***JAVA***

*By James Gosling the OAK*

*Sun Microsystems 1991*

***Java****is a class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It is a general-purpose programming language intended to let application developers****write once, run anywhere****(WORA) meaning the machine that compiled Java code can run on all platforms that support Java without the need for recompilation.*

*Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture. The syntax of Java is similar to*[*C*](https://en.wikipedia.org/wiki/C_(programming_language))*and*[*C++*](https://en.wikipedia.org/wiki/C%2B%2B)*, but has fewer low-level facilities than either of them. As of 2019, Java was one of the most popular programming languages in use according to GitHub, particularly for client-server web applications, with a reported 9 million developers.*

*Sun has defined and supports four editions of Java targeting different application environments and segmented many of its APIs so that they belong to one of the platforms. The platforms are:*

* *Java Card for smart-cards.*
* *Java Platform, Micro Edition (Java ME) – targeting environments with limited resource.*
* *Java Platform, Standard Edition (Java SE) – targeting workstation environments.*
* *Java Platform, Enterprise Edition (Java EE) – targeting large distributed enterprise or Internet environments.*

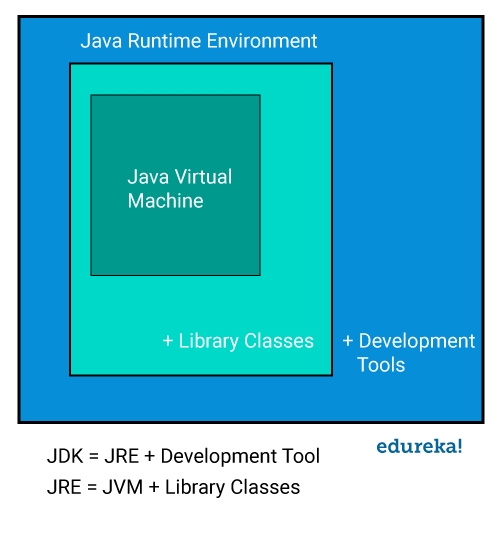
*The*[*classes*](https://en.wikipedia.org/wiki/Class_(computer_programming))*in the Java APIs are organized into separate groups called packages. Each package contains a set of related*[*interfaces*](https://en.wikipedia.org/wiki/Interface_(Java))*, classes, sub packages and exceptions.*

*They all run on JVM-Java Virtual Machine…*

*A****Java virtual machine****(****JVM****) is a*[*virtual machine*](https://en.wikipedia.org/wiki/Virtual_machine)*that enables a computer to run*[*Java*](https://en.wikipedia.org/wiki/Java_(software_platform))*programs as well as programs written in*[*other languages*](https://en.wikipedia.org/wiki/List_of_JVM_languages)*that are also compiled to*[*Java bytecode*](https://en.wikipedia.org/wiki/Java_bytecode)*.*

*The JVM is detailed by a*[*specification*](https://en.wikipedia.org/wiki/Specification_(technical_standard))*that formally describes what is required in a JVM implementation. Having a specification ensures interoperability of Java programs across different implementations so that program authors using the*[*Java Development Kit*](https://en.wikipedia.org/wiki/Java_Development_Kit)*(JDK) need not worry about idiosyncrasies of the underlying hardware platform.*

|  |  |  |
| --- | --- | --- |
| ***JDK*** | ***JRE*** | ***JVM*** |
| *It stands for Java Development Kit.* | *It stands for Java Runtime Environment.* | *It stands for Java Virtual Machine.* |
| *It is the tool necessary to compile, document and package Java programs.* | *JRE refers to a runtime environment in which Java bytecode can be executed.* | *It is an abstract machine. It is a specification that provides a run-time environment in which Java bytecode can be executed.* |
| *It contains JRE + development tools.* | *It’s an implementation of the JVM which physically exists.* | *JVM follows three notations: Specification,****Implementation,****and****Runtime Instance****.* |

**

***JAVA Features***



*The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language. The features of Java are also known as java****buzzwords.***

### ***1) Simple***

*Java is easy to learn and its syntax is quite simple, clean and easy to understand. The confusing and ambiguous concepts of C++ are either left out in Java or they have been re-implemented in a cleaner way.*

*Ex: Pointers and Operator Overloading are not there in java but were an important part of C++.*

### ***2) Object Oriented***

*In java, everything is an object which has some data and behavior. Java can be easily extended as it is based on Object Model. Following are some basic concept of OOP's.*

* *Object*
* *Class*
* *Inheritance*
* *Polymorphism*
* *Abstraction*
* *Encapsulation*

