurrence of the specified string.

String s = ”Learn Share Learn”;

int output = s.indexOf(“Share”); // returns 6

1. [**int indexOf (String s, int i)**](https://www.geeksforgeeks.org/java-string-indexof/)**:** Returns the index within the string of the first occurrence of the specified string, starting at the specified index.

String s = ”Learn Share Learn”;

int output = s.indexOf("ea",3); // returns 13

1. **int lastIndexOf(int ch):** Returns the index within this string of the last occurrence of the specified character.
2. [**Int lastIndexOf( String s)**](https://www.geeksforgeeks.org/java-lang-string-lastindexof-method/)**:** Returns the index within the string of the last occurrence of the specified string.

String s = ”Learn Share Learn”;

int output = s.lastIndexOf("a"); // returns 14

1. **int lastIndexOf(**[**String**](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html)**str, int fromIndex):** Returns the index within this string of the last occurrence of the specified substring, searching backward starting at the specified index.

System.out.println("hamanata".lastIndexOf("a",4)); //returns 3

1. **boolean matches(**[**String**](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html)**regex):** Tells whether or not this string matches the given [regular expression](https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html#sum). An invocation of this method of the form str.matches(regex) yields exactly the same result as the expression

[Pattern](https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html).[matches(regex, str)](https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html#matches-java.lang.String-java.lang.CharSequence-).

Example: System.out.println("hima".matches("h.\*")); //true

1. [**String replace (char oldChar, char newChar)**](https://www.geeksforgeeks.org/java-lang-string-replace-method-java/)**:** Returns new string by replacing all occurrences of oldChar with newChar.

String s1 = “feeksforfeeks“;

String s2 = s1.replace(‘f’ ,’g’); // returns “geeksgorgeeks”

Note:- s1 is still feeksforfeeks and s2 is geeksgorgeeks

1. [**String**](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html)**replace(**[**CharSequence**](https://docs.oracle.com/javase/8/docs/api/java/lang/CharSequence.html)**target,** [**CharSequence**](https://docs.oracle.com/javase/8/docs/api/java/lang/CharSequence.html)**replacement):** Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence. The replacement proceeds from the beginning of the string to the end, for example, replacing "aa" with "b" in the string "aaa" will result in "ba" rather than "ab".
2. [**String**](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html)**replaceFirst(**[**String**](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html)**regex,** [**String**](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html)**replacement):** Replaces the first substring of this string that matches the given [regular expression](https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html#sum) with the given replacement.
3. [**String**](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html)**[] split(**[**String**](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html)**regex):** Splits this string around matches of the given [regular expression](https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html#sum).

"boo:and:foo".split("o") //returns { "b", "", ":and:f" }

1. **boolean startsWith(**[**String**](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html)**prefix):** Tests if this string starts with the specified prefix.
2. [**String substring (int i)**](https://www.geeksforgeeks.org/substring-in-java/)**:** Return the substring from the ith  index character to end.

"GeeksforGeeks".substring(3); // returns “ksforGeeks”

1. [**String substring (int i, int j)**](https://www.geeksforgeeks.org/substring-in-java/)**:**Returns the substring from i to j-1 index.

"GeeksforGeeks".substring(2, 5); // returns “eks”

1. **char[] toCharArray():** Converts this string to a new character array.

1. [**String toLowerCase()**](https://www.geeksforgeeks.org/java-string-tolowercase-examples/)**:**Converts all the characters in the String to lower case.

String word1 = “HeLLo”;

String word3 = word1.toLowerCase(); // returns “hello"

1. [**String toUpperCase()**](https://www.geeksforgeeks.org/java-touppercase-examples/)**:** Converts all the characters in the String to upper case.

String word1 = “HeLLo”;

String word2 = word1.toUpperCase(); // returns “HELLO”

1. [**String trim()**](https://www.geeksforgeeks.org/java-string-trim-method-example/)**:** Returns the copy of the String, by removing whitespaces at both ends. It does not affect whitespaces in the middle.

String word1 = “ Learn Share Learn “;

String word2 = word1.trim(); // returns “Learn Share Learn”

1. **boolean regionMatches(int start\_OString, String another, int start\_AString, int no\_of\_char)** : It returns true if the region of original string starting with index start\_OString matches with the region of another string starting with string\_AString, and no\_of\_char refers to the number of character to be compared.
2. [**static String join(CharSequence de\_limiter, CharSequence… elements)**](https://www.geeksforgeeks.org/java-string-join-examples/) : It returns a string which contains all the elements joins by the de\_limiter.

String gfg = String.join(" < ", "Four", "Five", "Six", "Seven");

//gfg = “Four < Five < Six < Seven”

**StringBuffer class in Java**

* StringBuffer is a class in Java that represents a mutable sequence of characters, which means you can modify the contents of a string without creating a new object every time.
* The initial capacity of a StringBuffer can be specified when it is created, or it can be set later with the ensureCapacity() method.
* The append() method is used to add characters, strings, or other objects to the end of the buffer.
* The insert() method is used to insert characters, strings, or other objects at a specified position in the buffer.
* The delete() method is used to remove characters from the buffer.
* The reverse() method is used to reverse the order of the characters in the buffer.

**Advantages of StringBuffer over String:**

* StringBuffer objects are more efficient than creating new String objects each time you need to modify a string. As each modification to a String object creates a new object and discards the old one.
* StringBuffer is a peer class of String that provides much of the functionality of strings. The string represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences.
* StringBuffer may have characters and substrings inserted in the middle or appended to the end.

**Important Constructors of StringBuffer class**

1. **StringBuffer():** creates an empty string buffer with an initial capacity of 16.
2. **StringBuffer(String str):** creates a string buffer with the specified string.
3. **StringBuffer(int capacity):** creates an empty string buffer with the specified capacity as length.

Example:

public class StringBufferExample {

public static void main(String[] args)

{

StringBuffer sb = new StringBuffer();

System.out.println(sb.capacity()); // default capacity = 16

sb.append("Hello"); // sb = “Hello”

sb.append(false); // sb = “Hellofalse”

sb.insert(1, "Java"); // sb = “HJavaellofalse”

sb.repeat('z', 5); // sb = “HJavaellofalsezzzzz”

sb.replace(1, 5, "$&@"); // sb = “H$&@ellofalsezzzzz”

sb.delete(1, 4); // sb = “Hellofalsezzzzz”

System.out.println(sb.reverse()); //sb = “zzzzzeslafolleH”

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

// Now new capacity= (16\*2)+2=34 i.e (oldcapacity\*2)+2

}

}

Note: StringBuffer extends all methods of String class I.e. length(), chatAt() etc.

**StringBuilder Class in Java**

* The function of StringBuilder is very much similar to the StringBuffer class, as both of them provide an alternative to String Class by making a mutable sequence of characters. However, the StringBuilder class differs from the StringBuffer class on the basis of synchronization.
* The StringBuilder class provides no guarantee of synchronization whereas the StringBuffer class does.
* String Builder is not thread-safe and high in performance compared to String buffer which is thread-safe.

**Syntax:**

public final class StringBuilder

extends Object

implements Serializable, CharSequence

**Initialization:**

StringBuilder sb = new StringBuilder();

sb.append(“Hello”);

System.out.println(sb);

Note: It has all constructors and methods as StringBuffer class has.

**Some Methods of StringBuilder/StringBuffer**

1. [**StringBuilder append(X x)**](https://www.geeksforgeeks.org/stringbuilder-append-method-in-java-with-examples/)**:** This method appends the string representation of the X type argument to the sequence.
2. [**StringBuilder appendCodePoint(int codePoint)**](https://www.geeksforgeeks.org/stringbuilder-appendcodepoint-method-in-java-with-examples/)**:** This method appends the string representation of the codePoint argument to this sequence.
3. [**int capacity()**](https://www.geeksforgeeks.org/stringbuilder-capacity-in-java-with-examples/)**:** This method returns the current capacity.
4. [**char charAt(int index)**](https://www.geeksforgeeks.org/stringbuilder-charat-in-java-with-examples/)**:** This method returns the char value in this sequence at the specified index.
5. **IntStream chars():** This method returns a stream of int zero-extending the char values from this sequence.
6. [**int codePointAt(int index)**](https://www.geeksforgeeks.org/stringbuilder-codepointat-in-java-with-examples/)**:** This method returns the character (Unicode code point) at the specified index.
7. [**int codePointBefore(int index)**](https://www.geeksforgeeks.org/stringbuilder-codepointbefore-in-java-with-examples/)**:** This method returns the character (Unicode code point) before the specified index.
8. [**int codePointCount(int beginIndex, int endIndex)**](https://www.geeksforgeeks.org/stringbuilder-codepointcount-in-java-with-examples/)**:** This method returns the number of Unicode code points in the specified text range of this sequence.
9. **IntStream codePoints():** This method returns a stream of code point values from this sequence.
10. [**StringBuilder delete(int start, int end)**](https://www.geeksforgeeks.org/stringbuilder-delete-in-java-with-examples/)**:** This method removes the characters in a substring of this sequence.
11. **StringBuilder deleteCharAt(int index):** This method removes the char at the specified position in this sequence.
12. [**void ensureCapacity(int minimumCapacity)**](https://www.geeksforgeeks.org/stringbuilder-ensurecapacity-in-java-with-examples/)**:** This method ensures that the capacity is at least equal to the specified minimum.
13. [**void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)**](https://www.geeksforgeeks.org/stringbuilder-getchars-in-java-with-examples/)**:** This method characters are copied from this sequence into the destination character array dst.
14. [**int indexOf()**](https://www.geeksforgeeks.org/stringbuilder-indexof-method-in-java-with-examples/)**:** This method returns the index within this string of the first occurrence of the specified substring.
15. **StringBuilder insert(int offset, boolean b):** This method inserts the string representation of the boolean alternate argument into this sequence.
16. **StringBuilder insert():** This method inserts the string representation of the char argument into this sequence.
17. [**int lastIndexOf()**](https://www.geeksforgeeks.org/stringbuilder-lastindexof-method-in-java-with-examples/)**:** This method returns the index within this string of the last occurrence of the specified substring.
18. [**int length()**](https://www.geeksforgeeks.org/stringbuilder-length-in-java-with-examples/)**:** This method returns the length (character count).
19. **int offsetByCodePoints(int index, int codePointOffset):** This method returns the index within this sequence that is offset from the given index by codePointOffset code points.
20. [**StringBuilder replace(int start, int end, String str)**](https://www.geeksforgeeks.org/stringbuilder-replace-in-java-with-examples/)**:** This method replaces the characters in a substring of this sequence with characters in the specified String.
21. [**StringBuilder reverse()**](https://www.geeksforgeeks.org/stringbuilder-reverse-in-java-with-examples/)**:** This method causes this character sequence to be replaced by the reverse of the sequence.
22. [**void setCharAt(int index, char ch)**](https://www.geeksforgeeks.org/stringbuilder-setcharat-in-java-with-examples/)**:** In this method, the character at the specified index is set to ch.
23. [**void setLength(int newLength)**](https://www.geeksforgeeks.org/stringbuilder-setlength-in-java-with-examples/)**:** This method sets the length of the character sequence.
24. [**CharSequence subSequence(int start, int end)**](https://www.geeksforgeeks.org/stringbuilder-subsequence-in-java-with-examples/)**:** This method returns a new character sequence that is a subsequence of this sequence.
25. [**String substring()**](https://www.geeksforgeeks.org/stringbuilder-substring-method-in-java-with-examples/)**:** This method returns a new String that contains a subsequence of characters currently contained in this character sequence.
26. [**String toString()**](https://www.geeksforgeeks.org/stringbuilder-tostring-method-in-java-with-examples/)**:** This method returns a string representing the data in this sequence.
27. [**void trimToSize()**](https://www.geeksforgeeks.org/stringbuilder-trimtosize-method-in-java-with-examples/)**:** This method attempts to reduce storage used for the character sequence.

**When to use String class, StringBuffer and StringBuilder class:**

* If a string is going to remain constant throughout the program, then use the String class object because a String object is immutable.
* If a string can change (for example: lots of logic and operations in the construction of the string) and will only be accessed from a single thread, using a StringBuilder is good enough.
* If a string can change and will be accessed from multiple threads, use a StringBuffer because StringBuffer is synchronous, so you have thread-safety.
* If you don’t want thread-safety than you can also go with StringBuilder class as it is not synchronized.

**StringTokenizer Class in Java**

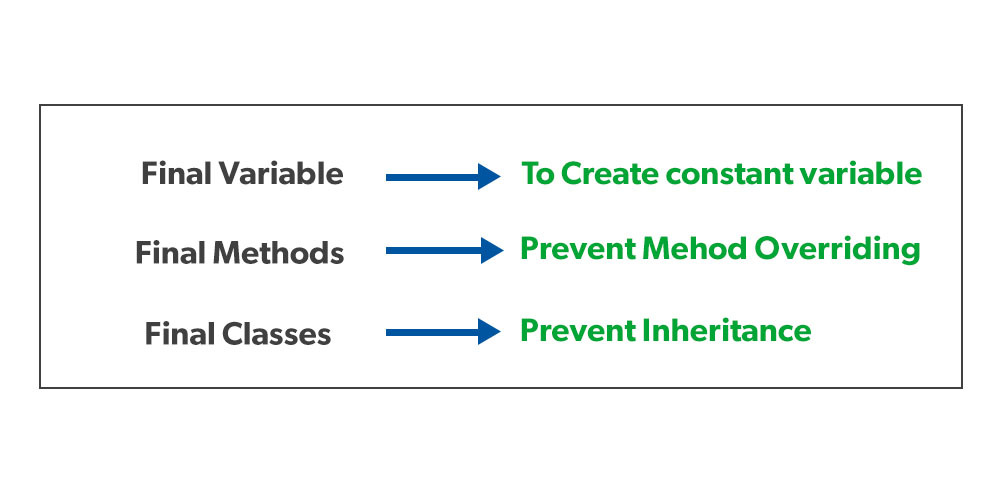
* It is used to break a string into tokens.
* It implements the Enumeration interface. This class is used for parsing data.
* To use the String Tokenizer class we have to specify an input string and a string that contains delimiters.
* Delimiters are the characters that separate tokens. Each character in the delimiter string is considered a valid delimiter.

| **Constructor** | **Description** |
| --- | --- |
| **StringTokenizer(String str)** | Default delimiters like newline, space, tab, carriage return, and form feed. |
| **StringTokenizer(String str, String delim)** | delim is a set of delimiters that are used to tokenize the given string. |
| **StringTokenizer(String str, String delim, boolean flag)** | The first two parameters have the same meaning wherein If the flag is true, delimiter characters are considered to be tokens. |

| **Method** | **Action Performed** |
| --- | --- |
| [**countTokens()**](https://www.geeksforgeeks.org/stringtokenizer-counttokens-method-in-java-with-examples/) | Returns the total number of tokens present |
| [**hasMoreToken()**](https://www.geeksforgeeks.org/stringtokenizer-hasmoretokens-method-in-java-with-examples/) | Tests if tokens are present for the StringTokenizer’s string |
| [**nextElement()**](https://www.geeksforgeeks.org/stringtokenizer-nextelement-method-in-java-with-examples/#:~:text=The nextElement() method of,Object rather than the String.) | Returns an Object rather than String |
| [**hasMoreElements()**](https://www.geeksforgeeks.org/stringtokenizer-hasmoreelements-method-in-java-with-examples/) | Returns the same value as hasMoreToken() |
| [**nextToken()**](https://www.geeksforgeeks.org/stringtokenizer-nexttoken-method-in-java-with-examples/) | Returns the next token from the given StringTokenizer. |

**final Keyword in Java:**

final is a [non-access modifier](https://www.geeksforgeeks.org/access-and-non-access-modifiers-in-java/) applicable only to a variable, a method, or a class. The following are different contexts where final is used.



1. **Final Variables:**

When a variable is declared with the final keyword, its value can't be modified, essentially, a constant. This also means that you must initialize a final variable.

If the final variable is a reference, this means that the variable cannot be re-bound to reference another object, but the internal state of the object pointed by that reference variable can be changed i.e. you can add or remove elements from the [final array](https://www.geeksforgeeks.org/final-arrays-in-java/) or final collection.

It is good practice to represent final variables in all uppercase, using underscore to separate words.

**Note**:

* Note the difference between C++ const variables and Java final variables. const variables in C++ must be assigned a value when declared. For final variables in Java, it is not necessary. A final variable can be assigned value later, but only once.
* final with [foreach loop](https://www.geeksforgeeks.org/for-each-loop-in-java/): final with for-each statement is a legal statement.

1. **Final Methods**

When a method is declared with final keyword, it is called a final method. A final method cannot be [overridden](https://www.geeksforgeeks.org/overriding-in-java/). We must declare methods with the final keyword for which we are required to follow the same implementation throughout all the derived classes.

1. **Final classes**

When a class is declared with final keyword, it is called a final class. A final class cannot be extended(inherited).

Another use of final with classes is to [create an immutable class](https://www.geeksforgeeks.org/create-immutable-class-java/) like the predefined  [String](https://www.geeksforgeeks.org/string-class-in-java/) class. One can not make a class immutable without making it final.

If a class is declared as final as by default all of the methods present in that class are automatically final, but variables are not.

**Java.lang.Math Class in Java**

Math Class methods helps to perform the numeric operations like square, square root, cube, cube root, exponential and trigonometric operations

Declaration :

public final class Math extends Object

**What is NaN argument?**A constant holding a Not-a-Number (NaN) value of type double. It is equivalent to the value returned by Double.longBitsToDouble(0x7ff8000000000000L).

**Methods of Math class :**

1. **static datatype abs(datatype arg):** returns the absolute value(non-negative) of any type of argument passed. This method can handle all the data types.
2. **static double floor(double arg):** returns the closest integer value which is either less or equal to the passed argument.  
   eg : 101.23 has floor value = 101
3. **static datatype round(datatype arg):** returns the round off value of the passed argument upto closest decimal places.
4. **static double ceil(double arg):** returns the closest possible integer value which is either greater or equal to the argument passed.
5. **static double IEEEremainder(double d1,double d2):** returns the remainder value when d1(dividend) is divided by d2(divisor).

Remainder value = d1 - d2 \* n, where, n = closest exact value of d1/d2

1. **static double pow(double a, double b):** returns ab (a  raise to the power of b).
2. **static double sqrt(double a):** returns the positive square root value of the argument passed to it.
3. **static double cbrt(double arg)**: returns the cube root of the passed argument.
4. **static double toRadians(double deg):** returns radians equivalent of the degree-argument passed.
5. **static double asin(double arg):** returns the arc sine value of the method argument passed. Returned angle is in the range -pi/2 to pi/2.   
   arc sine is inverse sine of the argument passed. i.e. **asin(arg) = sine-1 of arg**

Result is NaN,if the argument is NaN or its absolute value is greater than 1.

1. **static double acos(double a):**  returns the arc cosine value of the passed argument.
2. static double atan(double a): returns the arc tangent of the method argument value. The returned angle is in the range -pi/2 through pi/2.
3. **static double signum(double x):** returns the signum value of the argument passed.

-1 if x < 0

signum fun(x) = 0 if x = 0

1 if x > 0

1. **static double max(double v1, double v2):** returns v1 or v2 based on which number is greater. It can return either of the two if v1 = v2.

This method just compares using magnitude without considering any sign.

1. **static double log(double arg):** returns the base10 logarithmic value of the passed argument.
2. **static double exp(double arg):** returns the Euler’s number raised to the power of double argument.   
   Important cases:

* Result is NaN, if argument is NaN.
* Result is +ve infinity, if the argument is +ve infinity.
* Result is +ve zero, if argument is -ve infinity.

1. **static double cosh(double arg): r**eturns the hyperbolic cosine of the argument passed.   
   Special cases :

* Result is NaN, if argument is NaN.
* Result is 1.0, if the argument is zero.
* Result is +ve infinity if the argument is infinite.

**BigInteger Class in Java**

BigInteger class (java.math.BigInteger) is used for the mathematical operation which involves very big integer calculations that are outside the limit of all available primitive data types.