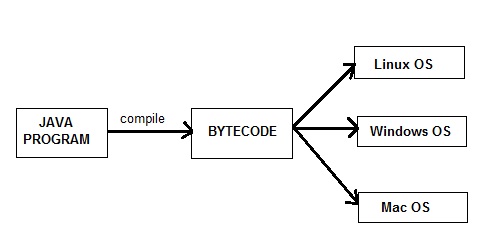
### ***3) Robust***

*Java makes an effort to eliminate error prone codes by emphasizing mainly on compile time error checking and runtime checking. But the main areas which Java improved were Memory Management and mishandled Exceptions by introducing automatic Garbage Collector and Exception Handling.*

### ***4) Platform Independent***

*Unlike other programming languages such as C, C++ etc. which are compiled into platform specific machines. Java is guaranteed to be write-once, run-anywhere language.*

*On compilation Java program is compiled into bytecode. This bytecode is platform independent and can be run on any machine, plus this bytecode format also provide security. Any machine with Java Runtime Environment can run Java Programs.*

**

### ***5) Secure***

*When it comes to security, Java is always the first choice. With java secure features it enable us to develop virus free, temper free system. Java program always runs in Java runtime environment with almost null interaction with system OS, hence it is more secure.*

### ***6) Multi-Threading***

*Java multithreading feature makes it possible to write program that can do many tasks simultaneously. Benefit of multithreading is that it utilizes same memory and other resources to execute multiple threads at the same time, like While typing, grammatical errors are checked along.*

### ***7) Architectural Neutral***

*Compiler generates bytecodes, which have nothing to do with a particular computer architecture, hence a Java program is easy to interpret on any machine.*

### ***8) Portable***

*Java Byte code can be carried to any platform. No implementation dependent features. Everything related to storage is predefined, example: size of primitive data types*

### ***9) High Performance***

*Java is an interpreted language, so it will never be as fast as a compiled language like C or C++. But, Java enables high performance with the use of just-in-time compiler.*

### ***10) Distributed***

*Java is also a distributed language. Programs can be designed to run on computer networks. Java has a special class library for communicating using TCP/IP protocols. Creating network connections is very much easy in Java as compared to C/C++.*

### ***New Features of JAVA 8***

*Below mentioned are some of the core upgrades done as a part of Java 8 release. Just go through them quickly, we will explore them in details later.*

* *Enhanced Productivity by providing Optional Classes feature, Lambda Expressions, Streams etc.*
* *Ease of Use*
* *Improved Polyglot programming. A Polyglot is a program or script, written in a form which is valid in multiple programming languages and it performs the same operations in multiple programming languages. So Java now supports such type of programming technique.*
* *Improved Security and performance.*

### ***New Features of JAVA 11***

*Java 11 is a recommended LTS version of Java that includes various important features. These features includes new and upgrades in existing topic. Just go through them quickly, we will explore them in details later.*

* *includes support for Unicode 10.0.0*
* *The HTTP Client has been standardized*
* *Lazy Allocation of Compiler Threads*
* *Updated Locale Data to Unicode CLDR v33*
* *JEP 331 Low-Overhead Heap Profiling*
* *JEP 181 Nest-Based Access Control*
* *Added Brain pool EC Support (RFC 5639)*
* *Enhanced Keystroke Mechanisms*
* *JEP 332 Transport Layer Security (TLS) 1.3*
* *JEP 330 Launch Single-File Source-Code Programs*

***JAVA Basic syntax!!!***

*The syntax is Java is almost the same as C/C++. But java does not support low-level programming functions like pointers. The codes in Java is always written in the form of Classes and objects.*

* ***Object****− Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behavior such as wagging their tail, barking, eating. An object is an instance of a class.*
* ***Class****− A class can be defined as a template/blueprint that describes the behavior/state that the object of its type supports.*
* ***Methods****− A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.*
* ***Instance Variables****− each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.*
* ***Case Sensitivity****− Java is case sensitive, which means identifier****Hello****and****hello****would have different meaning in Java.*

*Class Names − for all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.*

*Ex: class MyFirstJavaClass*

*Method Names − All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.*

*Ex: public void myMethodName ()*

*Program File Name − Name of the program file should exactly match the class name.*

* ***JAVA Comments****–*

*Single line: //*

*Multiline /\* \*/*

*Documentation //\* \*/*

* ***Java Identifiers***

*All Java components require names. Names used for classes, variables, and methods are called identifiers.*

*In Java, there are several points to remember about identifiers. They are as follows −*

* *All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (\_).*
* *After the first character, identifiers can have any combination of characters.*
* *A key word cannot be used as an identifier.*
* *Most importantly, identifiers are case sensitive.*
* *Examples of legal identifiers: age, $salary, \_value, \_\_1\_value.*
* *Examples of illegal identifiers: 123abc, -salary.*
* ***JAVA Keywords***

*The keywords in java have special meaning to the compiler and they are reserved words with function.*

*Ex: Boolean, int, break, static*

## ***JAVA Modifiers***

*Like other languages, it is possible to modify classes, methods, etc., by using modifiers. There are two categories of modifiers*

1. ***JAVA Access Modifiers****− default, public , protected, private*

## ***Default*** *Access Modifier - No Keyword*

*Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.*

*A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.*

***Private*** *Access Modifier - Private*

*Methods, variables, and constructors that are declared private can only be accessed within the declared class itself.*

*Private access modifier is the most restrictive access level. Class and interfaces cannot be private.*

*Variables that are declared private can be accessed outside the class, if public getter methods are present in the class.*

*Using the private modifier is the main way that an object encapsulates itself and hides data from the outside world.*

## ***Public*** *Access Modifier - Public*

*A class, method, constructor, interface, etc. declared public can be accessed from any other class. Therefore, fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.*

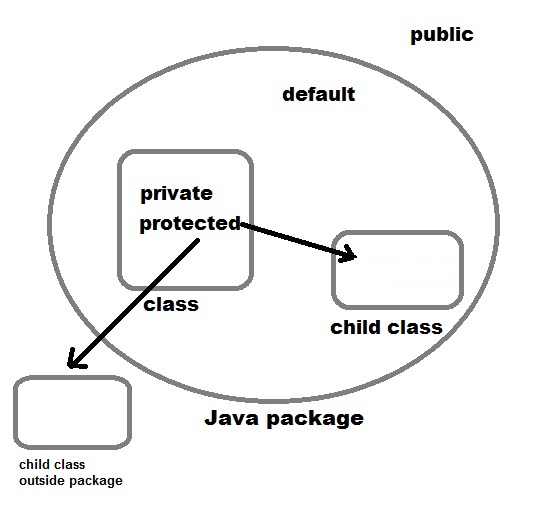
*However, if the public class we are trying to access is in a different package, then the public class still needs to be imported. Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.*

## ***Protected*** *Access Modifier - Protected*

*Variables, methods, and constructors, which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.*

*The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.*

*Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.*



1. ***JAVA Non-access Modifiers****− final, abstract, abstract, synchronized, Volatile.*

## ***Static Modifier***

### ***Static Variables***

*The static keyword is used to create variables that will exist independently of any instances created for the class. Only one copy of the static variable exists regardless of the number of instances of the class.*

*Static variables are also known as class variables. Local variables cannot be declared static.*

### ***Static Methods***

*The static keyword is used to create methods that will exist independently of any instances created for the class.*

*Static methods do not use any instance variables of any object of the class they are defined in. Static methods take all the data from parameters and compute something from those parameters, with no reference to variables.*

*Class variables and methods can be accessed using the class name followed by a dot and the name of the variable or method.*

## ***Final Modifier***

### ***Final Variables***

*A final variable can be explicitly initialized only once. A reference variable declared final can never be reassigned to refer to a different object.*

*However, the data within the object can be changed. So, the state of the object can be changed but not the reference.*

*With variables, the final modifier often is used with static to make the constant a class variable.*

### ***Final Methods***

*A final method cannot be overridden by any subclasses. As mentioned previously, the final modifier prevents a method from being modified in a subclass.*

*The main intention of making a method final would be that the content of the method should not be changed by any outsider.*

## ***Abstract Modifier***

### ***Abstract Class***

*An abstract class can never be instantiated. If a class is declared as abstract then the sole purpose is for the class to be extended.*

*A class cannot be both abstract and final (since a final class cannot be extended). If a class contains abstract methods then the class should be declared abstract. Otherwise, a compile error will be thrown.*

*An abstract class may contain both abstract methods as well normal methods.*

### ***Abstract Methods***

*An abstract method is a method declared without any implementation. The methods body (implementation) is provided by the subclass. Abstract methods can never be final or strict.*

*Any class that extends an abstract class must implement all the abstract methods of the super class, unless the subclass is also an abstract class.*

*If a class contains one or more abstract methods, then the class must be declared abstract. An abstract class does not need to contain abstract methods.*

*The abstract method ends with a semicolon. Example: public abstract sample();*

### ***Synchronized Modifier***

*The synchronized keyword used to indicate that a method can be accessed by only one thread at a time. The synchronized modifier can be applied with any of the four access level modifiers.*

### ***Volatile Modifier***

*The volatile modifier is used to let the JVM know that a thread accessing the variable must always merge its own private copy of the variable with the master copy in the memory.*

*Accessing a volatile variable synchronizes all the cached copied of the variables in the main memory. Volatile can only be applied to instance variables, which are of type object or private. A volatile object reference can be null.*

***JAVA Variables***

*When we want to store any information, we store it in an address of the computer. Instead of remembering the complex address where we have stored our information, we name that address. The naming of an address is known as variable. Variable is the name of memory location.*

*In other words, variable is a name which is used to store a value of any type during program execution.*