**Declaration**:

BigInteger A;

**Initialization**:

A = BigInteger.valueOf(54);

Or, A = new BigInteger(“54”);

Some constants are also defined in BigInteger class for ease of initialization as follows:

A = BigInteger.ONE;

// Other than this, available constant are BigInteger.ZERO and BigInteger.TEN

**Methods of BigInteger Class**

| **Method** | **Action Performed** |
| --- | --- |
| 1. **add(BigInteger val)** | Returns a BigInteger whose value is (this + val). |
| 1. [**abs()**](https://www.geeksforgeeks.org/biginteger-abs-method-in-java/) | Returns a BigInteger whose value is the absolute value of this BigInteger. |
| 1. [**and(BigInteger val)**](https://www.geeksforgeeks.org/biginteger-and-method-in-java/) | Returns a BigInteger whose value is (this & val). |
| 1. [**andNot(BigInteger val**](https://www.geeksforgeeks.org/biginteger-andnot-method-in-java/)**)** | Returns a BigInteger whose value is (this & ~val). |
| 1. [**bitCount()**](https://www.geeksforgeeks.org/biginteger-bitcount-method-in-java/) | Returns the number of bits in the two’s complement representation of this BigInteger that differ from its sign bit. |
| 1. [bitLength()](https://www.geeksforgeeks.org/biginteger-bitlength-method-in-java/) | Returns the number of bits in the minimal two’s-complement representation of this BigInteger, excluding a sign bit. |
| 1. [byteValueExact()](https://www.geeksforgeeks.org/java-8-biginteger-bytevalueexact-method-with-examples/) | Converts this BigInteger to a byte, checking for lost information. |
| 1. [clearBit(int n)](https://www.geeksforgeeks.org/biginteger-clearbit-method-in-java/) | Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit cleared. |
| 1. [**compareTo(BigInteger val)**](https://www.geeksforgeeks.org/biginteger-compareto-method-in-java/) | Compares this BigInteger with the specified BigInteger. |
| 1. **divide(BigInteger val)** | Returns a BigInteger whose value is (this / val). |
| 1. [**divideAndRemainder(BigInteger val)**](https://www.geeksforgeeks.org/java-8-biginteger-divideandremainder-method-with-examples/) | Returns an array of two BigIntegers containing (this / val) followed by (this % val). |
| 1. [**doubleValue()**](https://www.geeksforgeeks.org/biginteger-doublevalue-method-in-java/) | Converts this BigInteger to a double. |
| 1. [**equals(Object x)**](https://www.geeksforgeeks.org/biginteger-equals-method-in-java/) | Compares this BigInteger with the specified Object for equality. |
| 1. [flipBit(int n)](https://www.geeksforgeeks.org/biginteger-flipbit-method-in-java/) | Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit flipped. |
| 1. [**floatValue()**](https://www.geeksforgeeks.org/biginteger-floatvalue-method-in-java/) | Converts this BigInteger to a float. |
| 1. **gcd(BigInteger val)** | Returns a BigInteger whose value is the greatest common divisor of abs(this) and abs(val). |
| 1. [getLowestSetBit()](https://www.geeksforgeeks.org/biginteger-getlowestsetbit-method-in-java/) | Returns the index of the rightmost (lowest-order) one bit in this BigInteger (the number of zero bits to the right of the rightmost one bit). |
| 1. hashCode() | Returns the hash code for this BigInteger. |
| 1. [**intValue()**](https://www.geeksforgeeks.org/biginteger-intvalue-method-in-java/) | Converts this BigInteger to an int. |
| 1. [intValueExact()](https://www.geeksforgeeks.org/biginteger-intvalueexact-method-in-java-with-examples/) | Converts this BigInteger to an int, checking for lost information. |
| 1. **isProbablePrime(int certainty)** | Returns true if this BigInteger is probably prime, false if it’s definitely composite.(certainty > 1) |
| 1. [**longValue()**](https://www.geeksforgeeks.org/biginteger-longvalue-method-in-java/) | Converts this BigInteger to a long. |
| 1. [longValueExact()](https://www.geeksforgeeks.org/java-8-biginteger-longvalueexact-method-with-examples/) | Converts this BigInteger to a long, checking for lost information. |
| 1. [**max(BigInteger val)**](https://www.geeksforgeeks.org/biginteger-max-and-min-methods-in-java/) | Returns the maximum of this BigInteger and val. |
| 1. [**min(BigInteger val**](https://www.geeksforgeeks.org/biginteger-max-and-min-methods-in-java/)**)** | Returns the minimum of this BigInteger and val. |
| 1. [**mod(BigInteger m**](https://www.geeksforgeeks.org/biginteger-mod-method-in-java/) | Returns a BigInteger whose value is (this mod m). |
| 1. [modInverse(BigInteger m)](https://www.geeksforgeeks.org/java-math-biginteger-modinverse-method-in-java/) | Returns a BigInteger whose value is (this-1 mod m). |
| 1. [**modPow(BigInteger exponent, BigInteger m**](https://www.geeksforgeeks.org/biginteger-modpow-method-in-java/) | Returns a BigInteger whose value is (this exponent mod m). |
| 1. **multiply(BigInteger val)** | Returns a BigInteger whose value is (this \* val). |
| 1. [**negate()**](https://www.geeksforgeeks.org/biginteger-negate-method-in-java/) | Returns a BigInteger whose value is (-this). |
| 1. **nextProbablePrime()** | Returns the first integer greater than this BigInteger that is probably prime. |
| 1. [**not()**](https://www.geeksforgeeks.org/biginteger-not-method-in-java/) | Returns a BigInteger whose value is (~this). |
| 1. [**or(BigInteger val)**](https://www.geeksforgeeks.org/biginteger-or-method-in-java/) | Returns a BigInteger whose value is (this | val). |
| 1. [**pow(int exponent)**](https://www.geeksforgeeks.org/biginteger-pow-method-in-java/) | Returns a BigInteger whose value is (thisexponent). |
| 1. [**probablePrime(int bitLength, Random rnd)**](https://www.geeksforgeeks.org/java-math-biginteger-probableprime-method-in-java/) | Returns a positive BigInteger that is probably prime, with the specified bitLength. |
| 1. [**remainder(BigInteger val)**](https://www.geeksforgeeks.org/biginteger-remainder-method-in-java/) | Returns a BigInteger whose value is (this % val). |
| 1. [**setBit(int n)**](https://www.geeksforgeeks.org/biginteger-setbit-method-in-java/) | Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit set. |
| 1. [**shiftLeft(int n)**](https://www.geeksforgeeks.org/biginteger-shiftleft-method-in-java/) | Returns a BigInteger whose value is (this << n). |
| 1. [**shiftRight(int n)**](https://www.geeksforgeeks.org/biginteger-shiftright-method-in-java/) | Returns a BigInteger whose value is (this >> n). |
| 1. [shortValueExact()](https://www.geeksforgeeks.org/java-8-biginteger-shortvalueexact-method-with-examples/) | Converts this BigInteger to a short, checking for lost information. |
| 1. [**signum()**](https://www.geeksforgeeks.org/biginteger-signum-method-in-java/) | Returns the signum function of this BigInteger. |
| 1. [**sqrt()**](https://www.geeksforgeeks.org/biginteger-sqrt-method-in-java/) | Returns the integer square root of this BigInteger. |
| 1. **sqrtAndRemainder()** | Returns an array of two BigIntegers containing the integer square root s of this and its remainder this – s\*s, respectively. |
| 1. **subtract(BigInteger val)** | Returns a BigInteger whose value is (this – val). |
| 1. [testBit(int n)](https://www.geeksforgeeks.org/biginteger-testbit-method-in-java/) | Returns true if and only if the designated bit is set. |
| 1. [**toByteArray()**](https://www.geeksforgeeks.org/biginteger-tobytearray-method-in-java/) | Returns a byte array containing the two’s-complement representation of this BigInteger. |
| 1. [**toString()**](https://www.geeksforgeeks.org/biginteger-tostring-method-in-java/) | Returns the decimal String representation of this BigInteger. |
| 1. [**toString(int radix)**](https://www.geeksforgeeks.org/biginteger-tostring-method-in-java/) | Returns the string representation of this BigInteger in the given radix. |
| 1. [**valueOf(long val)**](https://www.geeksforgeeks.org/biginteger-valueof-method-in-java/) | Returns a BigInteger whose value is equal to that of the specified long. |
| 1. [**xor(BigInteger val)**](https://www.geeksforgeeks.org/biginteger-xor-method-in-java/) | Returns a BigInteger whose value is (this ^ val). |

**Note**: BigInteger class internally uses an array of integers for processing, the operation on an object of BigIntegers are not as fast as on primitives. Like, add function on BigIntgers doesn’t take the constant time it takes time proportional to the length of BigInteger, so the complexity of program will change accordingly.

**BigDecimal Class in Java**

The BigDecimal class provides operations on double numbers for arithmetic, scale handling, rounding, comparison, format conversion and hashing. It can handle very large and very small floating point numbers with great precision but compensating with the time complexity a bit.

**Input :** double a=0.03;  
 double b=0.04;  
 double c=b-a;  
 System.out.println(c);  
**Output :**0.009999999999999998  
  
**Input :** BigDecimal \_a = new BigDecimal("0.03");  
 BigDecimal \_b = new BigDecimal("0.04");  
 BigDecimal \_c = \_b.subtract(\_a);  
 System.out.println(\_c);  
**Output :**0.01

**Need Of BigDecimal**

* The two java primitive types(double and float) are floating point numbers, which is stored as a binary representation of a fraction and a exponent.
* Other primitive types(except boolean) are fixed-point numbers. Unlike fixed point numbers, floating point numbers will most of the times return an answer with a small error (around 10^-19) This is the reason why we end up with 0.009999999999999998 as the result of 0.04-0.03 in the above example.

But BigDecimal provides us with the exact answer.

**Declaration**:

BigDecimal A

**Initialization**:

A = BigDecimal.valueOf(5.4);

Or , A = new BigDecimal(“5.4”);

**Mathematical operations:**

int c = a + b;  
 BigDecimal C = A.add(B);   
Other similar function are subtract() , multiply(), divide(), pow().

But all these functions, except pow() which takes integer as its argument, take BigDecimal as their argument.

**Extraction of value from BigDecimal:**

// value should be in limit of double x  
double x = A.doubleValue();   
// To get string representation of BigDecimal A  
String z = A.toString();

Note: BigDecimal has all other methods as BigInteger has.

**Collections in Java**

* Any group of individual objects that are represented as a single unit is known as a Java Collection of Objects.
* In Java, a separate framework named the “Collection Framework” has been defined in JDK 1.2 which holds all the Java Collection Classes and Interface in it.
* In Java, the Collection interface (java.util.Collection) and Map interface (java.util.Map) are the two main “root” interfaces of Java collection classes.

**Note:-** A framework is a set of classes and interfaces which provide a ready-made architecture.

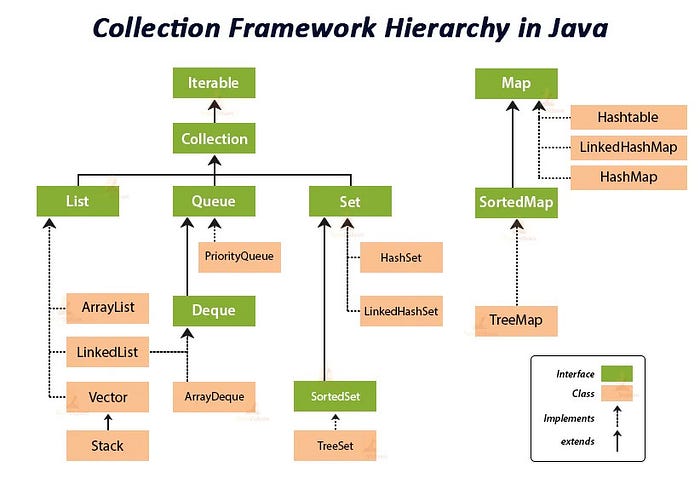
**Advantages of the Java Collection Framework:**

1. **Consistent API:** The API has a basic set of interfaces like Collection, Set, List, or Map, all the classes (ArrayList, LinkedList, Vector, etc) that implement these interfaces have some common set of methods.
2. **Reduces programming effort:** A programmer doesn’t have to worry about the design of the Collection but rather he can focus on its best use in his program.

1. **Increases program speed and quality:** Increases performance by providing high-performance implementations of useful data structures and algorithms.

**Hierarchy of the Collection Framework in Java:**

* The utility package, (**java.util**) contains all the classes and interfaces that are required by the collection framework.
* The collection framework contains an interface named an **iterable interface** which provides the iterator to iterate through all the collections. This interface is extended by the main collection interface which acts as **a root for the collection framework.**
* All the collections extend this collection interface thereby extending the properties of the iterator and the methods of this interface.



**Methods of the Collection Interface:**

| Method | Description |
| --- | --- |
| 1. [add(Object)](https://www.geeksforgeeks.org/collection-add-method-in-java-with-examples/) | add an object to the collection. |
| 1. [addAll(Collection c)](https://www.geeksforgeeks.org/collections-addall-method-in-java-with-examples/) | adds all the elements in the given collection to this collection. |
| 1. [clear()](https://www.geeksforgeeks.org/collection-clear-method-in-java-with-examples/) | removes all of the elements from this collection. |
| 1. [contains(Object o)](https://www.geeksforgeeks.org/collection-contains-method-in-java-with-examples/) | returns true if the collection contains the specified element. |
| 1. [containsAll(Collection c)](https://www.geeksforgeeks.org/java-collection-containsall-method/) | returns true if the collection contains all of the elements in the given collection. |
| 1. [equals(Object o)](https://www.geeksforgeeks.org/equals-hashcode-methods-java/) | compares the specified object with this collection for equality. |
| 1. [hashCode()](https://www.geeksforgeeks.org/equals-hashcode-methods-java/) | used to return the hash code value for this collection. |
| 1. [isEmpty()](https://www.geeksforgeeks.org/collection-isempty-method-in-java-with-examples/) | returns true if this collection contains no elements. |
| 1. [iterator()](https://www.geeksforgeeks.org/iterators-in-java/) | returns an iterator over the elements in this collection. |
| 1. [max()](https://www.geeksforgeeks.org/collections-max-method-in-java-with-examples/) | return the maximum value present in the collection. |
| 1. [parallelStream()](https://www.geeksforgeeks.org/what-is-java-parallel-streams/) | returns a parallel Stream with this collection as its source. |
| 1. [remove(Object o)](https://www.geeksforgeeks.org/java-program-to-remove-a-specific-element-from-a-collection/) | remove the given object from the collection. If there are duplicate values, then this method removes the first occurrence of the object. |
| 1. [removeAll(Collection c)](https://www.geeksforgeeks.org/java-collection-removeall-method/) | remove all the objects mentioned in the given collection from the collection. |
| 1. [removeIf(Predicate filter)](https://www.geeksforgeeks.org/java-collection-removeif-method/) | remove all the elements of this collection that satisfy the given [predicate](https://www.geeksforgeeks.org/mathematic-logic-predicates-quantifiers/). |
| 1. [retainAll(Collection c)](https://www.geeksforgeeks.org/java-collection-retainall-method/) | retain only the elements in this collection that are contained in the specified collection. |
| 1. [size()](https://www.geeksforgeeks.org/how-to-get-a-size-of-collection-in-java/) | return the number of elements in the collection. |
| 1. [spliterator()](https://www.geeksforgeeks.org/java-collection-spliterator-with-examples/) | create a [Spliterator](https://www.geeksforgeeks.org/java-program-to-convert-iterator-to-spliterator/) over the elements in this collection. |
| 1. [stream()](https://www.geeksforgeeks.org/stream-in-java/) | return a sequential Stream with this collection as its source. |
| 1. [toArray()](https://www.geeksforgeeks.org/arraylist-toarray-method-in-java-with-examples/) | return an array containing all of the elements in this collection. |

**Collections Class in Java:**

* Collections class in Java is one of the utility classes in Java Collections Framework. The java.util package contains the Collections class in Java.
* Java Collections class is used with the [static methods](https://www.geeksforgeeks.org/static-methods-vs-instance-methods-java/) that operate on the collections or return the collection.
* All the methods of this class throw the NullPointerException if the collection or object passed to the methods is null.

**Fields:**

The collection class contains 3 fields as listed below:

* EMPTY\_LIST to get an immutable empty List
* EMPTY\_SET to get an immutable empty Set
* EMPTY\_MAP to get an immutable empty Map

**Methods:**

| Methods | Description |
| --- | --- |
| addAll(Collection< T> c, T… elements) | It is used to insert the specified collection elements to the specified collection. |
| [asLifoQueue(Deque<T> deque)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-aslifoqueue-method-in-java-with-examples/&sa=U&ved=2ahUKEwjR2pzb-NnsAhXB63MBHfsmCzQ4HhAWMAR6BAgEEAI&usg=AOvVaw0cqoFANseci3YfUQzT91qJ) | This method returns a view of a Deque as a Last-in-first-out (Lifo) Queue. |
| [binarySearch(List<T> list, T key)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-binarysearch-java-examples/&sa=U&ved=2ahUKEwi6uP_199nsAhXj73MBHZbDDPMQFjAJegQIBRAC&usg=AOvVaw3Z6NwOyP0de-c-hRasRUsf) | Returns index of key in sorted list sorted in  ascending order. |
| [binarySearch(List<T> list, T key, Comparator<T> c)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-binarysearch-java-examples/&sa=U&ved=2ahUKEwi6uP_199nsAhXj73MBHZbDDPMQFjAJegQIBRAC&usg=AOvVaw3Z6NwOyP0de-c-hRasRUsf) | Returns index of key in sorted list sorted in  order defined by Comparator c. |
| [checkedCollection(Collection<E> c, Class<E> type)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-checkedcollection-method-in-java-with-examples/&sa=U&ved=2ahUKEwjR2pzb-NnsAhXB63MBHfsmCzQ4HhAWMAJ6BAgJEAI&usg=AOvVaw1gjbCCS3zLq1m9WpQf0BAH) | This method returns a dynamically typesafe view of the specified collection. |
| checkedList(List<E> list, Class<E> type) | This method returns a dynamically typesafe view of the specified list. |
| checkedMap(Map<K,​V> m, Class<K> keyType, Class<V> valueType) | This method returns a dynamically typesafe view of the specified map. |
| checkedNavigableMap(NavigableMap<K,​V> m, Class<K> keyType, Class<V> valueType) | This method returns a dynamically typesafe view of the specified navigable map. |
| checkedNavigableSet(NavigableSet<E> s, Class<E> type) | This method returns a dynamically typesafe view of the specified navigable set. |
| checkedQueue(Queue<E> queue, Class<E> type) | This method returns a dynamically typesafe view of the specified queue. |
| checkedSet(Set<E> s, Class<E> type) | This method returns a dynamically typesafe view of the specified set. |
| checkedSortedMap(SortedMap<K,​V> m, Class<K> keyType, Class<V> valueType) | This method returns a dynamically typesafe view of the specified sorted map. |
| checkedSortedSet(SortedSet<E> s, Class<E> type) | This method returns a dynamically typesafe view of the specified sorted set. |
| [copy(List<T> dest, List<T> src)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-copy-method-in-java-with-examples/&sa=U&ved=2ahUKEwjJkK-8-NnsAhWe7HMBHWaOAiA4FBAWMAF6BAgHEAI&usg=AOvVaw0dFOeP7RLXpc_ORg_6Wj2U) | copies all of the elements from one list into another. This method throws **IndexOutOfBoundsException**, if the destination list is too small to contain the entire source List. |
| [disjoint(Collection<T> c1, Collection<T> c2)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/java-util-collections-disjoint-method-java-examples/&sa=U&ved=2ahUKEwjKuPOa-NnsAhWF6XMBHV52Bxg4ChAWMAF6BAgIEAI&usg=AOvVaw3c8uSUuGg43wQFmZCF_Bj4) | This method returns true if the two specified collections have no elements in common. |
| emptyEnumeration() | returns an enumeration that has no elements. |
| emptyIterator() | This method returns an iterator that has no elements. |
| emptyList() | This method returns an empty list (immutable). |
| emptyListIterator() | This method returns a list iterator that has no elements. |
| emptyMap() | This method returns an empty map (immutable). |
| emptyNavigableMap() | returns an empty navigable map (immutable). |
| emptyNavigableSet() | returns an empty navigable set (immutable). |
| emptySet() | This method returns an empty set (immutable). |
| emptySortedMap() | returns an empty sorted map (immutable). |
| emptySortedSet() | returns an empty sorted set (immutable). |
| [enumeration(Collection<T> c)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-enumeration-method-in-java-with-examples/&sa=U&ved=2ahUKEwjKuPOa-NnsAhWF6XMBHV52Bxg4ChAWMAl6BAgAEAI&usg=AOvVaw2zQdzpk5s0qYfnqwiTKmSC) | returns an enumeration over the specified collection. |
| [fill(List<T> list, T obj)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-fill-method-in-java-with-examples/&sa=U&ved=2ahUKEwi2_5_3-NnsAhWCheYKHZBvD4E4MhAWMAZ6BAgGEAI&usg=AOvVaw1xqIqDpotyXSl0SNNWLwsG) | replaces all of the elements of the specified list with the specified element. |
| [frequency(Collection<T> c, T obj)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/java-util-collections-frequency-java/&sa=U&ved=2ahUKEwjJkK-8-NnsAhWe7HMBHWaOAiA4FBAWMAR6BAgFEAI&usg=AOvVaw0Myrb9MDi7AANHD6aNLTCl) | returns the number of elements in the specified collection equal to the specified object. |
| indexOfSubList(List<T> source, List<T> target) | returns the starting position of the first occurrence of the specified target list within the specified source list, or -1 if there is no such occurrence. |
| lastIndexOfSubList(List<T> source, List<T> target) | returns the starting position of the last occurrence of the specified target list within the specified source list, or -1 if there is no such occurrence. |
| [list(Enumeration<T> e)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-list-method-in-java-with-examples/&sa=U&ved=2ahUKEwjKuPOa-NnsAhWF6XMBHV52Bxg4ChAWMAB6BAgDEAI&usg=AOvVaw0y-jLp0AaKB86tfyJIDziW) | This method returns an array list containing the elements returned by the specified enumeration in the order they are returned by the enumeration. |
| [max(Collection<T> coll)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-max-method-in-java-with-examples/&sa=U&ved=2ahUKEwi6uP_199nsAhXj73MBHZbDDPMQFjAEegQIAxAC&usg=AOvVaw2qA6HgjzzeCI8AHdWdeBBA) | returns the maximum element of the given collection, according to the natural ordering of its elements. |
| [max(Collection<T> coll, Comparator<T> comp)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-max-method-in-java-with-examples/&sa=U&ved=2ahUKEwi6uP_199nsAhXj73MBHZbDDPMQFjAEegQIAxAC&usg=AOvVaw2qA6HgjzzeCI8AHdWdeBBA) | This method returns the maximum element of the given collection, according to the order induced by the specified comparator. |
| [min(Collection<T> coll)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-min-method-in-java-with-examples/&sa=U&ved=2ahUKEwi6uP_199nsAhXj73MBHZbDDPMQFjAIegQIABAC&usg=AOvVaw1-7_OhEytkSHKGWgUrm2bv) | returns the minimum element of the given collection, according to the natural ordering of its elements. |
| [min(Collection<T> coll, Comparator<T> comp)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-min-method-in-java-with-examples/&sa=U&ved=2ahUKEwi6uP_199nsAhXj73MBHZbDDPMQFjAIegQIABAC&usg=AOvVaw1-7_OhEytkSHKGWgUrm2bv) | This method returns the minimum element of the given collection, according to the order induced by the specified comparator. |
| nCopies(int n, T obj) | This method returns an immutable list consisting of n copies of the specified object. |
| newSetFromMap(Map<E, Boolean> map) | returns a set backed by the specified map. |
| replaceAll(List<T> list, T oldVal, T newVal) | This method replaces all occurrences of one specified value in a list with another. |
| [reverse(List<?> list)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-reverse-java-examples/&sa=U&ved=2ahUKEwjJkK-8-NnsAhWe7HMBHWaOAiA4FBAWMAJ6BAgJEAI&usg=AOvVaw3rE6jlGifup0vkoP2GQX9W) | reverses the order of the elements in the specified list |
| reverseOrder() | returns a **comparator** that imposes the reverse of the natural ordering on a collection of objects that implement the Comparable interface. |
| [reverseOrder(Comparator<T> cmp)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-reverseorder-java-examples/&sa=U&ved=2ahUKEwjJkK-8-NnsAhWe7HMBHWaOAiA4FBAWMAB6BAgGEAE&usg=AOvVaw3WhEucy8Tqt4XilMMbuJX-) | This method returns a comparator that imposes the reverse ordering of the specified comparator. |
| [rotate(List<?> list, int distance)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/java-util-collections-rotate-method-java-examples/&sa=U&ved=2ahUKEwi6uP_199nsAhXj73MBHZbDDPMQFjAFegQIBBAC&usg=AOvVaw0Xe95qWfyXw8Yex5VrDlmu) | This method rotates the elements in the specified list by the specified distance. |
| [shuffle(List<?> list)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-shuffle-java-examples/&sa=U&ved=2ahUKEwjJkK-8-NnsAhWe7HMBHWaOAiA4FBAWMAZ6BAgDEAI&usg=AOvVaw1nrYJRpZjAKQrorR3tjl_D) | This method randomly permutes the specified list using a default source of randomness. |
| [shuffle(List<?> list, Random rand)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-shuffle-java-examples/&sa=U&ved=2ahUKEwjJkK-8-NnsAhWe7HMBHWaOAiA4FBAWMAZ6BAgDEAI&usg=AOvVaw1nrYJRpZjAKQrorR3tjl_D) | This method randomly permute the specified list using the specified source of randomness. |
| singletonMap(K key, V value) | This method returns an immutable map, mapping only the specified key to the specified value. |
| [singleton(T o)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-singleton-method-java/&sa=U&ved=2ahUKEwjR2pzb-NnsAhXB63MBHfsmCzQ4HhAWMAZ6BAgFEAI&usg=AOvVaw3_CPxwqk8yCljhuDtMnNRh) | This method returns an immutable set containing only the specified object. |
| [singletonList(T o)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-singleton-method-java/&sa=U&ved=2ahUKEwjR2pzb-NnsAhXB63MBHfsmCzQ4HhAWMAZ6BAgFEAI&usg=AOvVaw3_CPxwqk8yCljhuDtMnNRh) | This method returns an immutable list containing only the specified object. |
| [sort(List<T> list)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-sort-java-examples/&sa=U&ved=2ahUKEwi6uP_199nsAhXj73MBHZbDDPMQFjADegQIBxAC&usg=AOvVaw3LK6ysEznNr0ARFYCFDvYh) | This method sorts the specified list into ascending order, according to the natural ordering of its elements. |
| sort(List<T> list, Comparator<T> c) | This method sorts the specified list according to the order induced by the specified comparator. |
| [swap(List<?> list, int i, int j)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-swap-method-in-java-with-examples/&sa=U&ved=2ahUKEwi6uP_199nsAhXj73MBHZbDDPMQFjAGegQIBhAC&usg=AOvVaw0tePjZhO5Nvpl_md2ywFUR) | This method swaps the elements at the specified positions in the specified list. |
| [synchronizedCollection(Collection<T> c)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-synchronizedcollection-method-in-java-with-examples/&sa=U&ved=2ahUKEwjJkK-8-NnsAhWe7HMBHWaOAiA4FBAWMAh6BAgEEAI&usg=AOvVaw0Fv_A894e10Pn9caVxEgW0) | This method returns a synchronized (thread-safe) collection backed by the specified collection. |
| [synchronizedList(List<T> list)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-synchronizedlist-method-in-java-with-examples/&sa=U&ved=2ahUKEwjR2pzb-NnsAhXB63MBHfsmCzQ4HhAWMAN6BAgIEAI&usg=AOvVaw0jrt_wb0z28gyEz3mYhJQe) | This method returns a synchronized (thread-safe) list backed by the specified list. |
| synchronizedMap(Map<K, V> m) | This method returns a synchronized (thread-safe) map backed by the specified map. |
| synchronizedNavigableMap(NavigableMap<K, V> m) | This method returns a synchronized (thread-safe) navigable map backed by the specified navigable map. |
| synchronizedNavigableSet(NavigableSet<T> s) | This method returns a synchronized (thread-safe) navigable set backed by the specified navigable set. |
| synchronizedSet(Set<T> s) | This method returns a synchronized (thread-safe) set backed by the specified set. |
| synchronizedSortedMap(SortedMap<K, V> m) | This method returns a synchronized (thread-safe) sorted map backed by the specified sorted map. |
| synchronizedSortedSet(SortedSet<T> s) | This method returns a synchronized (thread-safe) sorted set backed by the specified sorted set. |
| [unmodifiableCollection(Collection<T> c)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-unmodifiablecollection-method-in-java-with-examples/&sa=U&ved=2ahUKEwjKuPOa-NnsAhWF6XMBHV52Bxg4ChAWMAN6BAgHEAI&usg=AOvVaw0unP0-Y8nc8QwwxZOSTy-6) | This method returns an unmodifiable view of the specified collection. |
| [unmodifiableList(List<T> list)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-unmodifiablelist-method-in-java-with-examples/&sa=U&ved=2ahUKEwjR2pzb-NnsAhXB63MBHfsmCzQ4HhAWMAF6BAgHEAI&usg=AOvVaw0_6E0BcWaD1Io5J6s0_27V) | This method returns an unmodifiable view of the specified list. |
| unmodifiableNavigableMap(NavigableMap<K, V> m) | This method returns an unmodifiable view of the specified navigable map. |
| unmodifiableNavigableSet(NavigableSet<T> s) | This method returns an unmodifiable view of the specified navigable set. |
| [unmodifiableSet(Set<T> s)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collections-unmodifiableset-method-in-java-with-examples/&sa=U&ved=2ahUKEwi2_5_3-NnsAhWCheYKHZBvD4E4MhAWMAl6BAgAEAI&usg=AOvVaw3MiwTquD0hZgPYmIf9vWdV) | returns an unmodifiable view of the specified set. |
| unmodifiableSortedMap(SortedMap<K, V> m) | This method returns an unmodifiable view of the specified sorted map. |
| unmodifiableSortedSet(SortedSet<T> s) | This method returns an unmodifiable view of the specified sorted set. |

**Note**: These above methods are static, so can be called without creating a object of Collections class. Like *Collections.sort(list)*

1. **List Interface in Java**

* The List interface is found in java.util package and inherits the Collection interface.
* It is an ordered collection of objects in which duplicate values can be stored.
* Since List preserves the insertion order, it allows positional access and insertion of elements.
* The implementation classes of the List interface are ArrayList, LinkedList, Stack, and Vector.
* Since List is an [interface](https://www.geeksforgeeks.org/interfaces-in-java/), objects cannot be created of the type list. We always need a class that implements this List in order to create an object.

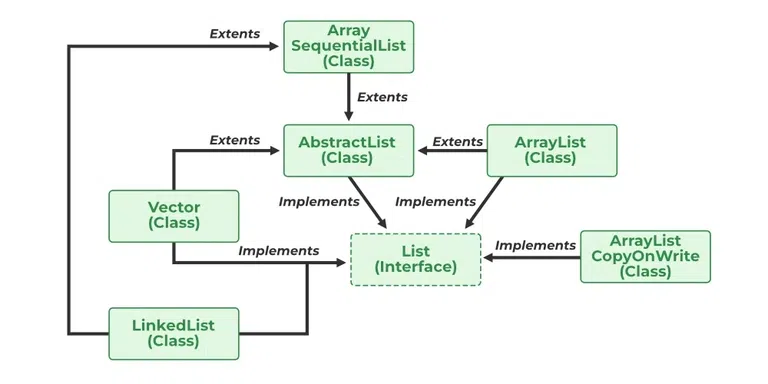
**Syntax of Java List:**

List<Obj> list = new ArrayList<Obj> ();

**Methods of the List Interface** (other than of Collections interface)**:**

| **Method** | **Description** |
| --- | --- |
| 1. [add(int index, T element)](https://www.geeksforgeeks.org/list-addint-index-e-element-method-in-java/) | This method is used with Java List Interface to add an element at a particular index in the list. When a single parameter is passed, it simply adds the element at the end of the list. |
| 1. [addAll(int index, Collection c)](https://www.geeksforgeeks.org/list-addall-method-in-java-with-examples/) | used to add all the elements in the given collection to the list. When a single parameter is passed, it adds all the elements of the given collection at the end of the list. |
| 1. [remove(int index)](https://www.geeksforgeeks.org/list-removeint-index-method-in-java-with-examples/) | removes an element from the specified index. It shifts subsequent elements(if any) to left and decreases their indexes by 1. |
| 1. [get(int index)](https://www.geeksforgeeks.org/list-get-method-in-java-with-examples/) | returns elements at the specified index. |
| 1. [set(int index, T element)](https://www.geeksforgeeks.org/arraylist-set-method-in-java-with-examples/) | replaces elements at a given index with the new element. This function returns the element which was just replaced by a new element. |
| 1. [indexOf(element)](https://www.geeksforgeeks.org/list-indexof-method-in-java-with-examples/) | returns the first occurrence of the given element or -1 if the element is not present in the list. |
| 1. [lastIndexOf(element)](https://www.geeksforgeeks.org/list-lastindexof-method-in-java-with-examples/) | returns the last occurrence of the given element or -1 if the element is not present in the list. |
| 1. [containsAll(Collection collection)](https://www.geeksforgeeks.org/list-containsall-method-in-java-with-examples/) | used to check if the list contains all the elements of collection. |
| 1. sort(Comparator comp) | sort the elements of the list on the basis of the given  [comparator](https://www.geeksforgeeks.org/comparator-interface-java/). |

**Classes Association with a Java List Interface:**



1. **AbstractList:** This class is used to implement an unmodifiable list, for which one needs to only extend this AbstractList Class and implement only the get() and the size() methods.
2. **CopyOnWriteArrayList:** This class implements the list interface. It is an enhanced version of [ArrayList](https://www.geeksforgeeks.org/arraylist-in-java/) in which all the modifications(add, set, remove, etc.) are implemented by making a fresh copy of the list.
3. **AbstractSequentialList:** This class implements the [Collection interface](https://www.geeksforgeeks.org/collections-in-java-2/) and the AbstractCollection class. This class is used to implement an unmodifiable list, for which one needs to only extend this AbstractList Class and implement only the get() and the size() methods.
4. **ArrayList in Java**

ArrayList is a part of [**collection framework**](https://www.cdn.geeksforgeeks.org/collections-in-java-2/) and is present in java.util package. It provides us with dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed.



Since ArrayList is a dynamic array and we do not have to specify the size while creating it, the size of the arraylist automatically increases when we dynamically add and remove items. Though the actual library implementation may be more complex, the following is a very basic idea explaining the working of the array when the array becomes full and if we try to add an item:

* Creates a bigger-sized memory on heap memory (for example memory of double size).
* Copies the current memory elements to the new memory.
* New item is added now as there is bigger memory available now.
* Delete the old memory.

**Important Features:**

* ArrayList inherits [AbstractList](https://www.geeksforgeeks.org/abstractlist-in-java-with-examples/) class and implements the [List interface](https://www.geeksforgeeks.org/list-interface-java-examples/).
* ArrayList is initialized by the size. However, the size is increased automatically if the collection grows or shrinks if the [objects](https://www.geeksforgeeks.org/classes-objects-java/) are removed from the collection.
* Java ArrayList allows us to randomly access the list.
* ArrayList can not be used for [primitive types](https://www.geeksforgeeks.org/data-types-in-java/), like int, char, etc. We need a [wrapper class](https://www.geeksforgeeks.org/wrapper-classes-java/) for such cases.
* ArrayList in Java can be seen as a [vector in C++](https://www.geeksforgeeks.org/vector-in-cpp-stl/).
* ArrayList is not Synchronized. Its equivalent synchronized class in Java is [Vector](https://www.geeksforgeeks.org/java-util-vector-class-java/).

**Constructors in the ArrayList**

1. **ArrayList():** This constructor is used to build an empty array list. If we wish to create an empty ArrayList with the name arr, then, it can be created as:

ArrayList arr = new ArrayList(); 

2. **ArrayList(Collection c):** This constructor is used to build an array list initialized with the elements from the collection c. Suppose, we wish to create an ArrayList arr which contains the elements present in the collection c, then, it can be created as:   
  ArrayList arr = new ArrayList(c); 

3. **ArrayList(int capacity):** This constructor is used to build an array list with initial capacity being specified. Suppose we wish to create an ArrayList with the initial size being N, then, it can be created as:

ArrayList arr = new ArrayList(N);

1. **Generic integer ArrayList:**

ArrayList<Integer> arr = new ArrayList<Integer>();

**Methods in Java ArrayList :**

| Method | Description |
| --- | --- |
| 1. [add(int index, Object element)](https://www.geeksforgeeks.org/java-util-arraylist-add-method-java/) | insert a specific element at a specific position index in a list. |
| 1. [add(Object o)](https://www.geeksforgeeks.org/java-util-arraylist-add-method-java/) | append a specific element to the end of a list. |
| 1. [addAll(Collection C)](https://www.geeksforgeeks.org/java-util-arraylist-addall-method-java/) | append all the elements from a specific collection to the end of the mentioned list, in such an order that the values are returned by the specified collection’s iterator. |
| 1. [addAll(int index, Collection C)](https://www.geeksforgeeks.org/java-util-arraylist-addall-method-java/) | Used to insert all of the elements starting at the specified position from a specific collection into the mentioned list. |
| 1. [clear()](https://www.geeksforgeeks.org/arraylist-clear-java-examples/) | remove all the elements from any list. |
| 1. [clone()](https://www.geeksforgeeks.org/clone-method-in-java-2/) | return a shallow copy of an ArrayList. |
| 1. [contains(Object o)](https://www.geeksforgeeks.org/arraylist-contains-java/) | Returns true if this list contains the specified element. |
| 1. [ensureCapacity(int minCapacity)](https://www.geeksforgeeks.org/arraylist-ensurecapacity-method-in-java-with-examples/) | Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument. |
| 1. [forEach(Consumer<E> action)](https://www.geeksforgeeks.org/arraylist-foreach-method-in-java/) | Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception. |
| 1. [get(int index)](https://www.geeksforgeeks.org/arraylist-get-method-java-examples/) | Returns the element at the specified position in this list. |
| 1. [indexOf(Object O)](https://www.geeksforgeeks.org/java-util-arraylist-indexof-java/) | Returns the index of the first occurrence of a specific element, or -1 in case the element is not in the list. |
| 1. [isEmpty()](https://www.geeksforgeeks.org/arraylist-isempty-java-example/) | Returns true if this list contains no elements. |
| 1. [lastIndexOf(Object O)](https://www.geeksforgeeks.org/arraylist-lastindexof-java-example/) | The index of the last occurrence of a specific element is either returned or -1 in case the element is not in the list. |
| 1. [listIterator()](https://www.geeksforgeeks.org/arraylist-listiterator-method-in-java-with-examples/) | Returns a list iterator over the elements in this list (in proper sequence). |
| 1. [listIterator(int index)](https://www.geeksforgeeks.org/arraylist-listiterator-method-in-java-with-examples/) | Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list. |
| 1. [remove(int index)](https://www.geeksforgeeks.org/arraylist-linkedlist-remove-methods-java-examples/) | Removes the element at the specified position in this list. |
| 1. [remove(Object o)](https://www.geeksforgeeks.org/arraylist-linkedlist-remove-methods-java-examples/) | Removes the first occurrence of the specified element from this list, if it is present. |
| 1. [removeAll(Collection c)](https://www.geeksforgeeks.org/arraylist-removeall-method-in-java-with-examples/) | Removes from this list all of its elements that are contained in the specified collection. |
| 1. [removeIf(Predicate filter)](https://www.geeksforgeeks.org/arraylist-removeif-method-in-java/) | Removes all of the elements of this collection that satisfy the given predicate. |
| 1. [removeRange(int fromIndex, int toIndex)](https://www.geeksforgeeks.org/arraylist-removerange-java-examples/) | Removes from this list all of the elements whose index is between fromIndex, inclusive, and toIndex, exclusive. |
| 1. [retainAll(Collection c)](https://www.geeksforgeeks.org/arraylist-retainall-method-in-java/) | Retains only the elements in this list that are contained in the specified collection. |
| 1. [set(int index, E element)](https://www.geeksforgeeks.org/arraylist-set-method-in-java-with-examples/) | Replaces the element at the specified position in this list with the specified element. |
| 1. [size()](https://www.geeksforgeeks.org/arraylist-size-method-in-java-with-examples/) | Returns the number of elements in this list. |
| 1. [spliterator()](https://www.geeksforgeeks.org/arraylist-spliterator-method-in-java/) | Creates a late-binding and fail-fast Spliterator over the elements in this list. |
| 1. [subList(int fromIndex, int toIndex)](https://www.geeksforgeeks.org/arraylist-sublist-method-in-java-with-examples/) | Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive. |
| 1. [toArray()](https://www.geeksforgeeks.org/arraylist-array-conversion-java-toarray-methods/) | return an array containing all of the elements in the list in the correct order. |
| 1. [toArray(Object[] O)](https://www.geeksforgeeks.org/arraylist-array-conversion-java-toarray-methods/) | return an array containing all of the elements in this list in the correct order same as the previous method. |
| 1. [trimToSize()](https://www.geeksforgeeks.org/arraylist-trimtosize-java-example/) | This method is used to trim the capacity of the instance of the ArrayList to the list's current size. |

1. **LinkedList in Java**

* Linked List is a part of the [Collection framework](https://www.geeksforgeeks.org/collections-in-java-2/) present in [java.util package](https://www.geeksforgeeks.org/java-util-package-java/).
* This class is an implementation of the [LinkedList data structure](https://www.geeksforgeeks.org/data-structures/linked-list/) which is a linear data structure where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part.
* The elements are linked using pointers and addresses. Each element is known as a node.
* Due to the dynamicity and ease of insertions and deletions, they are preferred over the arrays.
* It also has a few disadvantages like the nodes cannot be accessed directly instead we need to start from the head and follow through the link to reach a node we wish to access.

**Constructors in the LinkedList**

**1. LinkedList():** This constructor is used to create an empty linked list. If we wish to create an empty LinkedList with the name ll, then, it can be created as:

LinkedList ll = new LinkedList();

**2. LinkedList(Collection C):** This constructor is used to create an ordered list that contains all the elements of a specified collection, as returned by the collection's iterator. If we wish to create a LinkedList with the name ll, then, it can be created as:

LinkedList ll = new LinkedList(C);

**Methods for Java LinkedList:**

| **Method** | **Description** |
| --- | --- |
| 1. [add(int index, E element)](https://www.geeksforgeeks.org/java-util-linkedlist-add-method-in-java/) | Inserts the specified element at the specified position in this list. |
| 1. [add(E elements)](https://www.geeksforgeeks.org/java-util-linkedlist-add-method-in-java/) | appends the specified element to the end of this list. |
| 1. [addAll(int index, Collection<E> c)](https://www.geeksforgeeks.org/java-util-linkedlist-addall-method-in-java/) | Inserts all of the elements in the specified collection into this list, starting at the specified position. |
| 1. [addAll(Collection<E> c)](https://www.geeksforgeeks.org/java-util-linkedlist-addall-method-in-java/) | appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator. |
| 1. [addFirst(E e)](https://www.geeksforgeeks.org/linkedlist-addfirst-method-in-java/) | Inserts the specified element at the beginning of this list. |
| 1. [addLast(E e)](https://www.geeksforgeeks.org/linkedlist-addlast-method-in-java/) | Appends the specified element to the end of this list. |
| 1. [clear()](https://www.geeksforgeeks.org/linkedlist-clear-method-in-java/) | Removes all of the elements from this list. |
| 1. [clone()](https://www.geeksforgeeks.org/linkedlist-clone-method-in-java/) | Returns a shallow copy of this LinkedList. |
| 1. [contains(Object o)](https://www.geeksforgeeks.org/linkedlist-contains-method-in-java/) | Returns true if this list contains the specified element. |
| 1. [descendingIterator()](https://www.geeksforgeeks.org/linkedlist-descendingiterator-method-in-java-with-examples/) | Returns an iterator over the elements in this linked-list in reverse sequential order. |
| 1. [element()](https://www.geeksforgeeks.org/linkedlist-element-method-in-java-with-%20examples/) | Retrieves but does not remove, the head (first element) of this list. |
| 1. [get(int index)](https://www.geeksforgeeks.org/linkedlist-get-method-in-java/) | Returns the element at the specified position in this list. |
| 1. [getFirst()](https://www.geeksforgeeks.org/java-util-linkedlist-get-getfirst-getlast-java/) | Returns the first element in this list. |
| 1. [getLast()](https://www.geeksforgeeks.org/linkedlist-getlast-method-in-java/) | Returns the last element in this list. |
| 1. [indexOf(Object o)](https://www.geeksforgeeks.org/linkedlist-indexof-method-in-java/) | Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element. |
| 1. [lastIndexOf(Object o)](https://www.geeksforgeeks.org/linkedlist-lastindexof-method-in-java/) | Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element. |
| 1. [listIterator(int index)](https://www.geeksforgeeks.org/linkedlist-listiterator-method-in-java/) | Returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list. |
| 1. [offer(E e)](https://www.geeksforgeeks.org/java-util-linkedlist-offer-offerfirst-offerlast-java/) | Adds the specified element as the tail (last element) of this list. |
| 1. [offerFirst(E e)](https://www.geeksforgeeks.org/java-util-linkedlist-offer-offerfirst-offerlast-java/) | Inserts the specified element at the front of this list. |
| 1. [offerLast(E e)](https://www.geeksforgeeks.org/java-util-linkedlist-offer-offerfirst-offerlast-java/) | Inserts the specified element at the end of this list. |
| 1. [peek()](https://www.geeksforgeeks.org/java-util-linkedlist-peek-peekfirst-peeklast-java/) | Retrieves but does not remove, the head (first element) of this list. |
| 1. [peekFirst()](https://www.geeksforgeeks.org/java-util-linkedlist-peek-peekfirst-peeklast-java/) | This method retrieves, but does not remove, the first element of this list, or returns null if this list is empty. |
| 1. [peekLast()](https://www.geeksforgeeks.org/java-util-linkedlist-peek-peekfirst-peeklast-java/) | This method retrieves, but does not remove, the last element of this list, or returns null if this list is empty. |
| 1. [poll()](https://www.geeksforgeeks.org/java-util-linkedlist-poll-pollfirst-polllast-%20examples-java/) | Retrieves and removes the head (first element) of this list. |
| 1. [pollFirst()](https://www.geeksforgeeks.org/java-util-linkedlist-poll-pollfirst-polllast-%20examples-java/) | Retrieves and removes the first element of this list, or returns null if this list is empty. |
| 1. [pollLast()](https://www.geeksforgeeks.org/java-util-linkedlist-poll-pollfirst-polllast-%20examples-java/) | Retrieves and removes the last element of this list, or returns null if this list is empty. |
| 1. [pop()](https://www.geeksforgeeks.org/linkedlist-pop-method-in-java/) | Pops an element from the stack represented by this list. |
| 1. [push(E e)](https://www.geeksforgeeks.org/linkedlist-push-method-in-java/) | Pushes an element onto the stack represented by this list. |
| 1. [remove()](https://www.geeksforgeeks.org/linkedlist-remove-method-in-java/) | Retrieves and removes the head (first element) of this list. |
| 1. [remove(int index)](https://www.geeksforgeeks.org/linkedlist-remove-method-in-java/) | Removes the element at the specified position in this list. |
| 1. [remove(Object o)](https://www.geeksforgeeks.org/linkedlist-remove-method-in-java/) | Removes the first occurrence of the specified element from this list if it is present. |
| 1. [removeFirst()](https://www.geeksforgeeks.org/linkedlist-removefirst-method-in-java/) | Removes and returns the first element from this list. |
| 1. [removeFirstOccurrence(Object o)](https://www.geeksforgeeks.org/linkedlist-removefirstoccurrence-method-in-%20java/) | Removes the first occurrence of the specified element in this list (when traversing the list from head to tail). |
| 1. [removeLast()](https://www.geeksforgeeks.org/linkedlist-removelast-method-in-java/) | Removes and returns the last element from this list. |
| 1. [removeLastOccurrence(Object o)](https://www.geeksforgeeks.org/linkedlist-removelastoccurrence-method-in-java-with-example/) | Removes the last occurrence of the specified element in this list (when traversing the list from head to tail). |
| 1. [set(int index, E element)](https://www.geeksforgeeks.org/linkedlist-set-method-in-java/) | Replaces the element at the specified position in this list with the specified element. |
| 1. [size()](https://www.geeksforgeeks.org/linkedlist-size-method-in-java/) | Returns the number of elements in this list. |
| 1. [spliterator()](https://www.geeksforgeeks.org/linkedlist-spliterator-method-in-java/) | Creates a late-binding and fail-fast spliterator over the elements in this list. |
| 1. [toArray()](https://www.geeksforgeeks.org/linkedlist-toarray-method-in-java-with-example/) | Returns an array containing all of the elements in this list in proper sequence (from first to last element). |
| 1. [toArray(T[] a)](https://www.geeksforgeeks.org/linkedlist-toarray-method-in-java-with-example/) | Returns an array containing all of the elements in this list in proper sequence (from first to last element) |
| 1. toString() | Returns a string containing all of the elements in this list in proper sequence (from first to the last element), each element is separated by commas and the String is enclosed in square brackets. |