*To declare the variable in Java, we can use following syntax*

*datatype variableName;*

*Here, datatype refers to type of variable which can any like: int, float etc. and variableName can be any like: empId, amount, price etc.*

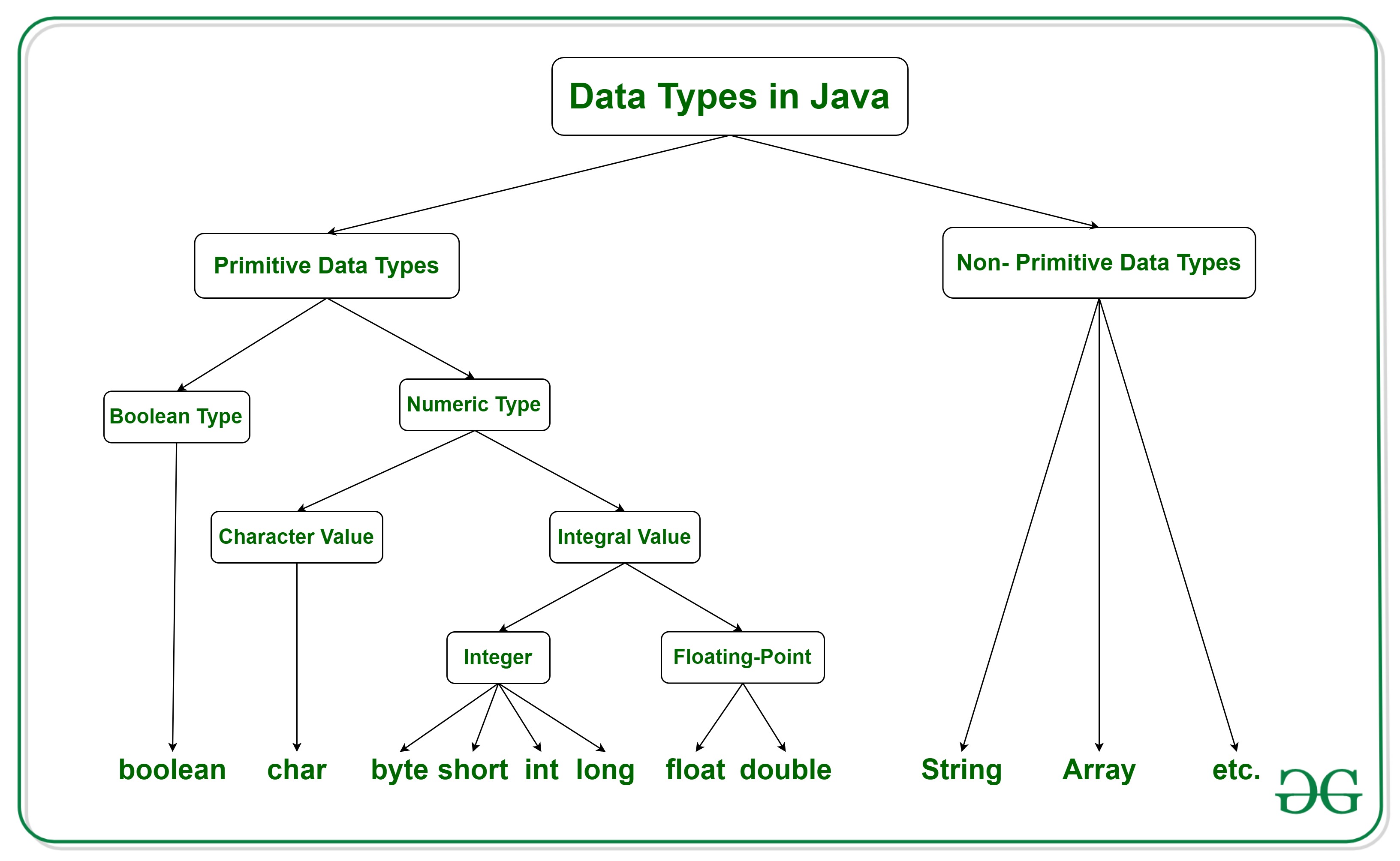
*Java Programming language defines mainly three kind of variables.*

1. *Instance Variables*
2. *Static Variables (Class Variables)*
3. *Local Variables*

***JAVA Data Types***

*Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:*

* *Primitive data types: The primitive data types include Boolean, char, byte, short, int, long, float and double.*
* *Non-primitive data types: The non-primitive data types include*[*Classes*](https://www.javatpoint.com/object-and-class-in-java)*,*[*Interfaces*](https://www.javatpoint.com/interface-in-java)*, and*[*Arrays*](https://www.javatpoint.com/array-in-java)



***JAVA Operators***

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

***JAVA If statements***

*The*[*Java*](https://www.javatpoint.com/java-tutorial)*if statement is used to test the condition. It checks*[*Boolean*](https://www.javatpoint.com/boolean-keyword-in-java)*condition: true or false. There are various types of if statement in Java.*