1. **Vector Class in Java**

* It is found in[java.util package](https://www.geeksforgeeks.org/java-util-package-java/) and implement the [List](https://www.geeksforgeeks.org/list-interface-java-examples/) interface, so we can use all the methods of the List interface.
* Vector implements a dynamic array which means it can grow or shrink as required. Like an array, it contains components that can be accessed using an integer index.
* They are very similar to [ArrayList](https://www.geeksforgeeks.org/arraylist-in-java/), but Vector is synchronized -> poor performance and has some legacy methods that the collection framework does not contain.

**Important points regarding the Increment of vector capacity are as follows:**

If the increment is specified, Vector will expand according to it in each allocation cycle. Still, if the increment is not specified, then the vector’s capacity gets doubled in each allocation cycle. Vector defines three protected data members:

* int capacityIncrement: Contains the increment value.
* int elementCount: Number of elements currently in vector stored in it.
* Object elementData[]: Array that holds the vector is stored in it.

**Constructors:**

**1. Vector():**

Vector<E> v = new Vector<E>();

**2. Vector(int size):** Creates a vector whose initial capacity is specified by size.

Vector<E> v = new Vector<E>(int size);

**3. Vector(int size, int incr):**Creates a vector whose initial capacity is specified by size and increment is specified by incr. It specifies the number of elements to allocate each time a vector is resized upward.

Vector<E> v = new Vector<E>(int size, int incr);

**4. Vector(Collection c):**Creates a vector that contains the elements of collection c.

Vector<E> v = new Vector<E>(Collection c);

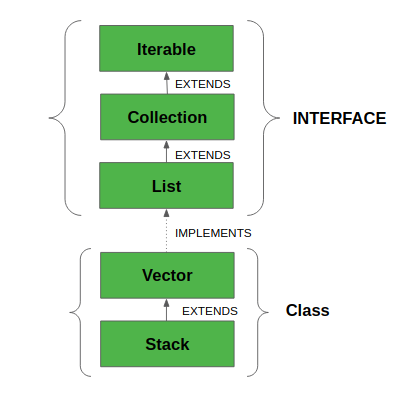
**Methods in Vector Class:**

| **Method** | **Description** |
| --- | --- |
| [add(E e)](https://www.geeksforgeeks.org/vector-add-method-in-java/) | Appends the specified element to the end of this Vector. |
| [add(int index, E element)](https://www.geeksforgeeks.org/vector-add-method-in-java/) | Inserts the specified element at the specified position in this Vector. |
| [addAll(Collection<?](https://www.geeksforgeeks.org/vector-addall-method-in-java/)  [extends E> c)](https://www.geeksforgeeks.org/vector-addall-method-in-java/) | Appends all of the elements in the specified Collection to the end of this Vector, in the order that they are returned by the specified Collection’s Iterator. |
| [addAll(int index,](https://www.geeksforgeeks.org/vector-addall-method-in-java/)  [Collection<E> c)](https://www.geeksforgeeks.org/vector-addall-method-in-java/) | Insert all of the elements in the specified Collection into this Vector at the specified position. |
| [addElement(E obj)](https://www.geeksforgeeks.org/vector-addelement-method-in-java/) | Adds the specified component to the end of this vector, increasing its size by one. |
| [capacity()](https://www.geeksforgeeks.org/vector-capacity-method-in-java/) | Returns the current capacity of this vector. |
| [clear()](https://www.geeksforgeeks.org/vector-clear-method-in-java/) | Removes all of the elements from this Vector. |
| [clone()](https://www.geeksforgeeks.org/vector-clone-method-in-java-with-examples/) | Returns a clone of this vector. |
| [contains(Object o)](https://www.geeksforgeeks.org/vector-contains-method-in-java/) | Returns true if this vector contains the specified element. |
| [containsAll(Collection<?> c)](https://www.geeksforgeeks.org/vector-containsall-method-in-java/) | Returns true if this Vector contains all of the elements in the specified Collection. |
| [copyInto(Object[] anArray)](https://www.geeksforgeeks.org/vector-copyinto-method-in-java/) | Copies the components of this vector into the specified array. |
| [elementAt(int index)](https://www.geeksforgeeks.org/vector-elementat-method-in-java/) | Returns the component at the specified index. |
| [elements()](https://www.geeksforgeeks.org/vector-elements-method-in-java/) | Returns an enumeration of the components of this vector. |
| [ensureCapacity(int minCapacity)](https://www.geeksforgeeks.org/arraylist-ensurecapacity-method-in-java-with-examples/) | Increases the capacity of this vector, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument. |
| [equals(Object o)](https://www.geeksforgeeks.org/vector-equals-method-in-java/) | Compares the specified Object with this Vector for equality. |
| [firstElement()](https://www.geeksforgeeks.org/vector-firstelement-method-in-java/) | Returns the first component (the item at index 0) of this vector. |
| [forEach(Consumer<?](https://www.geeksforgeeks.org/vector-foreach-method-in-java/)  [super E> action)](https://www.geeksforgeeks.org/vector-foreach-method-in-java/) | Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception. |
| [get(int index)](https://www.geeksforgeeks.org/vector-get-method-in-java/) | Returns the element at the specified position in this Vector. |
| [hashCode()](https://www.geeksforgeeks.org/vector-hashcode-method-in-java/) | Returns the hash code value for this Vector. |
| [indexOf(Object o)](https://www.geeksforgeeks.org/vector-indexof-method-in-java/#:~:text=) | Returns the index of the first occurrence of the specified element in this vector,  or -1 if this vector does not contain the element. |
| [indexOf(Object o, int index)](https://www.geeksforgeeks.org/vector-indexof-method-in-java/#:~:text=) | Returns the index of the first occurrence of the specified element in this vector, searching forwards from the index, or returns -1 if the element is not found. |
| [insertElementAt(E obj, int index)](https://www.geeksforgeeks.org/vector-insertelementat-method-in-java/) | Inserts the specified object as a component in this vector at the specified index. |
| [isEmpty()](https://www.geeksforgeeks.org/vector-isempty-method-in-java/) | Tests if this vector has no components. |
| [iterator()](https://www.geeksforgeeks.org/vector-iterator-method-in-java-with-examples/) | Returns an iterator over the elements in this list in a proper sequence. |
| [lastElement()](https://www.geeksforgeeks.org/vector-lastelement-method-in-java/) | Returns the last component of the vector. |
| [lastIndexOf(Object o)](https://www.geeksforgeeks.org/vector-lastindexof-method-in-java/) | Returns the index of the last occurrence of the specified element in this vector,  or -1 if this vector does not contain the element. |
| [lastIndexOf(Object o, int index)](https://www.geeksforgeeks.org/vector-lastindexof-method-in-java/) | Returns the index of the last occurrence of the specified element in this vector, searching backward from the index, or returns -1 if the element is not found. |
| [listIterator()](https://www.geeksforgeeks.org/vector-listiterator-method-in-java-with-examples/) | Returns a list iterator over the elements in this list (in proper sequence). |
| [listIterator(int index)](https://www.geeksforgeeks.org/vector-listiterator-method-in-java-with-examples/) | Returns a list iterator over the elements in this list (in proper sequence),  starting at the specified position in the list. |
| [remove(int index)](https://www.geeksforgeeks.org/vector-remove-method-in-java/) | Removes the element at the specified position in this Vector. |
| [remove(Object o)](https://www.geeksforgeeks.org/vector-remove-method-in-java/) | Removes the first occurrence of the specified element in this Vector. If the Vector does not contain the element, it is unchanged. |
| [removeAll(Collection<?> c)](https://www.geeksforgeeks.org/vector-removeall-method-in-java/) | Removes from this Vector all of its elements contained in the specified Collection. |
| [removeAllElements()](https://www.geeksforgeeks.org/vector-removeallelements-method-in-java-with-example/) | Removes all components from this vector and sets its size to zero. |
| [removeElement(Object obj)](https://www.geeksforgeeks.org/vector-removeelement-method-in-java-with-example/) | Removes the first (lowest-indexed) occurrence of the argument from this vector. |
| [removeElementAt(int index)](https://www.geeksforgeeks.org/vector-removeelementat-method-in-java/) | Deletes the component at the specified index. |
| [removeIf(Predicate<? super E> filter)](https://www.geeksforgeeks.org/vector-removeif-method-in-java/) | Removes all of the elements of this collection that satisfy the given predicate. |
| [removeRange(int fromIndex,](https://www.geeksforgeeks.org/vector-removerange-method-in-java-with-example/)  [int toIndex)](https://www.geeksforgeeks.org/vector-removerange-method-in-java-with-example/) | Removes from this list all of the elements whose index is between fromIndex, inclusive, and toIndex, exclusive. |
| replaceAll(UnaryOperator<E> operator) | Replaces each element of this list with the result of applying the operator to that element. |
| [retainAll(Collection<?> c)](https://www.geeksforgeeks.org/vector-retainall-method-in-java-with-examples/) | Retains only the elements in this Vector contained in the specified Collection. |
| [set(int index, E element)](https://www.geeksforgeeks.org/vector-set-method-in-java/#:~:text=set() method is used,Vector class, with another element.&text=) | Replaces the element at the specified position in this Vector with the specified element. |
| [setElementAt(E obj, int index)](https://www.geeksforgeeks.org/vector-setelementat-method-in-java-with-example/) | Sets the component at the specified index of this vector to be the specified object. |
| [setSize(int newSize)](https://www.geeksforgeeks.org/vector-setsize-method-in-java-with-example/) | Sets the size of this vector. |
| [size()](https://www.geeksforgeeks.org/vector-size-method-in-java/) | Returns the number of components in this vector. |
| [sort(Comparator<? super E> c)](https://www.geeksforgeeks.org/collections-sort-java-examples/) | Sorts this list according to the order induced by the specified Comparator. |
| [spliterator()](https://www.geeksforgeeks.org/arraylist-spliterator-method-in-java/) | Creates a late-binding and fail-fast Spliterator over the elements in this list. |
| [subList(int fromIndex, int toIndex)](https://www.geeksforgeeks.org/vector-sublist-method-in-java/) | Returns a view of the portion of this List between fromIndex, inclusive, and toIndex, exclusive. |
| [toArray()](https://www.geeksforgeeks.org/vector-toarray-method-in-java-with-examples/) | Returns an array containing all of the elements in this Vector in the correct order. |
| [toArray(T[] a)](https://www.geeksforgeeks.org/vector-toarray-method-in-java-with-examples/) | Returns an array containing all of the elements in this Vector in the correct order; the runtime type of the returned array is that of the specified array. |
| [toString()](https://www.geeksforgeeks.org/vector-tostring-method-in-java-with-example/) | Returns a string representation of this Vector, containing the String representation of each element. |
| [trimToSize()](https://www.geeksforgeeks.org/vector-trimtosize-method-in-java-with-example/) | Trims the capacity of this vector to be the vector’s current size. |

1. **Stack Class in Java**

Java [Collection framework](https://www.geeksforgeeks.org/collections-in-java-2/) provides a Stack class that models and implements a [Stack data structure](https://www.geeksforgeeks.org/stack-data-structure/). The class is based on the basic principle of last-in-first-out. In addition to the basic push and pop operations, the class provides three more functions of empty, search, and peek. The class can also be said to extend Vector and treats the class as a stack with the five mentioned functions. The class can also be referred to as the subclass of Vector.

**Hierarchy of the Stack class:**



**How to Create a Stack?**

1. import java.util.stack package and use the Stack() constructor of this class.
2. Stack<E> stack = new Stack<E>(); OR Stack stack = new Stack();

Here E is the type of Object.

**Methods in Stack Class:**

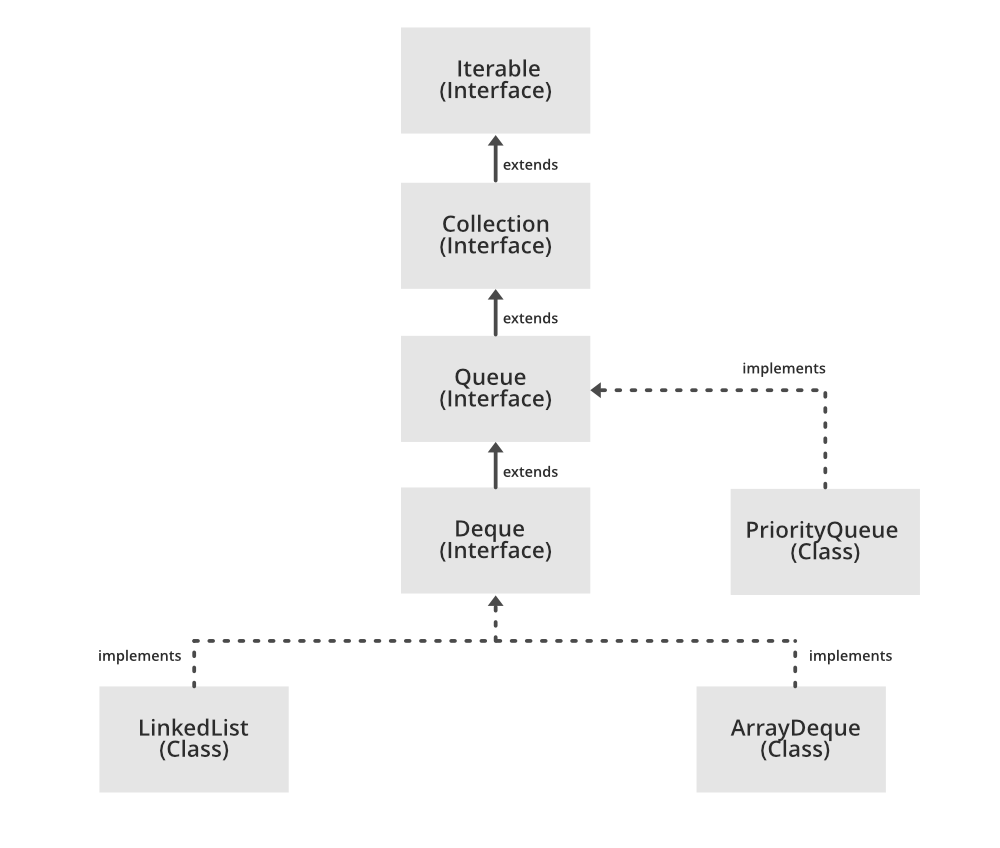
| **METHOD** | **DESCRIPTION** |
| --- | --- |
| is[Empty()](https://www.geeksforgeeks.org/stack-empty-method-in-java/) | It returns true if nothing is on the top of the stack. Else, returns false. |
| [peek()](https://www.geeksforgeeks.org/stack-peek-method-in-java/) | Returns the element on the top of the stack, but does not remove it. |
| [pop()](https://www.geeksforgeeks.org/stack-pop-method-in-java/) | Removes and returns the top element of the stack. An 'EmptyStackException'  is thrown if we call pop() when the invoking stack is empty. |
| [push(Object element)](https://www.geeksforgeeks.org/stack-push-method-in-java/) | Pushes an element on the top of the stack. |
| [search(Object element)](https://www.geeksforgeeks.org/stack-search-method-in-java/) | It determines whether an object exists in the stack. If the element is found, it returns the position of the element from the top of the stack. Else, it returns -1 |

**Note:**

1. Stack class in Java is a legacy class and inherits all methods from [Vector class in Java](https://www.geeksforgeeks.org/java-util-vector-class-java/).
2. It is a thread-safe class and hence involves overhead when we do not need thread safety.
3. It is recommended to use [ArrayDeque](https://www.geeksforgeeks.org/arraydeque-in-java/) for stack implementation as it is more efficient in a single-threaded environment.

**2. Queue Interface In Java**

The Queue interface is present in [java.util](https://www.geeksforgeeks.org/java-util-package-java/) package and extends the [Collection interface](https://www.geeksforgeeks.org/collections-in-java-2/). It is an ordered list of objects with its use limited to inserting elements at the end of the list and deleting elements from the start of the list, (i.e.), it follows the FIFO or the First-In-First-Out principle.



Being an interface the queue needs a concrete class for the declaration and the most common classes are the [PriorityQueue](https://www.geeksforgeeks.org/priority-queue-class-in-java-2/) and [LinkedList](https://www.geeksforgeeks.org/linked-list-in-java/) in Java. Note that neither of these implementations is thread-safe. [PriorityBlockingQueue](https://www.geeksforgeeks.org/priorityblockingqueue-class-in-java/) is one alternative implementation if the thread-safe implementation is needed.

**Creating Queue Objects:** Since Queue is an [interface](https://www.geeksforgeeks.org/interfaces-in-java/), objects cannot be created of the type queue. We always need a class which extends this list in order to create an object.

// Obj is the type of the object to be stored in Queue

Queue<Obj> queue = new PriorityQueue<Obj> ();

**Characteristics of a Queue:**

1. The Java Queue supports all methods of Collection interface including insertion, deletion, etc.
2. [**LinkedList**](https://www.geeksforgeeks.org/linked-list-in-java/), **ArrayBlockingQueue** and [**PriorityQueue**](https://www.geeksforgeeks.org/priority-queue-class-in-java-2/)are the most frequently used implementations.
3. If any null operation is performed on **BlockingQueues**, NullPointerException is thrown.
4. The Queues which are available in **java.util** package are Unbounded Queues.
5. The Queues which are available in **java.util.concurrent** package are the Bounded Queues.
6. All Queues except the **Deques** supports insertion at the tail and removal at head of the queue. The Deques support element insertion and removal at both ends.

**Methods of Queue Interface**

| Method | Description |
| --- | --- |
| [iterator()](https://www.geeksforgeeks.org/priorityqueue-iterator-method-in-java/) | Returns an iterator over the elements in this queue. |
| [size()](https://www.geeksforgeeks.org/list-size-method-in-java-with-examples/) | This method is used to return the size of the queue. |
| [clear()](https://www.geeksforgeeks.org/list-clear-method-in-java-with-examples/) | This method is used to remove all the elements in the queue. However, the reference of the queue created is still stored. |
| remove(element) | This method is used to remove and return the first occurrence of the given element in the queue. |
| boolean add(object) | This method is used to insert the specified element into a queue and return true upon success. |
| Object poll() | This method is used to retrieve and removes the head of the queue, or returns null if the queue is empty. |
| Object element() | This method is used to retrieves, but does not remove, the head of queue. |
| Object peek() | This method is used to retrieves, but does not remove, the head of this queue, or returns null if this queue is empty. |

1. **PriorityQueue in Java**

A PriorityQueue is used when the objects are supposed to be processed based on the priority. It is known that a [Queue](https://www.geeksforgeeks.org/queue-interface-java/) follows the First-In-First-Out algorithm, but sometimes the elements of the queue are needed to be processed according to the priority, that's when the PriorityQueue comes into play.

The PriorityQueue is based on the priority heap. The elements of the priority queue are ordered according to the natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used.

**Characteristics of Priority Queue:**

* PriorityQueue doesn’t permit null.
* We can’t create a PriorityQueue of Objects that are non-comparable.
* PriorityQueue are unbound queues.
* The head of this queue is the least element with respect to the specified ordering. If multiple elements are tied for the least value, the head is one of those elements -- ties are broken arbitrarily.
* It provides O(log(n)) time for add and poll methods.
* It inherits methods from AbstractQueue, AbstractCollection, Collection, and Object class.

**Constructors:**

**1. PriorityQueue():** Creates a PriorityQueue with the default initial capacity (11) that orders its elements according to their natural ordering.

PriorityQueue<E> pq = new PriorityQueue<E>();

**2. PriorityQueue(Collection<E> c):** Creates a PriorityQueue containing the elements in the specified collection.

PriorityQueue<E> pq = new PriorityQueue<E>(Collection<E> c);

**3. PriorityQueue(int initialCapacity):** Creates a PriorityQueue with the specified initial capacity that orders its elements according to their natural ordering.

PriorityQueue<E> pq = new PriorityQueue<E>(int initialCapacity);

**4. PriorityQueue(int initialCapacity, Comparator<E> comparator):** Creates a PriorityQueue with the specified initial capacity that orders its elements according to the specified comparator.

PriorityQueue<E> pq = new PriorityQueue(int initialCapacity, Comparator<E> comparator);

**5. PriorityQueue(PriorityQueue<E> c):** Creates a PriorityQueue containing the elements in the specified priority queue.

PriorityQueue<E> pq = new PriorityQueue(PriorityQueue<E> c);

**6. PriorityQueue(SortedSet<E> c):** Creates a PriorityQueue containing the elements in the specified sorted set.

PriorityQueue<E> pq = new PriorityQueue<E>(SortedSet<E> c);

**Applications:**

* Implementing [Dijkstra's](https://www.geeksforgeeks.org/dijkstras-algorithm-for-adjacency-list-representation-greedy-algo-8/) and [Prim's](https://www.geeksforgeeks.org/prims-algorithm-using-priority_queue-stl/) algorithms.
* [Maximize array sum after K negations](https://www.geeksforgeeks.org/maximize-array-sum-k-negations-set-2/)

**Note**: PriorityQueue by default prioritize integer element in ascending order.

Example:

// Java program to iterate elements to a PriorityQueue

import java.util.\*;

public class PriorityQueueDemo {

// Main Method

public static void main(String args[])

{

PriorityQueue<String> pq = new PriorityQueue<>();

pq.add("Geeks");

pq.add("For");

pq.add("Geeks");

Iterator iterator = pq.iterator();

while (iterator.hasNext()) {

System.out.print(iterator.next() + " ");

}

}

}

**2.2 Deque interface in Java**

* Deque Interface present in [java.util](https://www.geeksforgeeks.org/java-util-package-java/) package is a subtype of the [queue](https://www.geeksforgeeks.org/queue-interface-java/) interface.
* The Deque is related to the double-ended queue that supports adding or removing elements from either end of the data structure.
* It can either be used as a [queue(first-in-first-out/FIFO)](https://www.geeksforgeeks.org/queue/) or as a [stack(last-in-first-out/LIFO)](https://www.geeksforgeeks.org/stack/).

Creating Deque Objects

// Obj is the type of the object to be stored in Deque Deque<Obj> deque = new ArrayDeque<Obj> ();

**Example:**

// Java program to demonstrate the working of a Deque in Java

import java.util.\*;

public class DequeExample {

public static void main(String[] args)

{

Deque<String> deque = new LinkedList<String>();

// We can add elements to the queue n various ways

// Add at the last

deque.add("Element 1 (Tail)");

// Add at the first

deque.addFirst("Element 2 (Head)");

// Add at the last

deque.addLast("Element 3 (Tail)");

// Add at the first

deque.push("Element 4 (Head)");

// Add at the last

deque.offer("Element 5 (Tail)");

// Add at the first

deque.offerFirst("Element 6 (Head)");

System.out.println(deque + "\n");

// We can remove the first element or the last element.

deque.removeFirst();

deque.removeLast();

System.out.println("Deque after removing first and last: "+ deque);

}

}

**Methods of Deque Interface:**

| **Method** | **Description** |
| --- | --- |
| [add(element)](https://www.geeksforgeeks.org/deque-add-method-in-java/) | add an element at the tail of the queue. If the Deque is capacity restricted and no space is left for insertion, it returns an IllegalStateException. The function returns true on successful insertion. |
| [addFirst(element)](https://www.geeksforgeeks.org/deque-addfirst-method-in-java-with-examples/) | add an element at the head of the queue. If the Deque is capacity restricted and no space is left for insertion, it returns an IllegalStateException. The function returns true on successful insertion. |
| [addLast(element)](https://www.geeksforgeeks.org/deque-addlast-method-in-java/) | add an element at the tail of the queue. If the Deque is capacity restricted and no space is left for insertion, it returns an IllegalStateException. The function returns true on successful insertion. |
| [contains()](https://www.geeksforgeeks.org/deque-contains-method-in-java/) | check whether the queue contains the given object or not. |
| [descendingIterator()](https://www.geeksforgeeks.org/deque-descendingiterator-method-in-java/) | Returns an iterator for the deque. The elements will be returned in order from last(tail) to first(head). |
| [element()](https://www.geeksforgeeks.org/deque-element-method-in-java/) | Used to retrieve, but not remove, the head of the queue represented by this deque. |
| [getFirst()](https://www.geeksforgeeks.org/deque-getfirst-method-in-java/) | Used to retrieve, but not remove, the first element of this deque. |
| [getLast()](https://www.geeksforgeeks.org/deque-getlast-method-in-java/) | Used to retrieve, but not remove, the last element of this deque. |
| [iterator()](https://www.geeksforgeeks.org/deque-iterator-method-in-java/) | Returns an iterator for the deque. The elements will be returned in order from first (head) to last (tail). |
| [offer(element)](https://www.geeksforgeeks.org/deque-offer-method-in-java/) | Add an element at the tail of the queue. This method is preferable to add() method since this method does not throws an exception when the capacity of the container is full since it returns false. |
| [offerFirst(element)](https://www.geeksforgeeks.org/deque-offerfirst-method-in-java/) | Add an element at the head of the queue. This method is preferable to addFirst() method since this method does not throws an exception when the capacity of the container is full since it returns false. |
| [offerLast(element)](https://www.geeksforgeeks.org/deque-offerlast-method-in-java/) | Add an element at the tail of the queue. |
| peek() | Used to retrieve the element at the head of the deque but doesn’t remove the element from the deque. This method returns null if the deque is empty. |
| peekFirst() | Used to retrieve the element at the head of the deque but doesn’t remove the element from the deque. This method returns null if the deque is empty. |
| peekLast() | Used to retrieve the element at the tail of the deque but doesn’t remove the element from the deque. This method returns null if the deque is empty. |
| poll() | Used to retrieve and remove the element at the head of the deque. This method returns null if the deque is empty. |
| pollFirst() | Used to retrieve and remove the element at the head of the deque. This method returns null if the deque is empty. |
| pollLast() | Used to retrieve and remove the element at the tail of the deque. This method returns null if the deque is empty. |
| removeFirst() | Remove an element from the head of the queue. |
| removeLast() | Remove an element from the tail of the queue. |
| size() | This method is used to find and return the size of the deque. |

[**ArrayDeque**](https://www.geeksforgeeks.org/arraydeque-in-java/) **Class**

* The ArrayDeque class is an implementation of the Deque interface that uses a resizable array to store its elements.
* This is a special kind of queue that grows and allows users to add or remove an element from both sides of the queue.
* Array deques have no capacity restrictions and they grow as necessary to support usage.
* They are not thread-safe which means that in the absence of external synchronization, ArrayDeque does not support concurrent access by multiple threads.
* ArrayDeque class is likely to be faster than Stack when used as a stack.
* ArrayDeque class is likely to be faster than LinkedList when used as a queue.
* Null elements are prohibited in the ArrayDeque.

**Constructors:**

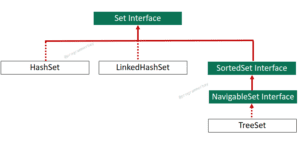
1. ArrayDeque<E> dq = new ArrayDeque<E>();
2. ArrayDeque<E> dq = new ArrayDeque<E>(Collection col);
3. ArrayDeque<E> dq = new ArrayDeque<E>(int capacity);

**Methods in ArrayDeque:**

| **METHOD** | **DESCRIPTION** |
| --- | --- |
| [add(Element e)](https://www.geeksforgeeks.org/arraydeque-add-method-in-java/) | Adds particular element at the end of the deque. |
| [addAll(Collection<E> c)](https://www.geeksforgeeks.org/arraydeque-addall-method-in-java/) | Adds all of the elements in the specified collection at the end of this deque. |
| [addFirst(Element e)](https://www.geeksforgeeks.org/arraydeque-addfirst-method-in-java/) | Adds particular element at the start of the deque. |
| [addLast(Element e)](https://www.geeksforgeeks.org/arraydeque-addlast-method-in-java/) | Adds a particular element at the end of the deque. It is similar to the add() method |
| [clear()](https://www.geeksforgeeks.org/arraydeque-clear-method-in-java/) | Removes all deque elements. |
| [clone()](https://www.geeksforgeeks.org/arraydeque-clone-method-in-java/) | Copies the deque. |
| [contains(Obj)](https://www.geeksforgeeks.org/arraydeque-contains-method-in-java/) | Checks whether a deque contains the element or not |
| [element()](https://www.geeksforgeeks.org/arraydeque-element-method-in-java/) | Returns element at the head of the deque |
| [forEach(Consumer<E> action)](https://www.geeksforgeeks.org/arraydeque-foreach-method-in-java/#:~:text=Method traverses each element of,are passed to the caller.) | Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception. |
| [getFirst()](https://www.geeksforgeeks.org/arraydeque-getfirst-method-in-java/) | Returns first element of the deque |
| [getLast()](https://www.geeksforgeeks.org/arraydeque-getlast-method-in-java/) | Returns last element of the deque |
| [isEmpty()](https://www.geeksforgeeks.org/arraydeque-isempty-method-in-java/) | Checks whether the deque is empty or not. |
| [iterator()](https://www.geeksforgeeks.org/arraydeque-iterator-method-in-java/#:~:text=util.&text=iterator() method is used,Iterator iterate_value = Array_Deque.) | Returns an iterator over the elements in this deque. |
| [offer(Element e)](https://www.geeksforgeeks.org/arraydeque-offer-method-in-java/) | Adds element at the end of deque. |
| [offerFirst(Element e)](https://www.geeksforgeeks.org/arraydeque-offerfirst-method-in-java/) | The method inserts element at the front of deque. |
| [offerLast(Element e)](https://www.geeksforgeeks.org/arraydeque-offerlast-method-in-java/) | The method inserts element at the end of the deque. |
| [peek()](https://www.geeksforgeeks.org/arraydeque-peek-method-in-java/) | The method returns head element without removing it. |
| [poll()](https://www.geeksforgeeks.org/arraydeque-poll-method-in-java/) | The method returns head element and also removes it |
| [pop()](https://www.geeksforgeeks.org/arraydeque-pop-method-in-java/) | Pops out an element for stack represented by deque |
| [push(Element e)](https://www.geeksforgeeks.org/arraydeque-push-method-in-java/) | Pushes an element onto stack represented by deque |
| [remove()](https://www.geeksforgeeks.org/arraydeque-remove-method-in-java/) | The method returns head element and also removes it |
| [remove(Object o)](https://www.geeksforgeeks.org/arraydeque-remove-method-in-java/#:~:text=remove() method is used,the head of the Deque.&text=Parameters: The method does not,the head of the Deque.) | Removes a single instance of the specified element from this deque. |
| [removeAll(Collection c)](https://www.geeksforgeeks.org/arraydeque-removeall-method-in-java/) | Removes all of this collection’s elements that are also contained in the specified collection (optional operation). |
| [removeFirst()](https://www.geeksforgeeks.org/arraydeque-removefirst-method-in-java/) | Returns the first element and also removes it |
| [removeFirstOccurrence​(Object o)](https://www.geeksforgeeks.org/arraydeque-removefirstoccurrence-method-in-java/) | Removes the first occurrence of the specified element in this deque (when traversing the deque from head to tail). |
| [removeIf(Predicate<E> filter)](https://www.geeksforgeeks.org/java-8-arraydeque-removeif-method-in-java-with-examples/) | Removes all of the elements of this collection that satisfy the given predicate. |
| [removeLast()](https://www.geeksforgeeks.org/arraydeque-removelast-method-in-java/) | The method returns the last element and also removes it |
| [removeLastOccurrence(Object o)](https://www.geeksforgeeks.org/arraydeque-removelastoccurrence-method-in-java/#:~:text=util.,specific element from this deque.&text=Return Value: This method returns,Deque else it returns false.) | Removes the last occurrence of the specified element in this deque (when traversing the deque from head to tail). |
| retainAll(Collection c) | Retains only the elements in this collection that are contained in the specified collection (optional operation). |
| [size()](https://www.geeksforgeeks.org/arraydeque-size-method-in-java/#:~:text=ArrayDeque.,elements present in the Deque.&text=Parameters: The method does not,elements present in the Deque.) | Returns the number of elements in this deque. |
| [spliterator()](https://www.geeksforgeeks.org/arraydeque-spliterator-method-in-java/) | Creates a late-binding and fail-fast Spliterator over the elements in this deque. |
| [toArray()](https://www.geeksforgeeks.org/arraydeque-toarray-method-in-java/#:~:text=toArray() method is used,deque to a new array.&text=Parameters: The method does not take any parameters.) | Returns an array containing all of the elements in this deque in proper sequence (from first to the last element). |
| toArray(T[] a) | Returns an array containing all of the elements in this deque in proper sequence (from first to the last element); the runtime type of the returned array is that of the specified array. |

1. **Set Inteface in Java**

* The set interface is present in [java.util](https://www.geeksforgeeks.org/java-util-package-java/) package and extends the [Collection interface](https://www.geeksforgeeks.org/collections-in-java-2/).
* It is an unordered collection of objects in which duplicate values cannot be stored.
* It is an interface that implements the mathematical set.
* This interface contains the methods inherited from the Collection interface and adds a feature that restricts the insertion of the duplicate elements.
* There are two interfaces that extend the set implementation namely [SortedSet](https://www.geeksforgeeks.org/sortedset-java-examples/) and [NavigableSet](https://www.geeksforgeeks.org/navigableset-java-examples/).
* Null values are accepted by the Set.
* The insertion order is not retained in the Set.



**Creating Set Objects:**

Set<Obj> set = new HashSet<Obj> ();

**Methods of Set Interface:**

| **Method** | **Description** |
| --- | --- |
| [add(element)](https://www.geeksforgeeks.org/set-add-method-in-java-with-examples/) | Adds a specific element to the set. The function adds the element only if the specified element is not already present in the set else the function returns False if the element is already present in the Set. |
| [addAll(collection)](https://www.geeksforgeeks.org/set-addall-method-in-java-with-examples/) | Appends all of the elements from the mentioned collection to the existing set. The elements are added randomly without following any specific order. |
| [clear()](https://www.geeksforgeeks.org/set-clear-method-in-java-with-examples/) | Removes all the elements from the set but not delete the set. The reference for the set still exists. |
| [contains(element)](https://www.geeksforgeeks.org/set-contains-method-in-java-with-examples/) | Check whether a specific element is present in the Set or not. |
| [containsAll(collection)](https://www.geeksforgeeks.org/set-containsall-method-in-java-with-examples/) | Checks whether the set contains all the elements present in the given collection or not. This method returns true if the set contains all the elements and returns false if any of the elements are missing. |
| [hashCode()](https://www.geeksforgeeks.org/set-hashcode-method-in-java-with-examples/) | Used to get the hashCode value for this instance of the Set. It returns an integer value which is the hashCode value for this instance of the Set. |
| isEmpty() | Checks whether the set is empty or not. |
| [iterator()](https://www.geeksforgeeks.org/set-iterator-method-in-java-with-examples/) | Returns the [iterator](https://www.geeksforgeeks.org/iterators-in-java/) of the set. The elements from the set are returned in a random order. |
| [remove(element)](https://www.geeksforgeeks.org/set-remove-method-in-java-with-examples/) | Removes the given element from the set. This method returns True if the specified element is present in the Set otherwise it returns False. |
| [removeAll(collection)](https://www.geeksforgeeks.org/set-removeall-method-in-java-with-examples/) | Removes all the elements from the collection which are present in the set. This method returns true if this set changed as a result of the call. |
| [retainAll(collection)](https://www.geeksforgeeks.org/set-retainall-method-in-java-with-example/) | Retains all the elements from the set which are mentioned in the given collection. This method returns true if this set changed as a result of the call. |
| [size()](https://www.geeksforgeeks.org/set-size-method-in-java-with-example/) | Returns the number of elements in the set. |
| [toArray()](https://www.geeksforgeeks.org/set-toarray-method-in-java-with-example/) | Used to form an array of the same elements as that of the Set. |

**Example:**

Set<Integer> setA = new HashSet<Integer>();

setA.addAll(Arrays.asList(new Integer[] {2,4,6,8,10,12}));

Set<Integer> setB = new HashSet<Integer>(Arrays.asList(new Integer[] {3,6,9,12,15}));

//to find Union

Set<Integer> union = new HashSet<>(setA);

union.addAll(setB);

System.out.println("Union of set A and B: "+union);

//to find Intersection

Set<Integer> intersect = new HashSet<>(setA);

intersect.retainAll(setB);

System.out.println("Intersection of set A and B: "+intersect);

1. **HashSet in Java**

* Implements [Set Interface](https://www.geeksforgeeks.org/set-in-java/).
* The underlying data structure for HashSet is [Hashtable](https://www.geeksforgeeks.org/hashtable-in-java/).
* Objects that you insert in HashSet are not guaranteed to be inserted in the same order. Objects are inserted based on their hash code.
* NULL elements are allowed in HashSet.
* HashSet also implements Serializable and Cloneable interfaces.

**Internal working of a HashSet:**

All the classes of Set interface are internally backed up by Map. HashSet uses HashMap for storing its object internally. You must be wondering that to enter a value in HashMap we need a key-value pair, but in HashSet, we are passing only one value.

**Storage in HashMap:**

Actually the value we insert in HashSet acts as a key to the map Object and for its value, java uses a constant variable. So in the key-value pair, all the values will be the same.

**Note**: HashSet not only stores unique Objects but also a unique Collection of Object like [ArrayList<E>](https://www.geeksforgeeks.org/arraylist-in-java/), [LinkedList<E>](https://www.geeksforgeeks.org/linked-list-in-java/), [Vector<E>](https://www.geeksforgeeks.org/java-util-vector-class-java/#:~:text=The Vector class implements a growable array of objects.&text=They are very similar to,AbstractList and implements List interfaces.),..etc.

Example:

import java.util.\*;

class CollectionObjectStorage {

public static void main(String[] args)

{

// Instantiate an object of HashSet

HashSet<ArrayList> set = new HashSet<>();

// create ArrayList list1

ArrayList<Integer> list1 = new ArrayList<>();

// create ArrayList list2

ArrayList<Integer> list2 = new ArrayList<>();

// Add elements using add method

list1.add(1);

list1.add(2);

list2.add(1);

list2.add(2);

set.add(list1);

set.add(list2);

// print the set size to understand the internal storage of ArrayList in Set

System.out.println(set.size()); //output = 1

}

}

Before storing an Object, HashSet checks whether there is an existing entry using [hashCode() and equals() methods.](https://www.geeksforgeeks.org/equals-hashcode-methods-java/) In the above example, two lists are considered equal if they have the same elements in the same order.

When you invoke the [hashCode()](https://docs.oracle.com/javase/7/docs/api/java/util/AbstractList.html#hashCode()) method on the two lists, they both would give the same hash since they are equal. So, it stores only the first one, here it is list1.

**Constructors of HashSet class:**

1. **HashSet():** This constructor is used to build an empty HashSet object in which the default initial capacity is 16 and the default load factor is 0.75.

HashSet<E> hs = new HashSet<E>();

**2. HashSet(int initialCapacity):** This constructor is used to build an empty HashSet object in which the initialCapacity is specified at the time of object creation. Here, the default loadFactor remains 0.75.

HashSet<E> hs = new HashSet<E>(int initialCapacity);

**3. HashSet(int initialCapacity, float loadFactor):** This constructor is used to build an empty HashSet object in which the initialCapacity and loadFactor are specified at the time of object creation.

HashSet<E> hs = new HashSet<E>(int initialCapacity, float loadFactor);

**4. HashSet(Collection):** This constructor is used to build a HashSet object containing all the elements from the given collection.

HashSet<E> hs = new HashSet<E>(Collection C);

**Methods:**

HashSet has all methods of Set, and Collections interface.

1. **LinkedHashSet in Java**

* The LinkedHashSet is an ordered version of HashSet that maintains a doubly-linked List across all elements.
* When the insertion order is needed to be maintained this class is used. When iterating through a [HashSet](https://www.geeksforgeeks.org/hashset-in-java/) the order is unpredictable, while a LinkedHashSet lets us iterate through the elements in the order in which they were inserted.
* It extends the HashSet class and implements the Set interface.

**Constructors of LinkedHashSet Class**

1. **LinkedHashSet():**

LinkedHashSet<E> hs = new LinkedHashSet<E>();

1. **LinkedHashSet(Collection C):**

LinkedHashSet<E> hs = new LinkedHashSet<E>(Collection c);

1. **LinkedHashSet(int capacity):**

LinkedHashSet<E> hs = new LinkedHashSet<E>(int capacity);

1. **LinkedHashSet(int capacity, float fillRatio):**Can be used to initialize both the capacity and the fill ratio, also called the load capacity of the LinkedHashSet with the arguments mentioned in the parameter. When the number of elements exceeds the capacity of the LinkedHashSet is multiplied with the fill ratio thus expanding the capacity of the LinkedHashSet.

LinkedHashSet<E> hs = new LinkedHashSet<E>(int capacity, int fillRatio);

**Note**: Keeping the insertion order in both LinkedHashmap and LinkedHashset have additional associated costs, both in terms of spending additional CPU cycles and needing more memory. If you do not need the insertion order maintained, it is recommended to use the lighter-weight [HashSet](https://www.geeksforgeeks.org/hashset-in-java/) and [HashMap](https://www.cdn.geeksforgeeks.org/hashmap-treemap-java/) instead.