* *if statement*
* *if-else statement*
* *if-else-if ladder*
* *nested if statement*

1. ***public******class****IfElseIfExample {*
2. ***public******static******void****main(String[] args) {*
3. ***int****marks=65;*
5. ***if****(marks<50){*
6. *System.out.println("fail");*
7. *}*
8. ***else******if****(marks>=50 && marks<60){*
9. *System.out.println("D grade");*
10. *}*
11. ***else******if****(marks>=60 && marks<70){*
12. *System.out.println("C grade");*
13. *}*
14. ***else******if****(marks>=70 && marks<80){*
15. *System.out.println("B grade");*
16. *}*
17. ***else******if****(marks>=80 && marks<90){*
18. *System.out.println("A grade");*
19. *}****else******if****(marks>=90 && marks<100){*
20. *System.out.println("A+ grade");*
21. *}****else****{*
22. *System.out.println("Invalid!");*
23. *}*
24. *}*
25. *}*

***JAVA Loops***



|  |  |  |
| --- | --- | --- |
| *//for loop*  *for(int i=1;i<=10;i++){*  *System.out.println(i);*  *}* | *//while loop*  *int i=1;*  *while(i<=10){*  *System.out.println(i);*  *i++;*  *}* | *//do-while loop*  *int i=1;*  *do{*  *System.out.println(i);*  *i++;*  *}while(i<=10);* |

***JAVA Nested loop***

1. ***public******class****PyramidExample {*
2. ***public******static******void****main(String[] args) {*
3. ***for****(****int****i=1;i<=5;i++){*
4. ***for****(****int****j=1;j<=i;j++){*
5. *System.out.print("\* ");*
6. *}*
7. *System.out.println();//new line*
8. *}*
9. *}*
10. *}*

*Output:*

*\**

*\* \**

*\* \* \**

*\* \* \* \**

*\* \* \* \* \**

***JAVA Array***

*An array is a collection of similar data types. Array is a container object that hold values of homogeneous type. It is also known as static data structure because size of an array must be specified at the time of its declaration.*

*Array starts from zero index and goes to n-1 where n is length of the array.*

*In Java, array is treated as an object and stores into heap memory. It allows to store primitive values or reference values.*

*Array can be single dimensional or multidimensional in Java.*

*class Demo*

*{*

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

*{*

*int arr[ ][ ] = {{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15}};*

*for(int i=0;i<3;i++)*

*{*

*for (int j = 0; j < 5; j++) {*

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

*}*

*System.out.println();*

*}*

*System.out.println("element at first row and second column: " +arr[0][1]);*

*}*

*}*

***JAVA Methods***

*Method in Java is similar to a function defined in other programming languages. Method describes behavior of an object. A method is a collection of statements that are grouped together to perform an operation.*

*For example, if we have a class Human, then this class should have methods like eating(), walking(), talking() etc. which describes the behavior of the object.*

*Declaring method is similar to function. See the syntax to declare the method in Java.*

*public String getName(String st)*

*{*

*String name="StudyTonight";*

*name=name+st;*

*return name;*

*}*

***JAVA Wrapper class***

*In Java, Wrapper Class is used for converting primitive data type into object and object into a primitive data type. For each primitive data type, a pre-defined class is present which is known as Wrapper class.*

*From J2SE 5.0 version the feature of auto boxing and unboxing is used for converting primitive data type into object and object into a primitive data type automatically.*

*As we knew that in Java when input is given by the user, it is in the form of String. To convert a string into different data types, Wrapper classes are used.*

*We can use wrapper class each time when want to convert primitive to object or vice versa.*

*The following are the Primitive Data types with their Name of wrapper class and the method used for the conversation.*

***JAVA Four pillars of OOP***

***Abstraction***

*As abstraction is one of the core principles of Object-oriented programming practices and Java following all OOPs principles, abstraction is one of the major building blocks of java language. In java, abstraction is achieved by**interfaces**and**abstract classes. Interfaces allows you to abstract the implementation completely.*

*Abstraction is very easy to understand when we relate it to the real-time example. For example, when we drive our car, we do not have to be concerned with the exact internal working of the car. What we are concerned with is interacting with the car via its interfaces like steering wheel, brake pedal, accelerator pedal, etc. Here the knowledge we have of the car is abstract.*

*In computer science, abstraction is the process by which data and programs are defined with a representation similar in form to its meaning (*semantics*) while hiding away the implementation details.*

*In more simple terms, abstraction is to hide information that is not relevant to context or rather shows only relevant information and to simplify it by comparing it to something similar in the real world.*

***JAVA Inheritance***

*Inheritance is all about inheriting the****common state and behavior****of parent class (super class) by its derived class (sub class or child class). A sub class can inherit all****non-private members****from super class, by default. In java,****extends****keyword is used for inheritance between classes.*