1. **TreeSet in Java**
2. TreeSet provides an implementation of the SortedSet Interface and SortedSet extends Set Interface.
3. TreeSet uses tree data structure for storage.
4. Objects are stored in sorted, ascending order. But we can iterate in descending order using method TreeSet.descendingIterator().
5. Access and retrieval times are very fast which make TreeSet an excellent choice for storage of large volume of data in sorted format.
6. TreeSet doesn’t use hashCode() and equals() methods to compare it’s elements. It uses compare() (or compareTo()) method to determine the equality of two elements.
7. The implementation of a TreeSet is not synchronized.

**Internal Working of TreeSet in Java:**

TreeSet is basically an implementation of a self-balancing binary search tree like a [Red-Black Tree](https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/). Therefore operations like add, remove, and search takes O(log(N)) time. The reason is that in a self-balancing tree, it is made sure that the height of the tree is always O(log(N)) for all the operations. Therefore, this is considered as one of the most efficient data structures in order to store the huge sorted data and perform operations on it. However, operations like printing N elements in the sorted order take O(N) time.

**Constructors of TreeSet Class:**

**1. TreeSet():** This constructor is used to build an empty TreeSet object in which elements will get stored in default natural sorting order.

TreeSet ts = new TreeSet();

**2. TreeSet(Comparator):**This constructor is used to build an empty TreeSet object in which elements will need an external specification of the sorting order.

TreeSet ts = new TreeSet(Comparator comp);

**3. TreeSet(Collection):** This constructor is used to build a TreeSet object containing all the elements from the given collection in which elements will get stored in default natural sorting order.

TreeSet t = new TreeSet(Collection col);

1. **TreeSet(SortedSet):** This constructor is used to build a TreeSet object containing all the elements from the given [sortedset](https://www.geeksforgeeks.org/sortedset-java-examples/) in which elements will get stored in default natural sorting order.

TreeSet t = new TreeSet(SortedSet s);

**Important Methods of TreeSet class:**

TreeSet implements [SortedSet](https://www.geeksforgeeks.org/sortedset-java-examples/) so it has the availability of all methods in Collection, [Set](https://www.geeksforgeeks.org/set-in-java/), and [SortedSet interfaces](https://www.geeksforgeeks.org/sortedset-java-examples/).

| **Method** | **Description** |
| --- | --- |
| [ceiling(E e)](https://www.geeksforgeeks.org/treeset-ceiling-method-in-java-with-examples/) | This method returns the least element in this set greater than or equal to the given element, or null if there is no such element. |
| [clone()](https://www.geeksforgeeks.org/treeset-clone-method-in-java/) | Returns a shallow copy of the set, which is just a simple copied set. |
| [Comparator comparator()](https://www.geeksforgeeks.org/treeset-comparator-method-in-java/) | This method will return the Comparator used to sort elements in TreeSet or it will return null if the default natural sorting order is used. |
| [descendingIterator()](https://www.geeksforgeeks.org/treeset-descendingiterator-method-in-java-with-examples/) | This method returns an iterator over the elements in this set in descending order. |
| [descendingSet()](https://www.geeksforgeeks.org/treeset-descendingset-method-in-java-with-examples/) | This method returns a reverse order view of the elements contained in this set. |
| [first()](https://www.geeksforgeeks.org/treeset-first-method-in-java/) | Returns the first element in TreeSet if TreeSet is not null else it will throw NoSuchElementException. |
| [floor(E e)](https://www.geeksforgeeks.org/treeset-floor-method-in-java-with-examples/) | This method returns the greatest element in this set less than or equal to the given element, or null if there is no such element. |
| [headSet(Object toElement)](https://www.geeksforgeeks.org/treeset-headset-method-in-java/) | Returns elements of TreeSet which are less than the specified element. |
| [higher(E e)](https://www.geeksforgeeks.org/treeset-higher-method-in-java-with-examples/) | Returns the least element in this set strictly greater than the given element, or null if there is no such element. |
| [last()](https://www.geeksforgeeks.org/treeset-last-method-in-java/) | Returns the last element in TreeSet if TreeSet is not null else it will throw NoSuchElementException. |
| [lower(E e)](https://www.geeksforgeeks.org/treeset-lower-method-in-java/) | Returns the greatest element in this set strictly less than the given element, or null if there is no such element. |
| [pollFirst()](https://www.geeksforgeeks.org/treeset-pollfirst-method-in-java/) | Retrieves and removes the first (lowest) element, or returns null if this set is empty. |
| [pollLast()](https://www.geeksforgeeks.org/treeset-polllast-method-in-java-with-example/) | Retrieves and removes the last (highest) element, or returns null if this set is empty. |
| [subSet(Object fromElement, Object toElement)](https://www.geeksforgeeks.org/treeset-subset-method-in-java/) | This method will return elements ranging from fromElement to toElement. fromElement is inclusive and toElement is exclusive. |
| [tailSet(Object fromElement)](https://www.geeksforgeeks.org/treeset-tailset-method-in-java/) | This method will return elements of TreeSet which are greater than or equal to the specified element. |

**Example:**

TreeSet<Integer> ts = new TreeSet<>();

ts.add(5);

ts.add(45);

ts.add(35);

ts.add(25);

System.out.println(ts);

// to check if treeset is empty or not.

if (ts.isEmpty())

System.out.print("Tree Set is empty.");

else

System.out.println("Tree Set size: " + ts.size());

// To get the smallest element from the set

System.out.println("First data: " + ts.first());

// To get the largest value from set

System.out.println("Last data: " + ts.last());

// remove 25from set.

ts.remove(25)

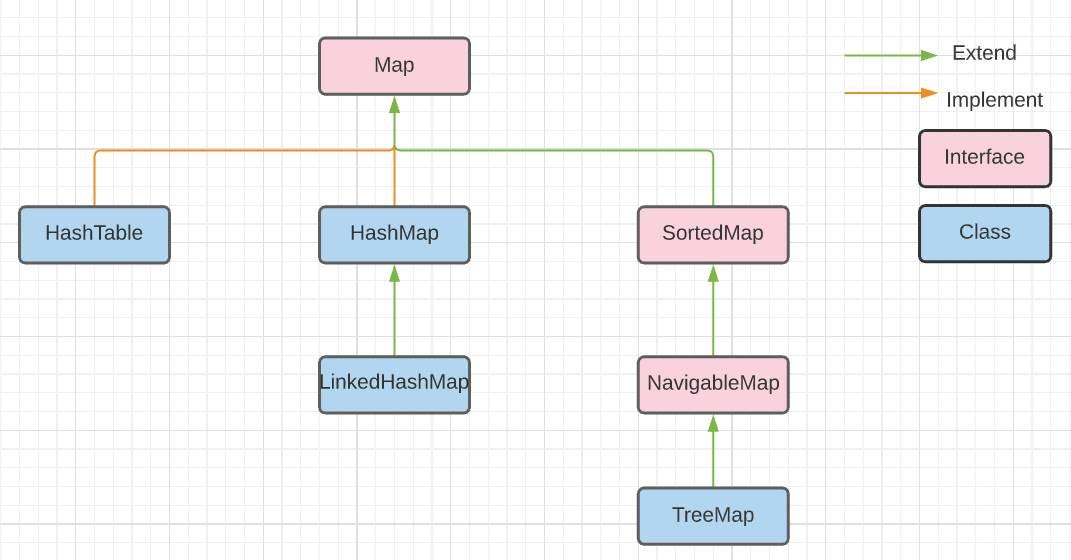
System.out.println("Tree set after removing 25”+ts);

**Note:**

* An object is said to be comparable if and only if the corresponding class implements a Comparable interface.
* [String](https://www.geeksforgeeks.org/string-class-in-java/), [StringBuffer](https://www.geeksforgeeks.org/stringbuffer-class-in-java/) class and all the Wrapper classes already implements Comparable interface Hence, we DO NOT get a ClassCastException. But if we are creating TreeSet of user defined classes or any Java classes which does not implements comparable interface we will get ClassCastException. to solve this problem we can either implement Comparable to our user defined class or we can pass Comparator object in Constructor while creating the set.

**Map Interface in Java**

* The map interface is present in [java.util](https://www.geeksforgeeks.org/java-util-package-java/) package represents a mapping between a key and a value.
* The Map interface is not a subtype of the [Collection interface](https://www.geeksforgeeks.org/collections-in-java-2/). Therefore it behaves a bit differently from the rest of the collection types.
* A map contains unique keys.



**Characteristics of a Map Interface:**

* A Map cannot contain duplicate keys and each key can map to at most one value. Some implementations allow null key and null values like the [HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java/) and  [LinkedHashMap](https://www.geeksforgeeks.org/linkedhashmap-class-java-examples/), but some do not like the [TreeMap](https://www.geeksforgeeks.org/treemap-in-java/).
* The order of a map depends on the specific implementations. For example,  [TreeMap](https://www.geeksforgeeks.org/treemap-in-java/) and [LinkedHashMap](https://www.geeksforgeeks.org/linkedhashmap-class-java-examples/) have predictable orders, while [HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java/) does not.

**Methods in Map Interface:**

| **Method** | **Description** |
| --- | --- |
| [clear()](https://www.geeksforgeeks.org/map-clear-method-in-java-with-example/) | clear and remove all of the elements or mappings from a specified Map collection. |
| [compute(K key, BiFunction remappingFunction)](https://www.geeksforgeeks.org/hashtable-compute-method-in-java-with-examples/) | Attempts to compute a mapping for the specified key and its current mapped value (or null if there is no current mapping). |
| [computeIfAbsent(K key, Function](https://www.geeksforgeeks.org/hashmap-computeifabsent-method-in-java-with-examples/) mappingFunction) | If the specified key is not already associated with a value (or is mapped to null), attempts to compute its value using the given mapping function and enters it into this map unless null. |
| [computeIfPresent(K key, BiFunction remappingFunction)](https://www.geeksforgeeks.org/hashmap-computeifpresentkey-bifunction-method-in-java-with-examples/) | If the value for the specified key is present and non-null, attempts to compute a new mapping given the key and its current mapped value. |
| [containsKey(key)](https://www.geeksforgeeks.org/map-containskey-method-in-java-with-examples/) | Used to check whether a particular key is being mapped into the Map or not. It takes the key element as a parameter and returns True if that element is mapped in the map. |
| [containsValue(value)](https://www.geeksforgeeks.org/map-containsvalue-method-in-java-with-examples/) | Used to check whether a particular value is being mapped by a single or more than one key in the Map. It takes the value as a parameter and returns True if that value is mapped by any of the key in the map. |
| [entrySet()](https://www.geeksforgeeks.org/map-entryset-method-in-java-with-examples/) | This method is used to create a set out of the same elements contained in the map. It basically returns a set view of the map or we can create a new set and store the map elements into them. |
| [equals(Object)](https://www.geeksforgeeks.org/map-equals-method-in-java-with-examples/) | This method is used to check for equality between two maps. It verifies whether the elements of one map passed as a parameter is equal to the elements of this map or not. |
| [get(key)](https://www.geeksforgeeks.org/map-get-method-in-java-with-examples/) | This method is used to retrieve or fetch the value mapped by a particular key mentioned in the parameter. It returns NULL when the map contains no such mapping for the key. |
| [hashCode()](https://www.geeksforgeeks.org/map-hashcode-method-in-java-with-examples/) | Generates a hashCode for the given map containing keys and values. |
| [isEmpty()](https://www.geeksforgeeks.org/map-isempty-method-in-java-with-examples/) | Checks if a map is having any entry for key and value pairs. If no mapping exists, then this returns true. |
| [keySet()](https://www.geeksforgeeks.org/map-keyset-method-in-java-with-examples/) | Returns a Set view of the keys contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. |
| [put(key, value)](https://www.geeksforgeeks.org/map-put-method-in-java-with-examples/) | This method is used to associate the specified value with the specified key in this map. |
| [putAll(Map)](https://www.geeksforgeeks.org/map-putall-method-in-java-with-examples/) | This method is used to copy all of the mappings from the specified map to this map. |
| [remove(key)](https://www.geeksforgeeks.org/map-remove-method-in-java-with-examples/) | This method is used to remove the mapping for a key from this map if it is present in the map. |
| [size()](https://www.geeksforgeeks.org/hashmap-size-method-in-java/) | Returns the number of key/value pairs available in the map. |
| [values()](https://www.geeksforgeeks.org/hashmap-values-method-in-java/) | Returns a Collection view of the values in the Map. |
| [getOrDefault(Object key, V defaultValue)](https://www.geeksforgeeks.org/hashmap-getordefaultkey-defaultvalue-method-in-java-with-examples/) | Returns the value to which the specified key is mapped, or defaultValue if this map contains no mapping for the key. |
| [merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction)](https://www.geeksforgeeks.org/hashmap-mergekey-value-bifunction-method-in-java-with-examples/) | If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value. |
| [putIfAbsent(K key, V value)](https://www.geeksforgeeks.org/hashmap-putifabsentkey-value-method-in-java-with-examples/) | If the specified key is not already associated with a value (or is mapped to null) associates it with the given value and returns null, else returns the current associate value. |

**Example:**

import java.util.\*;

public class HashMapExample {

public static void main(String[] args) {

Map<Integer, String> map1 = new HashMap<>();

map1.put(1, "Mango");

map1.put(2, "Apple");

map1.put(3, "Grapes");

System.out.println("HashMap1: ");

for(Map.Entry<Integer, String> ele : map1.entrySet()) {

System.out.println(ele.getKey()+": "+ele.getValue());

}

HashMap<Integer, String> map2 = new HashMap<>();

map2.put(1, "Yellow");

map2.put(3, "Red");

map2.put(5, "Green");

System.out.println("HashMap2: " + map2.toString());

//Merging two maps into map1

map2.forEach(

(key, value) -> map1.merge(key, value,

(v1, v2) -> v1.equalsIgnoreCase(v2) ?

v1 : v1+","+v2 ));

// print new mapping

System.out.println("New HashMap: " + map1);

}

}

**1. HashMap**

* [HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java-with-examples/) is a part of Java’s collection framework (java.util.HashMap).
* It provides the basic implementation of the Map interface of Java. It stores the data in (Key, Value) pairs. To access a value one must know its key.
* This class uses a technique called [Hashing](https://www.geeksforgeeks.org/hashing-data-structure/). Hashing is a technique of converting a large String to a small String that represents the same String. A shorter value helps in indexing and faster searches.
* HashMap does not maintain any order neither based on key nor on basis of value.
* Java HashMap is similar to [HashTable](https://www.geeksforgeeks.org/hashtable-in-java/), but it is unsynchronized.
* It may have one null key and multiple null values.

**Application:**  
HashMap is basically an implementation of [hashing](http://geeksquiz.com/hashing-set-1-introduction/). So wherever we need hashing with key value pairs, we can use HashMap. For example, in Web Applications username is stored as a key and user data is stored as a value in the HashMap, for faster retrieval of user data corresponding to a username.

**Note:** Keys and value can’t be primitive datatype. Key in Hashmap is valid if it implements [hashCode() and equals() method](https://www.geeksforgeeks.org/internal-working-of-hashmap-java/) , it should also be immutable (immutable custom object ) so that hashcode and equality remains constant. Value in hashmap can be any wrapper class, custom objects, arrays, any reference type or even null .

**Java HashMap Constructors**

1. HashMap()
2. HashMap(int initialCapacity)
3. HashMap(int initialCapacity, float loadFactor)
4. HashMap(Map map)

load factor = number of elements in the table / size of the hash table

**2. LinkedHashMap**

* [LinkedHashMap](https://www.geeksforgeeks.org/linkedhashmap-class-java-examples/) is just like HashMap with the additional feature of maintaining an order of elements inserted into it.
* A LinkedHashMap contains values based on the key.
* It implements the Map interface and extends the HashMap class.
* It may have one null key and multiple null values.
* It is non-synchronized.
* In this class, the data is stored in the form of nodes. The implementation of the LinkedHashMap is very similar to a [doubly-linked list](https://www.geeksforgeeks.org/doubly-linked-list/).

**Declaration:**

public class LinkedHashMap extends HashMap implements Map

**Constructors of LinkedHashMap Class:**

1. **LinkedHashMap():**

LinkedHashMap<K, V> lhm = new LinkedHashMap<K, V>();

1. **LinkedHashMap(int capacity):** It is used to initialize a particular LinkedHashMap with a specified capacity.

LinkedHashMap<K, V> lhm = new LinkedHashMap<K, V>(int capacity);

1. **LinkedHashMap(Map<K, V> map):** It is used to initialize a particular LinkedHash-Map with the elements of the specified map.

LinkedHashMap<K, V> lhm = new LinkedHashMap<K, V>(Map<K, V> map);

1. **LinkedHashMap(int capacity, float loadFactor):** It is used to initialize both the capacity and **loadFactor** for a LinkedHashMap. A loadFactor is a metric that determines when to increase the size of the LinkedHashMap automatically. By default, this value is 0.75 which means that the size of the map is increased when the map is 75% full.

LinkedHashMap<K, V> lhm = new LinkedHashMap<K, V>(int capacity, float fillRatio);

**LinkedHashMap(int capacity, float fillRatio, boolean Order):** For the **Order** attribute, true is passed for the last access order and false is passed for the insertion order.

LinkedHashMap<K, V> lhm = new LinkedHashMap<K, V>(int capacity, float fillRatio, boolean Order);

**boolean removeEldestEntry(Map.Entry *eldest*) method:**

It is used keep a track of whether the map removes any eldest entry from the map. So, each time a new element is added to the LinkedHashMap, the eldest entry is removed from the map.

This method is generally invoked after the addition of the elements into the map by the use of put() and putall() method.

LinkedHashMap<Integer, String> li\_hash\_map =

**new** LinkedHashMap<Integer, String>(MAX, 0.75, true) {

**protected** **boolean** removeEldestEntry(Map.Entry<Integer, String> eldest)

{ return size() > MAX; }

};

**3. TreeMap**

* TreeMap implements SortedMap interface. A Sorted Map interface is a child of Map Inteface.
* TreeMap is a member of the [Java Collections](https://www.geeksforgeeks.org/collections-in-java-2/) Framework.
* This map is sorted according to the natural ordering of its keys, or by a Comparator provided at map creation time, depending on which constructor is used. This proves to be an efficient way of sorting and storing the key-value pairs.
* The storing order maintained by the TreeMap must be consistent with equals just like any other sorted map, irrespective of the explicit comparators.
* Java.util.TreeMap uses a [**red-black tree**](https://www.cdn.geeksforgeeks.org/red-black-tree-set-1-introduction-2/)**(a Self Balancing Binary Search Tree)** in the background which makes sure that there are no duplicates; additionally it also maintains the elements in a sorted order.
* TreeMap in Java does not allow null keys (like Map) and thus a  [NullPointerException](https://www.geeksforgeeks.org/null-pointer-exception-in-java/) is thrown. However, multiple null values can be associated with different keys.
* The keys in a SortedMap (or TreeMap) must implement the java.lang.Comparable interface or a custom Comparator must be provided.

**Creating a Object:**

SortedMap<K, V> tmap = new TreeMap<K, V>();

**Constructors in TreeMap:**

1. **TreeMap():** This constructor is used to build an empty TreeMap that will be sorted by using the natural order of its keys.
2. **TreeMap(Comparator comp):** This constructor is used to build an empty TreeMap object in which the elements will need an external specification of the sorting order.
3. **TreeMap(Map M):** This constructor is used to initialize a TreeMap with the entries from the given map M which will be sorted by using the natural order of the keys.
4. **TreeMap(SortedMap sm):** This constructor is used to initialize a TreeMap with the entries from the given [sorted map](https://www.geeksforgeeks.org/sortedmap-java-examples/) which will be stored in the same order as the given sorted map.

**Methods in TreeMap** (except methods of Map Interface):

| **Method** | **Description** |
| --- | --- |
| [comparator()](https://www.geeksforgeeks.org/sortedmap-comparator-method-in-java-with-examples/) | Returns the comparator used to order the keys in this map, or null if this map uses the natural ordering of its keys. |
| [clone()](https://www.geeksforgeeks.org/treemap-clone-method-in-java/) | The method returns a shallow copy of this TreeMap. |
| [firstKey()](https://www.geeksforgeeks.org/sortedmap-firstkey-method-in-java/) | Returns the first (lowest) key currently in this map. |
| [headMap(K toKey)](https://www.geeksforgeeks.org/sortedmap-headmap-method-in-java/) | Returns a view of the portion of this map whose keys are strictly less than toKey. |
| [lastKey()](https://www.geeksforgeeks.org/sortedmap-lastkey-method-in-java/) | Returns the last (highest) key currently in this map. |
| [subMap(K fromKey, K toKey)](https://www.geeksforgeeks.org/sortedmap-submap-method-in-java/) | Returns a view of the portion of this map whose keys range from fromKey, inclusive, to toKey, exclusive. |
| [tailMap(K fromKey)](https://www.geeksforgeeks.org/sortedmap-tailmap-method-in-java/) | Returns a view of the portion of this map whose keys are greater than or equal to fromKey. |

1. **Hashtable in Java**

* It is similar to HashMap, but is synchronized -> poor in performance.
* Hashtable stores key/value pair in hash table.
* In Hashtable we specify an object that is used as a key, and the value we want to associate to that key. The key is then hashed, and the resulting hash code is used as the index at which the value is stored within the table.
* HashMap doesn’t provide any Enumeration, while Hashtable provides not fail-fast Enumeration.
* In order to create a Hashtable, we need to import it from java.util.Hashtable.

**Constructors of Hashtable:**

It has same constuctors as of the HashMap.

1. **Hashtable():** This creates an empty hashtable with the default load factor of 0.75 and an initial capacity is 11.

Hashtable<K, V> ht = new Hashtable<K, V>();

1. **Hashtable(int initialCapacity):** This creates a hash table that has an initial size specified by initialCapacity and the default load factor is 0.75.

Hashtable<K, V> ht = new Hashtable<K, V>(int initialCapacity);

1. **Hashtable(int size, float fillRatio):** This version creates a hash table that has an initial size specified by size and fill ratio specified by fillRatio. fill ratio: Basically, it determines how full a hash table can be before it is resized upward and its Value lies between 0.0 to 1.0.

Hashtable<K, V> ht = new Hashtable<K, V>(int size, float fillRatio);

1. **Hashtable(Map map):** This creates a hash table that is initialized with the elements in m.

Hashtable<K, V> ht = new Hashtable<K, V>(Map m);

**Methods of Hashtable** (except those of Map Interface)**:**

K – The type of the keys in the map.