1. *public class Employee*
2. *{*
3. *private Long id = 10L;*
4. *public Long getId() {*
5. *return id;*
6. *}*
7. *}*
8. *public class Manager extends Employee*
9. *{*
10. *private Long id = 20L;*
11. *public Long getId() {*
12. *return id;*
13. *}*
14. *}*
15. *public class Main*
16. *{*
17. *public static void main(String[] args)*
18. *{*
19. *Employee employee = new Employee();   //Actual object is Employee*
20. *Type*
21. *System.out.println(employee.getId());*
22. *Employee manager = new Manager();    //Actual object is Manager*
23. *Type*
24. *System.out.println(manager.getId());*
25. *Manager mgr = new Manager();    //Actual object is Manager Type*
26. *System.out.println(mgr.getId());*
27. *}*
28. *}*
29. *Output:*
30. *10*
31. *20*
32. *20*

***JAVA Polymorphism***

*Polymorphism is the ability to create a variable, a function, or an object that has more than one form.*

*In java, polymorphism is divided into two parts: method overloading and method overriding.*

*Some may argue that method overloading is not polymorphism. Then what does the term compile time “polymorphism” means??*

*Another term operator overloading is also there, e.g. “+” operator can be used to add two integers as well as concept two sub-strings. Well, this is the only available support for operator overloading in java, and you cannot have your own custom defined operator overloading in java.*

***JAVA Encapsulation***

*Wrapping data and methods within classes in combination with implementation hiding (through access control) is often called encapsulation. The result is a data type with characteristics and behaviors.****Encapsulation essentially has both i.e. information hiding and implementation hiding.***

***JAVA Constructors***

*In*[*Java*](https://www.javatpoint.com/java-tutorial)*, a constructor is a block of codes similar to the method. It is called when an instance of the*[*class*](https://www.javatpoint.com/object-and-class-in-java)*is created. At the time of calling constructor, memory for the object is allocated in the memory.*

*It is a special type of method which is used to initialize the object.*

*Every time an object is created using the new() keyword, at least one constructor is called.*

***JAVA Overloaded methods***

*Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different. It is similar to*[*constructor overloading*](https://beginnersbook.com/2013/05/constructor-overloading/)*in Java that allows a class to have more than one constructor having different argument lists.*

***JAVA Overloaded constructors***

*Java Constructor overloading is a technique in which a class can have any number of constructors that differ in parameter list. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.*

***JAVA Overriding method***

*If subclass (child class) has the same method as declared in the parent class, it is known as****method overriding in Java****.*

*In other words, if a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.*

### *Usage of Java Method Overriding*

* *Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.*
* *Method overriding is used for runtime polymorphism*

*Rules for Java Method Overriding*

1. *The method must have the same name as in the parent class*
2. *The method must have the same parameter as in the parent class.*
3. *There must be an IS-A relationship (inheritance).*

***JAVA toString()method***

*If you want to represent any object as a string, toString() method comes into existence.*

*The toString() method returns the string representation of the object.*

*If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object etc. depends on your implementation. By overriding the toString() method of the Object class, we can return values of the object, so we don't need to write much code.*

class HelloWorld {

    public static void main( String args[] ) {

        //Creating an integer of value 10

        Integer number=10;

        // Calling the toString() method as a function of the Integer variable

        System.out.println( number.toString() );

    }

}

***JAVA Exception handling***

*The****Exception Handling in Java****is one of the powerful*mechanism to handle the runtime errors*so that normal flow of the application can be maintained. In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.*

1. *TRY:*

*The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.*

1. *CATCH:*

*The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.*

1. *FINALLY:*

*The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.*

1. *THROW:*

*The "throw" keyword is used to throw an exception.*

1. *THROWS:*

*The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.*

***JAVA Enum***

*The****Enum in Java****is a data type which contains a fixed set of constants.*

*It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY) , directions (NORTH, SOUTH, EAST, and WEST), season (SPRING, SUMMER, WINTER, and AUTUMN or FALL), colors (RED, YELLOW, BLUE, GREEN, WHITE, and BLACK) etc. According to the Java naming conventions, we should have all constants in capital letters. So, we have enum constants in capital letters.*

*Java Enums can be thought of as classes which have a fixed set of constants (a variable that does not change). The Java enum constants are static and final implicitly.*

*Java Enum internally inherits the*Enum class*, so it cannot inherit any other class, but it can implement many interfaces. We can have fields, constructors, methods, and main methods in Java enum.*

***JAVA Collections***

*The****Collection in Java****is a framework that provides an architecture to store and manipulate the group of objects.*

*Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.*

*Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes ([ArrayList](https://www.javatpoint.com/java-arraylist), Vector, [LinkedList](https://www.javatpoint.com/java-linkedlist),*[*PriorityQueue*](https://www.javatpoint.com/java-priorityqueue)*, HashSet, LinkedHashSet, TreeSet).*

**

**1) AD -** Allow Duplicates: It tells whether that particular collection allows duplicate values  
to be inserted.

**2) AN -** Allow Null: It tells whether null objects can be inserted into that particular  
collection.

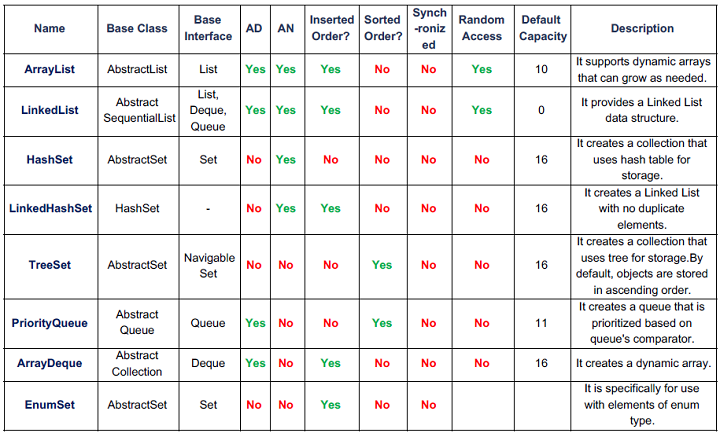
**3) Inserted Order:** It tells whether the objects are stored in the same order in which they  
were inserted.

**4) Sorted Order:** It tells whether the objects are stored in sorted order.

**5) Synchronized:** It tells whether the collection is thread-safe or not.

**6) Random Access:** It tells whether the collection has a get() method to returns the index  
of an object or return the object using an index.

**7) Default capacity:** The initial capacity of the collection when it is created using an empty  
constructor.



* ***JAVA ArrayList***

*Java****ArrayList****class uses a*dynamic [*array*](https://www.javatpoint.com/array-in-java)*for storing the elements. It is like an array, but there is*no size limit*. We can add or remove elements anytime. So, it is much more flexible than the traditional array. It is found in the*java.util*package. It is like the Vector in C++.*

*The ArrayList in Java can have the duplicate elements also. It implements the List interface so we can use all the methods of List interface here. The ArrayList maintains the insertion order internally.*

*It inherits the AbstractList class and implements*[*List interface*](https://www.javatpoint.com/java-list)*.*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | public class Arraylist {  public static void main(String args[])  {  ArrayList al= new ArrayList();  al.add("Edureka");  al.add("Java");  al.add("Arrays");  System.out.println("Size of arraylist:" +al.size());  System.out.println("Contents of al:" +al);  al.remove("Java"); // Removes Java element from the list  System.out.println("Contents of al:" +al);  int asize = al.size(); // returns the size of the array  System.out.println("Size of the array is:" +asize);  al.clear(); // Removes element from Array  }  } |

* ***JAVA LinkedList***

*Java LinkedList class uses a doubly linked list to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.*

*The important points about Java LinkedList are:*

* *Java LinkedList class can contain duplicate elements.*
* *Java LinkedList class maintains insertion order.*
* *Java LinkedList class is non synchronized.*
* *In Java LinkedList class, manipulation is fast because no shifting needs to occur.*
* *Java LinkedList class can be used as a list, stack or queue.*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53 | package MyPackage;    import java.util.LinkedList;  import java.util.ListIterator;    public class linkedlist {      public static void main(String args[]) {          /\* Linked List Declaration \*/       LinkedList<String> l\_list = new LinkedList<String>();        /\*add(String Item) is used for adding         \* the Items to the linked list\*/        l\_list.add("Java");        l\_list.add("Python");        l\_list.add("Scala");        l\_list.add("Swift");        System.out.println("Linked List Content: " +l\_list);          /\*Add Items at specified position\*/        l\_list.add(2, "JavaScript");        l\_list.add(3, "Kotlin");        System.out.println("l\_list Content after editing: " +l\_list);          /\*Add First and Last Item\*/        l\_list.addFirst("First Course");        l\_list.addLast("Last Course");        System.out.println("l\_list Content after addition: " +l\_list);          /\*Get and set Items in the list\*/        Object firstvar = l\_list.get(0);        System.out.println("First Item: " +firstvar);        l\_list.set(0, "Java9");        System.out.println("l\_list Content after updating first Item: " +l\_list);          /\* Remove from a position\*/        l\_list.remove(1);        l\_list.remove(2);        System.out.println("LinkedList after deletion of Item in 2nd and 3rd position " +l\_list);          /\*Remove first and last Item\*/        l\_list.removeFirst();        l\_list.removeLast();        System.out.println("Final Content after removing first and last Item: "+l\_list);          /\*Iterating the linked list\*/        ListIterator<String> itrator = l\_list.listIterator();        System.out.println("List displayed using iterator:");        while (itrator.hasNext()) {            System.out.println(itrator.next());       }  }  } |