V – The type of values mapped in the map.

| **Method** | **Description** |
| --- | --- |
| [clone()](https://www.geeksforgeeks.org/hashtable-clone-method-in-java/) | Creates a shallow copy of this hashtable. |
| [contains(Object value)](https://www.geeksforgeeks.org/hashtable-contains-method-in-java/) | Tests if some key maps into the specified value in this hashtable. |
| [elements()](https://www.geeksforgeeks.org/hashtable-elements-method-in-java/) | Returns an enumeration of the values in this hashtable. |
| [isEmpty()](https://www.geeksforgeeks.org/hashtable-isempty-method-in-java/) | Tests if this hashtable maps no keys to values. |
| [keys()](https://www.geeksforgeeks.org/hashtable-keys-method-in-java/) | Returns an enumeration of the keys in this hashtable. |
| rehash() | Increases the capacity of and internally reorganizes this hashtable, in order to accommodate and access its entries more efficiently. |
| [size()](https://www.geeksforgeeks.org/hashtable-size-method-in-java/) | Returns the number of keys in this hashtable. |
| [toString()](https://www.geeksforgeeks.org/hashtable-tostring-method-in-java/) | Returns a string representation of this Hashtable object in the form of a set of entries, enclosed in braces and separated by the ASCII characters “, ” (comma and space). |

**Example:**

import java.util.\*;

class AddElementsToHashtable {

public static void main(String args[])

{

// Initialization of a Hashtable using Generics

Hashtable<Integer, String> ht1

= new Hashtable<>(4, 0.75f); //initial capacity =4 and fillRatio = 0.75

// Inserting the Elements using put() method

ht1.put(1, "one");

ht1.put(2, "two");

ht1.put(3, "three");

// Print mappings to the console

System.out.println("Mappings of ht1 : " + ht1);

}

}

**Calendar Class in Java**

Calendar class in Java is an abstract class that provides methods for converting date between a specific instant in time and a set of calendar fields such as MONTH, YEAR, HOUR, etc. It inherits Object class and implements the Comparable, Serializable, Cloneable interfaces.  
  
As it is an Abstract class, so we cannot use a constructor to create an instance. Instead, we will have to use the static method Calendar.getInstance() to instantiate and implement a sub-class.

1. Calendar.getInstance(): return a Calendar instance based on the current time in the default time zone with the default locale.
2. Calendar.getInstance(TimeZone zone)
3. Calendar.getInstance(Locale aLocale)
4. Calendar.getInstance(TimeZone zone, Locale aLocale)

**Important Methods of Calendar class:**

| **Method** | **Description** |
| --- | --- |
| abstract void add(int field, int amount) | It is used to add or subtract the specified amount of time to the given calendar field, based on the calendar's rules. |
| int get(int field) | Returns the value of the given calendar field. |
| abstract int getMaximum(int field) | Returns the maximum value for the given calendar field of this Calendar instance. |
| abstract int getMinimum(int field) | Returns the minimum value for the given calendar field of this Calendar instance. |
| Date getTime() | Returns a Date object representing this Calendar's time value. |

**EXAMPLE:**

import java.util.Calendar;

import java.util.Locale;

import java.util.TimeZone;

public class CalendarClassExample {

public static void main(String[] args) {

Calendar c = Calendar.getInstance();

System.out.println("Current Date and time: "+c.getTime());

// displaying the date, time, time zone and locale using get method

System.out.print("Date: "

+ c.get(Calendar.DATE) + "/ "

+ c.get(Calendar.MONTH) + "/"

+ c.get(Calendar.YEAR) + "\n");

int max = c.getMaximum(Calendar.DAY\_OF\_WEEK);

System.out.println("Maximum number of days in a week: " + max);

max = c.getMaximum(Calendar.WEEK\_OF\_YEAR);

System.out.println("Maximum number of weeks in a year: " + max);

int min = c.getMinimum(Calendar.DAY\_OF\_WEEK);

System.out.println("Minimum number of days in week: " + min);

min = c.getMinimum(Calendar.WEEK\_OF\_YEAR);

System.out.println("Minimum number of weeks in year: " + min);

c.add(Calendar.DATE, -15);

System.out.println("15 days ago: " + c.getTime());

c.add(Calendar.MONTH, 4);

System.out.println("4 months later: " + c.getTime());

c.add(Calendar.YEAR, 2);

System.out.println("2 years later: " + c.getTime());

}

}

**Java.util.GregorianCalendar Class:**

GregorianCalendar is a concrete subclass of a Calendar that implements the most widely used Gregorian Calendar.

The major difference between **GregorianCalendar**and **Calendar**classes are that the Calendar Class being an abstract class cannot be instantiated. Whereas GregorianCalendar Class is initialized with the current date and time in the default locale and timezone.

GregorianCalendar Class defines two fields:

1. **AD** : referring to the common era(anno Domini)
2. **BC** : referring to before common era(Before Christ)

**Constructors**:

1. **GregorianCalendar():** initializes the object with the current date and time in the default locale and time zone.
2. **GregorianCalendar(int year, int month, int dayOfMonth):** initializes the object with the date-set passed as parameters in the default locale and time zone.
3. **GregorianCalendar(int year, int month, int dayOfMonth, int hours, int minutes):** initializes the object with the date and time-set passed as parameters in the default locale and time zone.
4. **GregorianCalendar(int year, int month, int dayOfMonth, int hours, int minutes, int seconds):** initializes the object with the date and more specific time-set passed as parameters in the default locale and time zone.
5. **GregorianCalendar(TimeZone timeZone):** initializes the object with the current date and time in the default locale and the time zone passed as parameters.
6. **GregorianCalendar(TimeZone timeZone, Locale locale)**: initializes the object with the current date and time in the locale and the time zone passed as parameters.

**Exception Handling in Java:**

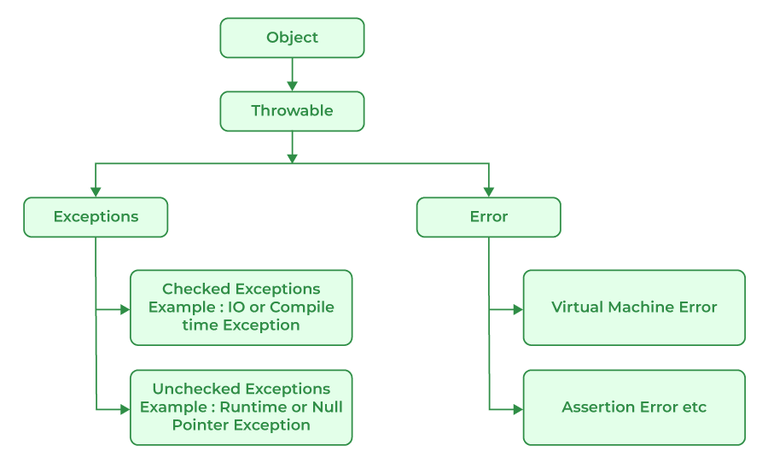
It is one of the effective means to handle runtime errors so that the regular flow of the application can be preserved.

**Exception:**

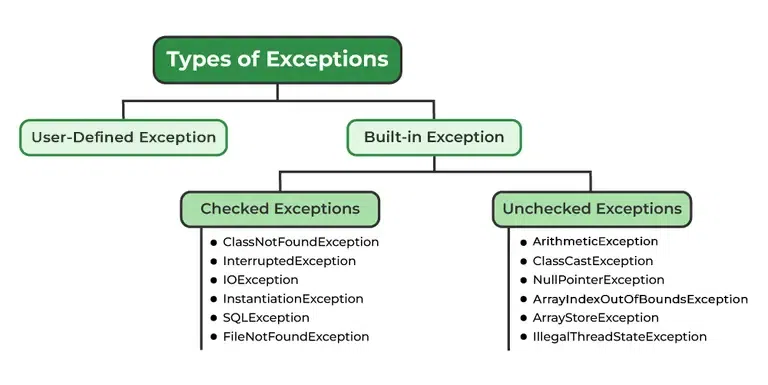
* Exception is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program’s instructions.
* Exceptions can be caught and handled by the program.
* When an exception occurs within a method, it creates an object. This object is called the exception object. It contains information about the exception, such as the name and description of the exception and the state of the program when the exception occurred.

**Errors:**  represent irrecoverable conditions such as Java virtual machine (JVM) running out of memory, memory leaks, stack overflow errors, library incompatibility, infinite recursion, etc. Errors are usually beyond the control of the programmer, and we should not try to handle errors.

**Exception Hierarchy:**



**Types of Exceptions**



1. **Built-in Exceptions**

Built-in exceptions are the exceptions that are available in Java libraries. These exceptions are suitable to explain certain error situations.

1. **Checked Exceptions:** Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using the [throws keyword](https://www.geeksforgeeks.org/throw-throws-java/).
2. **Unchecked Exceptions:** These are the exceptions that are not checked at compile time. In Java, exceptions under Error and RuntimeException classes are unchecked exceptions, everything else under throwable is checked.
3. **User-Defined Exceptions:**

Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, users can also create exceptions, which are called ‘user-defined Exceptions’ or ‘custom exception’.  In order to create a custom exception, we need to extend the Exception class that belongs to java.lang package.

**Methods to print the Exception information:**

1. **printStackTrace():** This method prints exception information in the format of the Name of the exception: description followed by stack trace.

**Example:**

//program to print the exception information using printStackTrace() method

import java.io.\*;

class GFG {

public static void main (String[] args) {

int a=5;

int b=0;

try{

System.out.println(a/b);

}

catch(ArithmeticException e){

e.printStackTrace();

}

}

}

Output:

java.lang.ArithmeticException: / by zero  
at GFG.main(File.java:10)

1. **toString():** The toString() method returns exception information in the format of the Name of the exception: description of the exception.

**Example:**

//program to print the exception information using toString() method

import java.io.\*;

class GFG1 {

public static void main (String[] args) {

int a=5;

int b=0;

try{

System.out.println(a/b);

}

catch(ArithmeticException e){

System.out.println(e.toString());

}

}

}

Output:

java.lang.ArithmeticException: / by zero

1. **getMessage() :** The getMessage() method returns only the description of the exception.

Output:

/ by zero

**How Programmer Handle an Exception?**

Java exception handling is managed via five keywords: try, catch, [throw](https://www.geeksforgeeks.org/throw-throws-java/), [throws](https://www.geeksforgeeks.org/throw-throws-java/), and finally.

1. **try {}:**

The [try](https://www.geeksforgeeks.org/flow-control-in-try-catch-finally-in-java/) block contains a set of statements where an exception can occur.

1. **catch() {}:**

If an exception occurs within the try block, it is thrown. So, to handle this exception in some rational manner, we need to catch this exception (using catch block).

1. **throw:**

System-generated exceptions are automatically thrown by the Java run-time system. To manually throw an exception, use the keyword throw. The throw keyword transfers control from the try block to the catch block.

1. **throws:**

The [throws](https://www.geeksforgeeks.org/throw-throws-java/) keyword is used for exception handling without try & catch block. It specifies the exceptions that a method can throw to the caller and does not handle itself.

* The throws keyword is required only for checked exceptions and usage of the throws keyword for unchecked exceptions is meaningless.
* The throws keyword is required only to convince the compiler and usage of the throws keyword does not prevent abnormal termination of the program.
* With the help of the throws keyword, we can provide information to the caller of the method about the exception.

1. **finally {}:**

* Any code that absolutely must be executed after a try block completes is put in a finally block.
* The finally block is optional.
* It always gets executed whether an exception occurred in try block or not. If an exception occurs, then it will be executed after try and catch blocks. And if an exception does not occur, then it will be executed after the try block.
* The finally block in Java is used to put important codes such as clean-up code e.g., closing the file or closing the connection.
* If we write System.exit(0) in the try block, then finally block will not be executed.

**Note:**

* In a method, there can be more than one statement that might throw an exception, So put all these statements within their own try block and provide a separate catch block for each of them.
* Each catch block is an exception handler that handles the exception to the type indicated by its argument.
* From Java 7, we can catch more than one exception with single catch block.

catch(ArrayIndexOutOfBoundsException || ArithmeticException e) { }

* For each try block, there can be zero or more catch blocks, but only one finally block.

**EXAMPLE:**

long power(int n, int p) throws Exception{

long res = n;

if(n >= 0 && p >= 0){

if(n > 0 || p > 0){

if(p == 0){

return 1;

}

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

res = res\*n;

}

}

else{

throw new Exception("n and p should not be zero.");

}

}

else{

throw new Exception("n or p should not be negative.");

}

return res;

}

**EXAMPLE-2:**

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

try {

int x = sc.nextInt();

int y = sc.nextInt();

System.out.println(x/y);

}

catch(InputMismatchException i) {

i.printStackTrace();

}

catch(ArithmeticException e) {

System.out.println(e.toString());

}

finally {

sc.close();

}

}

**EXAMPLE-3:**

// Java program that demonstrates the use of throw

class ThrowExcep {

static void fun()

{

try {

throw new NullPointerException("demo");

}

catch (NullPointerException e) {

System.out.println("Caught inside fun().");

throw e; // rethrowing the exception

}

}

public static void main(String args[])

{

try {

fun();

}

catch (NullPointerException e) {

System.out.println("Caught in main.");

}

}

}

Output:

Caught inside fun().

Caught in main.

**Note:**

Java supports nested exception (Chained exception), allow you to associate a cause with an exception in Java.

**Regular Expressions in Java**

Regular Expressions or Regex (in short) in Java is an API for defining String patterns that can be used for searching, manipulating, and editing a string in Java

Regular Expressions in Java are provided under java.util.regex package. This consists of 3 classes and 1 interface.

1. [**Pattern Class**](https://www.geeksforgeeks.org/pattern-class-in-java/)**:** Used for defining patterns
2. [**Matcher Class**](https://www.geeksforgeeks.org/matcher-class-in-java/)**:** Used for performing match operations on text using patterns
3. [**PatternSyntaxException Class**](https://www.geeksforgeeks.org/patternsyntaxexception-class-in-java/)**:** Used for indicating syntax error in a regular expression pattern
4. [**MatchResult Interface**](https://www.geeksforgeeks.org/matchresult-interface-in-java/)**:** Used for representing the result of a match operation

**Pattern Class**

* used to define various types of patterns.
* Has no public constructors.
* Object can be created by invoking the compile() method which accepts a regular expression as the first argument, thus returning a pattern after execution.

| **S. No.** | **Method** | **Description** |
| --- | --- | --- |
| 1. | [**compile(String regex)**](https://www.geeksforgeeks.org/pattern-compilestring-method-in-java-with-examples/) | compile the given regular expression into a pattern. |
| 2. | [compile(String regex, int flags)](https://www.geeksforgeeks.org/pattern-compilestringint-method-in-java-with-examples/) | compile the given regular expression into a pattern with the given flags. |
| 3. | [flags()](https://www.geeksforgeeks.org/pattern-flags-method-in-java-with-examples/) | return this pattern’s match flags. |
| 4. | [matcher(CharSequence input)](https://www.geeksforgeeks.org/pattern-matchercharsequence-method-in-java-with-examples/) | create a matcher that will match the given input against this pattern. |
| 5. | [**matches(String regex, CharSequence input)**](https://www.geeksforgeeks.org/pattern-matchesstring-charsequence-method-in-java-with-examples/) | compile the given regular expression and attempts to match the given input against it. |
| 6. | [pattern()](https://www.geeksforgeeks.org/pattern-pattern-method-in-java-with-examples/) | return the regular expression from which this pattern was compiled. |
| 7. | [quote(String s)](https://www.geeksforgeeks.org/pattern-quotestring-method-in-java-with-examples/) | return a literal pattern String for the specified String. |
| 8. | [split(CharSequence input)](https://www.geeksforgeeks.org/pattern-splitcharsequence-method-in-java-with-examples/) | split the given input sequence around matches of this pattern. |
| 10. | [toString()](https://www.geeksforgeeks.org/pattern-tostring-method-in-java-with-examples/) | It is used to return the string representation of this pattern. |

**Note:** flags are a bit mask that may include CASE\_INSENSITIVE, MULTILINE, DOTALL, UNICODE\_CASE, CANON\_EQ, UNIX\_LINES, LITERAL, UNICODE\_CHARACTER\_CLASS and COMMENTS Flags.

**Matcher class:**

* Has no public constructors.
* Object can be implemented by invoking a matcher() on any pattern object.

| **S. No.** | **Method** | **Description** |
| --- | --- | --- |
| 1. | [find()](https://www.geeksforgeeks.org/matcher-find-method-in-java-with-examples/) | mainly used for searching multiple occurrences of the regular expressions in the text. |
| 2. | [find(int start)](https://www.geeksforgeeks.org/matcher-findint-method-in-java-with-examples/) | used for searching occurrences of the regular expressions in the text starting from the given index. |
| 3. | [start()](https://www.geeksforgeeks.org/matcher-start-method-in-java-with-examples/) | Returns the start index of a match that is being found using find() method. |
| 4. | [end()](https://www.geeksforgeeks.org/matcher-end-method-in-java-with-examples/) | Returns the end index of a match that is being found using find() method. It returns the index of the character next to the last matching character. |
| 5. | [groupCount()](https://www.geeksforgeeks.org/matcher-groupcount-method-in-java-with-examples/) | finds the total number of the matched subsequence. |
| 6. | [group()](https://www.geeksforgeeks.org/matcher-group-method-in-java-with-examples/) | Finds the matched subsequence. |
| 7. | [matches()](https://www.geeksforgeeks.org/matcher-matches-method-in-java-with-examples/) | checks if the whole text matches with a pattern or not. |

Example:

public static void main(String[] args) {

//pattern for any alphabetical word of length 3

Pattern p = Pattern.*compile*("[a-z]{3}", Pattern.***CASE\_INSENSITIVE***);

Matcher m = p.matcher("how ARE you");

while(m.find()) {

System.***out***.println(m.start()+"-"+m.end()); //Output : 0-3, 4-7, 8-11

}

System.***out***.println("Flag is "+ p.flags());

System.***out***.println(m.matches()); //false as the whole text does not matches with pattern.

System.***out***.println(m.groupCount()); //Output : 0

System.***out***.println(p.pattern()); //Output :[a-zA-Z]{3}

System.***out***.println(Pattern.*matches*("an[0-9]\*","an101")); ////Output : true

}

**Regex Expressions:**

| **Regex** | **Description** |
| --- | --- |
| 1. [xyz] | x,y or z |
| 1. [^xyz] | Any characters other than x,y or z |
| 1. [a-zA-Z] | characters from range a to z or A to Z. |
| 1. [a-f[m-t]] | Union of a to f and m to t. |
| 1. [a-z && p-y] | All the range of elements intersection between two ranges |
| 1. [a-z && [^bc]] | a to z union with except b and c |
| 1. [a-z && [^m-p]] | a to z union with except range m to p |
| 1. X? | X appears once or not |
| 1. X+ | X appears once or more than once |
| 1. X\* | X appears zero or more times (upto infinite). |
| 1. X{n} | X appears n times |
| 1. X{n,} | X appears n times or more than n |
| 1. X{n,m} | X appears greater than equal to n times and less than m times. |
| 1. . | Any character |
| 1. \d | Any digits, [0-9] |
| 1. \D | Any non-digit, [^0-9] |
| 1. \s | Whitespace character, [\t\n\x0B\f\r] |
| 1. \S | Non-whitespace character, [^\s] |
| 1. \w | Word character, [a-zA-Z\_0-9] |
| 1. \W | Non-word character, [^\w] |
| 1. \b | Word boundary |
| 1. \B | Non -Word boundary |

**Note:**

1. The caret ( **^** ) symbol: The match must start at the beginning of the string or line. Example : ^\d{3} will match with patterns like "901" in "901-333-".
2. The dollar ( **$** ) symbol: The match must occur at the end of the string or before \n at the end of the line or string.
3. The Escape Symbol ( **\**  ) : If you want to match for the actual ‘+’, ‘.’ etc characters, add a backslash( \ ) before that character. This will tell the computer to treat the following character as a search character and consider it for a matching pattern.
4. Grouping Characters **( )** : A set of different symbols of a regular expression can be grouped together to act as a single unit and behave as a block, for this, you need to wrap the regular expression in the parenthesis( ).
5. Vertical Bar ( **|**) : Matches any one element separated by the vertical bar (|) character. Example : th(e|is|at) will match words - the, this and that.
6. **\number :** allows a previously matched sub-expression(expression captured or enclosed within circular brackets ) to be identified subsequently in the same regular expression. \n means that group enclosed within the n-th bracket will be repeated at current position. Example : ([a-z])\1 will match “ee” in Geek because the character at second position is same as character at position 1 of the match.
7. Comment ( ?# comment ): Inline comment: The comment ends at the first closing parenthesis. Example : \bA(?#This is an inline comment)\w+\b

Example:

// Regex to remove leading zeros from a string

String regex = "^0+(?!$)";

**Garbage Collection in Java:**

* Java garbage collection is an **automatic** process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects.
* **System.gc()** and **Runtime.getRuntime().gc()**are the methods which requests for Garbage collection to JVM explicitly but it doesn’t ensures garbage collection as the final decision of garbage collection is of JVM only.

**Ways to make an object eligible for Garbage Collector:**

1. Nullifying the reference variable
2. Re-assigning the reference variable
3. An object created inside the method
4. Island of Isolation

**Finalization:**

* Just before destroying an object, Garbage Collector calls finalize() method on the object to perform cleanup activities. Once finalize() method completes, Garbage Collector destroys that object.
* The *finalize()*method is called by Garbage Collector, not JVM.
* Object class finalize() method has an empty implementation. Thus, it is recommended to override the finalize() method to dispose of system resources or perform other cleanups.

**Memory leaks in Java:**

* Java does automatic Garbage collection. However there can be situations where garbage collector does not collect objects because there are references to them. There might be situations where an application creates lots of objects and does not use them. Just because every objects has valid references, garbage collector in Java can’t destroys the objects. Such types of useless objects are called as Memory leaks.
* If allocated memory goes beyond limit, program will be terminated by rising OutOfMemoryError. Hence if an object is no longer required, it is highly recommended to make that object eligible for garbage collector. Otherwise We should use some tools that do memory management to identifies useless objects or memory leaks like:

1. HP OVO
2. HP J METER
3. JProbe
4. IBM Tivoli

**Memory Management:**

* In Java, all objects are dynamically allocated on Heap. This is different from C++ where objects can be allocated memory either on Stack or on Heap.
* In Java, when we only declare a variable of a class type, only a reference is created (memory is not allocated for the object). To allocate memory to an object, we must use new(). So the object is always allocated memory on the heap.

**Stack Memory:**

* Stack is a temporary contiguous memory allocation scheme where the data members are accessible only if the method( ) that contained them is currently running.
* It allocates or de-allocates the memory automatically as soon as the corresponding method completes its execution.

**Heap Allocation:**

* Heap is the memory block where objects are created or stored. Here no automatic de-allocation feature is provided. We need to use a Garbage collector to remove the old unused objects in order to use the memory efficiently.

**Stack vs Heap Memory Allocation**

* The processing time(Accessing time) of Heap memory is quite slow as compared to Stack-memory.
* Heap memory is also not as threaded-safe as Stack-memory because data stored in Heap-memory are visible to all threads.
* The size of the Heap-memory is quite larger as compared to the Stack-memory.
* Heap memory is accessible or exists as long as the whole application(or java program) runs.

**Multithreading in Java**

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread.

**Threads** are light-weight processes within a process.

* A thread consists of three parts Virtual Cpu, Code and data.
* At run time threads share code and data i.e they use same address space.
* Every thread in java is an object of java.lang.Thread class.

Threads can be created by using two mechanisms :

1. Extending the Thread class
2. Implementing the Runnable Interface

**1. Thread creation by extending the Thread class**  
We create a class that extends the **java.lang.Thread** class. This class overrides the run() method available in the Thread class. A thread begins its life inside run() method. We create an object of our new class and call start() method to start the execution of a thread. Start() invokes the run() method on the Thread object.

Example:

// Java code for thread creation by extending the Thread class

class MultithreadingDemo extends Thread {

public void run(){

try {

// Displaying the thread that is running

System.out.println(

"Thread " + Thread.currentThread().getId()

+ " is running");

}

catch (Exception e) {

// Throwing an exception

System.out.println("Exception is caught");

}

}

}

// Main Class

public class Multithread {

public static void main(String[] args){

int n = 5; // Number of threads

for (int i = 0; i < n; i++) {

MultithreadingDemo object

= new MultithreadingDemo();

object.start();

}

}

}

**Output:**

Thread 15 is running

Thread 14 is running

Thread 16 is running

Thread 12 is running

Thread 11 is running

**2. Thread creation by implementing the Runnable Interface**  
We create a new class which implements java.lang.Runnable interface and override run() method. Then we instantiate a Thread object and call start() method on this object.

**Example:**

// Java code for thread creation by implementing the Runnable Interface

class MultithreadingDemo implements Runnable {

public void run()

{

try {

// Displaying the thread that is running

System.out.println(

"Thread " + Thread.currentThread().getId()

+ " is running");

}

catch (Exception e) {

// Throwing an exception

System.out.println("Exception is caught");

}

}

}

// Main Class

class Multithread {

public static void main(String[] args){

for (int i = 0; i < 5; i++) {

Thread object = new Thread(new MultithreadingDemo());

object.start();

}

}

}

Output:

Thread 13 is running

Thread 11 is running

Thread 12 is running

Thread 15 is running

Thread 14 is running

Note: Make sure to create an object of threads in which you have to pass the object of runnable.

**Thread Class vs Runnable Interface**

1. If we extend the Thread class, our class cannot extend any other class because Java doesn’t support multiple inheritance. But, if we implement the Runnable interface, our class can still extend other base classes.
2. We can achieve basic functionality of a thread by extending Thread class because it provides some inbuilt methods like yield(), interrupt() etc. that are not available in Runnable interface.
3. Using runnable will give you an object that can be shared amongst multiple threads.
4. Runnable interface can’t throw checked exception but RuntimeException can be thrown from the run().

**States of a Thread in Java:**

A thread can exist in any of the five states at a time:

1) **New**: When the instance of thread is created it will be in New state.

Ex : Thread t= new Thread();

In the above example t is in new state. The thread is created but not in active state to make it active we need to call start() method on it.

2) **Runnable state**: A thread can be in the runnable state in either of the following two ways :

* 1. When the start method is invoked or
  2. A thread can also be in runnable state after coming back from blocked or sleeping or waiting state.

3) **Running state:** If thread scheduler allocates cpu time, then the thread will be in running state.

4) **Waited /Blocked/Sleeping state**: In this state the thread can be made temporarily inactive for a short period of time. A thread can be in the above state in any of the following ways:

* 1. The thread waits to acquire lock of an object.
  2. The thread waits for another thread to complete.
  3. The thread waits for notification of other thread.

5) **Dead/Terminated State:** A thread is in dead state when thread’s run method execution is complete. It dies automatically when thread’s run method execution is completed and the thread object will be garbage collected.

**Implementing the Thread States in Java:**

In Java, to get the current state of the thread, use **Thread.getState()** method to get the current state of the thread. Java provides **java.lang.Thread.State** class that defines the ENUM constants for the state of a thread.

**Note**:

1. A thread can be started in java using **start()** method in java. If we call start method second time once it is started it will cause RunTimeException (**IllegalThreadStateException**). A runnable thread cannot be restarted.
2. If we try to restart a dead thread by using start method we will get run time exception since the thread is not alive.
3. Usually threads are created to perform different unrelated tasks but there may be situations where they may perform related tasks. Interthread communication in java is done with the help of following three methods :
   1. **Wait()** : wait() method() makes the thread current thread sleeps and releases the lock until some other thread acquires the lock and calls notify().
   2. **notify()** :notify() method wakes up the thread that called wait on the same object.
   3. **notfiyAll()** :notifyAll() method wakes up all the threads that are called wait() on the same object. The highest priority threads will run first.

All the above three methods are in object class and are called only in synchronized context.

All the above three methods must handle **InterruptedException** by using throws clause or by using try catch clause.

1. A process is a program in execution. Every process have their own memory space. Process are heavy weight and requires their own address space. One or more threads make a process.

**Some methods of Thread class:**

| **Methods** | **Action Performed** |
| --- | --- |
| currentThread() | Returns a reference to the currently executing thread object. |
| getName() | Returns this thread’s name |
| getPriority() | Returns this thread’s priority |
| getState() | Returns the state of this thread |
| isDaemon() | It checks whether the current thread is daemon or not |
| start() | It creates a new thread and starts the execution of the thread. |
| run() | If this thread was constructed using a separate Runnable run object, then that Runnable object’s run method is called; otherwise, this method does nothing and returns. |
| sleep(long millis) | It is a static method that puts the thread to sleep for a certain time been passed as an argument to it. Static method. Throw a checked exception. |
| join() | Waits for this thread to die |
| setName(String name) | Changes the name of this thread to be equal to the argument name. |
| setPriority(int newPriority) | Changes the priority of this thread |
| setDaemon() | It set the current thread as Daemon thread |
| stop() | It is used to stop the execution of the thread |
| resume() | It is used to resume the suspended thread. |

**Priorities in Threads**

Priorities in threads is a concept where each thread is having a priority which is represented by numbers ranging from 1 to 10.

* The default priority is set to 5 as excepted.
* Minimum priority is set to 1.
* Maximum priority is set to 10.

Here 3 constants are defined in it namely as follows:

1. public static int NORM\_PRIORITY
2. public static int MIN\_PRIORITY
3. public static int MAX\_PRIORITY

If two threads have the same priority then we can’t expect which thread will execute first. It depends on the thread scheduler’s algorithm( Round-Robin, First Come First Serve, etc)

[**Daemon thread**](https://www.geeksforgeeks.org/daemon-thread-java)is basically a service provider thread that provides services to the user thread. The scope for this thread start() or be it terminate() is completely dependent on the user’s thread as it supports in the backend for user threads being getting run. As soon as the user thread is terminated daemon thread is also terminated at the same time as being the service provider thread.

Hence, the characteristics of the Daemon thread are as follows:

* It is only the service provider thread not responsible for interpretation in user threads.
* So, it is a low-priority thread.
* It is a dependent thread as it has no existence on its own.
* JVM terminates the thread as soon as user threads are terminated and come back into play as the user’s thread starts.
* Most popular example is garbage collector in java.

**Main Thread:**

For each program, a Main thread is created by JVM(Java Virtual Machine). The “Main” thread first verifies the existence of the main() method, and then it initializes the class.

* It is the thread from which other “child” threads will be spawned.
* Often, it must be the last thread to finish execution because it performs various shutdown actions.

Deadlocking with use of Main Thread(only single thread):

We can create a deadlock by just using the Main thread, i.e. by just using a single thread.

public class Example {

public static void main(String[] args) {

try {

System.out.println("Entering into Deadlock");

// Joining the current thread

Thread.currentThread().join();

System.out.println("This statement will never execute");

}

catch (InterruptedException e) {

e.printStackTrace();

}

}

}

The statement “Thread.currentThread().join()”, will tell Main thread to wait for this thread(i.e. wait for itself) to die. Thus Main thread wait for itself to die, which is nothing but a deadlock.

**Thread.start() vs Thread.run() Method:**

|  |  |
| --- | --- |
| **Thread.start()** | **Thread.run()** |
| Creates a new thread and the run() method is executed on the newly created thread. | No new thread is created and the run() method is executed on the calling thread itself. |
| It can’t be invoked more than one time otherwise throws java.lang.IllegalStateException. | Multiple invocations are possible. |
| Defined in java.lang.Thread class. | Defined in java.lang.Runnable interface and must be overridden in the implementing class. |

**Java Thread Safety:**

* When multiple threads are working and manipulating the same data, then it results into inconsistency. This scenario is not thread-safe.
* Thread-Safety means the resources are being accessed by one thread at a time.

There are four ways to achieve Thread Safety in Java. These are:

1. Using **Synchronization:** Synchronization is the process of allowing only one thread at a time to complete the particular task. It uses a *synchronized keyword*.
2. Using **Volatile** Keyword: It is a field modifier that ensures that the object can be used by multiple threads at the same time without having any problem.
3. Using **Atomic** Variable: When variables are shared by multiple threads, the atomic variable ensures that threads don’t crash into each other.
4. Using **Final** Keyword:

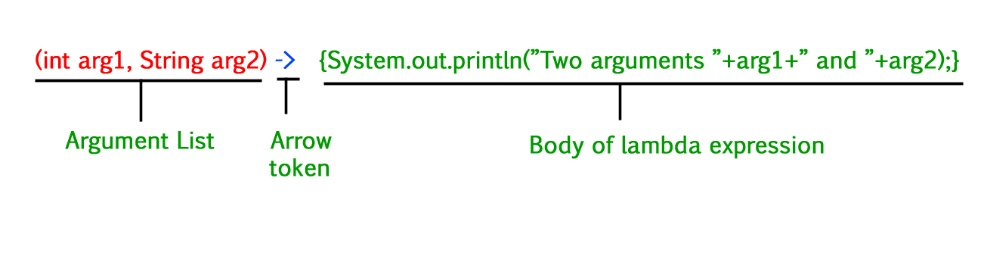
**Java 8 Features**

1. **Lambda Expressions:** Concise functional code using ->.
2. **Functional Interfaces:** Single-abstract method interfaces.
3. **Method References**: Refer to methods easily.
4. **Stream API**: Efficient Data Manipulation.
5. **Date/Time API**: Robust Date and Time Handling.
6. **Optional Class**: Handle null values safely.
7. Comparable and Comparator

**Lambda expressions in Java**

It represents the instances of **functional interfaces** (interfaces with a single abstract method).

Syntax:



Example:

interface FuncInterface

{

// An abstract function

void abstractFun(int x);

// A non-abstract (or default) function

default void normalFun()

{

System.out.println("Hello");

}

}

class Test

{

public static void main(String args[])

{

// lambda expression to implement above functional interface.

// This interface by default implements abstractFun()

FuncInterface fobj = (x)->System.out.println(2\*x);

// This calls above lambda expression and prints 10.

fobj.abstractFun(5);

}

}

**Common Built-in Functional Interfaces:**

* [**Comparable<T>**](https://www.geeksforgeeks.org/comparable-interface-in-java-with-examples/): int compareTo(T o);
* [**Comparator<T>**](https://www.geeksforgeeks.org/comparator-interface-java/): int compare(T o1, T o2);

These are commonly used in sorting and comparisons.

Note: Other commonly used interface include Predicate<T>, it is used to test conditions, Function<T, R>, it represent a function that take the argument of type T and return a result of type R and Supplier<T>, it represent a function that supplies results.

**Java Method References**

A **method reference in Java** is the shorthand syntax for a lambda expression to call a method. In general, one doesn’t have to pass arguments to method references. It was introduced in Java 8.

Key Benefits of Method References

* **Improved Readability:** Method references simplify the code by removing boilerplate syntax.
* **Reusability:** Existing methods can be directly reused, enhancing modularity.
* **Functional Programming Support:** They work seamlessly with functional interfaces and lambdas.

**Function as a Variable:**

In Java 8 we can use the method as if they were objects or primitive values, and we can treat them as a variable.

*// This square function is a variable getSquare.   
Function<Integer, Integer> getSquare = i -> i \* i ;*

*// Pass function as a argument to other function easily   
SomeFunction(a, b, getSquare) ;*

**Generic Syntax for Method References**

| **Aspect** | **Syntax** |
| --- | --- |
| Reference to a Static Method | ClassName::staticMethod |
| Refer to a method of a particular object | Object :: methodName |
| Reference to a method of an Arbitrary object of a Particular Type | ObjectType:: methodName;  Example: *String::compareToIgnoreCase* |
| Reference to a Constructor | ClassName::new; |
| Print all elements in a list | list.forEach(s -> System.out.println(s)); |
| Shorthand to print all elements in a list | list.forEach(System.out::println); |

import java.util.Arrays;

public class Test

{

public static void print(String s) {

System.out.println(s);

}

public static void main(String[] args)

{

String[] names = {"Alice", "Bob", "Charlie"};

Arrays.stream(names).forEach(Geeks::print);

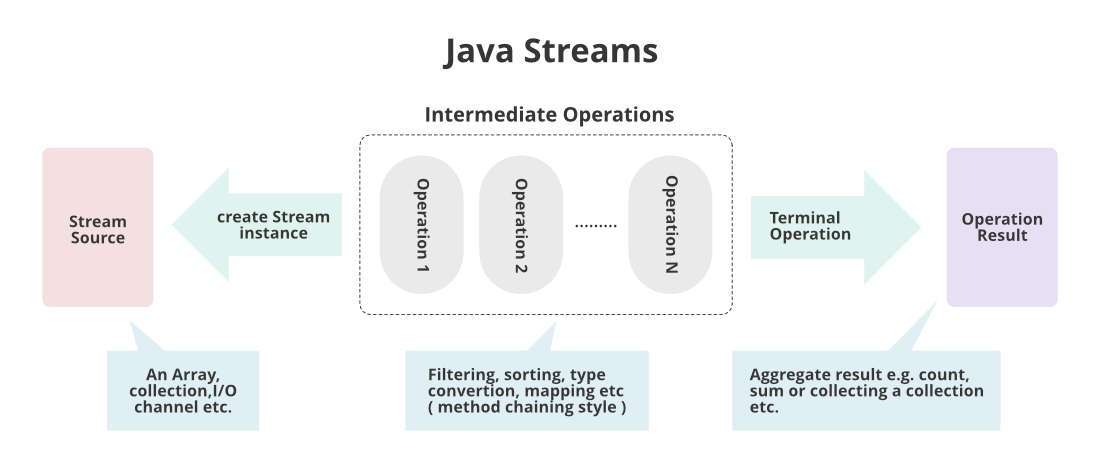
}

}

**Java 8 Stream**

* A Stream is a sequence of components that can be processed sequentially.
* It allows functional-style operations on the elements.
* The stream can be used by importing java.util.stream package.
* A stream is not a data structure instead it takes input from the Collections, Arrays, or I/O channels.
* Streams don’t change the original data structure, they only provide the result as per the pipelined methods.
* Each intermediate operation is lazily executed and returns a stream as a result, hence various intermediate operations can be pipelined. Terminal operations mark the end of the stream and return the result.

**How does Stream Work Internally?**



**Various Core Operations Over Streams:** There are basically 2 types of operations.

1. **Intermediate operations**: They transform a stream into another stream. Multiple intermediate operations can be chained together. They are lazy, meaning they are not executed until a terminal operation is invoked. They are the method of **Stream** interface.

Some common intermediate operations include:

* + 1. filter(): Filters elements based on a specified condition.

Syntax:

Stream<T> filter(Predicate<? super T> predicate)

Example:

List<Integer> list = Arrays.asList(10,5,4,6,55);

List<Integer> evenNo =  list.stream().filter(a->a%2==0).collect(Collectors.toList());

        evenNo.forEach(x->System.out.print(x+" "));

* + 1. map(): Transforms each element in a stream to another value.
    2. flatmap(): When handling nested structures such as lists of lists, flatMap simplifies the data into a single continuous stream.

Example:

List<Integer> flatSet = nestedList.stream().flatMap(Collection::stream).collect(Collectors.toList());

* + 1. distinct(): returns a stream consisting of distinct elements in a stream.
    2. sorted(): Used to sort elements in the stream, either in reverse order or in natural sorting order.
    3. peek(): The peek method allows developers to troubleshoot and visualize the stream data without altering the stream itself, acting as an excellent tool for debugging.
    4. limit(int n): Limits the number of elements to be returned.
    5. skip(int n): Skips first ‘n’ elements from the streams and returns remaining.

1. **Terminal operations:** Terminal Operations are the operations that on execution return a final result as an absolute value.
   1. forEach(): It iterates all the elements in a stream.
   2. collect(): It is used to return the result of the intermediate operations performed on the stream.
   3. reduce(): It is used to reduce the elements of a stream to a single value.

Example:

Optional<Integer> product = list.stream().reduce((a,b)->a+b);

* 1. max():

Example:

Optional<Integer> max = list.stream().max(Comparable::compareTo); // Optional[max]

* 1. min():
  2. toArray():

Example:

String[] nameArr = names.stream().toArray(String[]::new);

1. **Short-circuit operations**: Short-circuit operations provide performance benefits by avoiding unnecessary computations when the desired result can be obtained early.
   1. **allMatch():** is used to check if all elements in the stream satisfy a given predicate. It returns `true` if every element in the stream matches the predicate, and `false` otherwise.
   2. **anyMatch():** it checks whether at least one element in the stream matches a given predicate.

Example:

boolean isStartS = names.stream().anyMatch(name->name.startsWith("s"));

* 1. **noneMatch:** check if no elements in the stream match a given predicate.
  2. **findFirst():** It is used to retrieve the **first element** in a stream that matches a given condition or simply the first element in the stream if no filtering is applied. It returns the first element wrapped in an `Optional`, which is a container object that may or may not contain a non-null value.

Example: Optional<Integer> firstOdd = list.stream().filter(x->x%2!=0).findFirst();

* 1. **findAny:** It is used to retrieve a element in a stream that matched a given condition or simply the first element in the stream if no filtering is applied. It returns the element wrapped in an `Optional`.

1. **Collectors**: methods in the Collectors utility class.
   1. joining(): is used to concatenate the elements of a stream into a single string.

Syntax:

Collectors.joining(CharSequence delimiter, CharSequence prefix, CharSequence suffix)

Example:

String fullName = names.stream().collect(Collectors.joining(","));

* 1. groupBy:
  2. toList:
  3. toMap:
  4. partitioningBy
  5. counting:
  6. summerizingInt:
  7. averagingInt:
  8. mapping
  9. reducing:

**New Date-Time API in Java 8**

New date-time API is introduced in Java 8 to overcome the following drawbacks of old date-time API :

1. **Not thread safe :**Unlike old java.util.Date which is not thread safe the new date-time API is *immutable* and doesn’t have setter methods.
2. **Less operations :**In old API there are only few date operations but the new API provides us with many date operations.

Java 8 under the package **java.time** introduced a new date-time API, most important classes among them are :

1. **Local :**Simplified date-time API with no complexity of timezone handling.
2. **Zoned :**Specialized date-time API to deal with various timezones.

* **LocalDate/LocalTime**and **LocalDateTime API :**Use it when time zones are NOT required.

import java.time.\*;

import java.time.format.DateTimeFormatter;

public class LocalDateTimeExample {

static void LocalDataTimeApi(){

LocalDate date = LocalDate.now();

System.out.println("Current Date: "+date);

LocalTime time = LocalTime.now();

System.out.println("Current Time: "+time);

LocalDateTime dateTime = LocalDateTime.now();

System.out.println("Current Date and Time: "+dateTime);

DateTimeFormatter format = DateTimeFormatter.ofPattern("dd-MM-YYYY hh:mm:ss:a");

String formatedDateTime = dateTime.format(format);

System.out.println("Formated Date Time: "+formatedDateTime);

LocalDate date2 = LocalDate.of(1998, 10, 13);

System.out.println("Birthday: "+date2);

// printing date with current time.

LocalDateTime specificDate = dateTime.withDayOfMonth(07).withMonth(12).withYear(2000);

System.out.println("Specific date with current time: "+specificDate.format(format));

// gap

Period gap = Period.between(date, date2);

System.out.println("My Age :"+gap);

LocalTime time2 = LocalTime.of(03, 10, 00);

Duration timeGap = Duration.between(time2, time);

System.out.println("TIme gap: " +timeGap);

}

public static void main(String[] args) {

LocalDataTimeApi();

}

}

* **Zoned date-time API** : Use it when time zones are to be considered

**Java 8 Optional Class**

Optional is a container object which may or may not contain a non-null value.

Optional class is available in ***java.util package***.

If a value is present, **isPresent()** will return true and **get()** will return the value.

Additional methods that depend on the presence or absence of a contained value are provided, such as **orElse()** which returns a default value if the value is not present, and **ifPresent()** which executes a block of code if the value is present.

This is a ***value-based*** class, i.e their instances are :

* Final and immutable (though may contain references to mutable objects).
* Considered equal solely based on equals(), not based on reference equality(==).
* Do not have accessible constructors.

Example:

import java.util.Optional;

public class OptionalDemo {

public static void main(String[] args)

{

String[] words = new String[10];

Optional<String> checkNull = Optional.ofNullable(words[5]);

if (checkNull.isPresent()) {

String word = words[5].toLowerCase();

System.out.print(word);

}

else

System.out.println("word is null");

}

}