* ***JAVA HashSet***

*Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.*

*The important points about Java HashSet class are:*

* *HashSet stores the elements by using a mechanism called****hashing.***
* *HashSet contains unique elements only.*
* *HashSet allows null value.*
* *HashSet class is none synchronized.*
* *HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.*
* *HashSet is the best approach for search operations.*
* *The initial default capacity of HashSet is 16, and the load factor is 0.75.*

***Difference between List and Set.*** *A list can contain duplicate elements whereas Set contains unique elements only.*

|  |  |
| --- | --- |
| *1*  *2*  *3*  *4*  *5*  *6*  *7*  *8*  *9*  *10*  *11*  *12*  *13*  *14*  *15* | *import java.util.\*;*  *class HashsetExample{*  *public static void main(String args[]){*    *HashSet&amp;amp;lt;String&amp;amp;gt; al=new HashSet(); // creating*  *hashSet*  *al.add("Rachit");                 // adding elements*  *al.add("Amit");*  *al.add("jack");*  *Iterator&amp;amp;lt;String&amp;amp;gt; itr=al.iterator();*  *while(itr.hasNext()){*  *System.out.println(itr.next());*  *}*  *}*  *}* |

* ***JAVA LinkedHashSet***

*Java LinkedHashSet class is a Hashtable and Linked list implementation of the set interface. It inherits HashSet class and implements Set interface.*

*The important points about Java LinkedHashSet class are:*

* *Java LinkedHashSet class contains unique elements only like HashSet.*
* *Java LinkedHashSet class provides all optional set operation and permits null elements.*
* *Java LinkedHashSet class is none synchronized.*
* *Java LinkedHashSet class maintains insertion order.*

*The LinkedHashSet class extends HashSet class which implements Set interface. The Set interface inherits Collection and Iterable interfaces in hierarchical order.*

|  |  |
| --- | --- |
| *1*  *2*  *3*  *4*  *5*  *6*  *7*  *8*  *9*  *10*  *11*  *12*  *13*  *14* | *import java.util.\*;*  *class LinkedHashsetExample{*  *public static void main(String args[]){*  *LinkedHashSet&amp;amp;lt;String&amp;amp;gt; al=new LinkedHashSet();*  *// creating linkedhashset*  *al.add("Mariana"); // adding elements*  *al.add("Rick");*  *al.add("Sam");*  *Iterator&amp;amp;lt;String&amp;amp;gt; itr=al.iterator();*  *while(itr.hasNext()){*  *System.out.println(itr.next());*  *}*  *}*  *}*  *}* |

* ***JAVA TreeSet***

*Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements the NavigableSet interface. The objects of the TreeSet class are stored in ascending order.*

*The important points about Java TreeSet class are:*

* *Java TreeSet class contains unique elements only like HashSet.*
* *Java TreeSet class access and retrieval times are quiet fast.*
* *Java TreeSet class doesn't allow null element.*
* *Java TreeSet class is non synchronized.*
* *Java TreeSet class maintains ascending order.*

*As shown in the above diagram, Java TreeSet class implements the NavigableSet interface. The NavigableSet interface extends SortedSet, Set, Collection and Iterable interfaces in hierarchical order.*

|  |  |
| --- | --- |
| *1*  *2*  *3*  *4*  *5*  *6*  *7*  *8*  *9*  *10*  *11*  *12*  *13*  *14* | *import java.util.\*;*  *class TreeSetExample{*  *public static void main(String args[]){*  *TreeSet&amp;amp;lt;String&amp;amp;gt; al=new*  *TreeSet&amp;amp;lt;String&amp;amp;gt;();  // creating treeSet*  *al.add("John");                            // adding elements*  *al.add("Sam");*  *al.add("Rick");*  *Iterator&amp;amp;lt;String&amp;amp;gt; itr=al.iterator();*  *while(itr.hasNext()){*  *System.out.println(itr.next());*  *}*  *}*  *}* |

* ***JAVA Queue & Deque***

*Java Queue interface orders the element in FIFO(First In First Out) manner. In FIFO, first element is removed first and last element is removed at last.*

*Java Deque Interface is a linear collection that supports element insertion and removal at both ends. Deque is an acronym for****"double ended queue".***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | import java.util.\*;  class QueueExample {  public static void main(String args[]){  PriorityQueue<String> queue=new PriorityQueue<String>();  // creating priority queue  queue.add("Amit");  // adding elements  queue.add("Rachit");  queue.add("Rahul");  System.out.println("head:"+queue.element());  System.out.println("head:"+queue.peek());  System.out.println("iterating the queue elements:");  Iterator itr=queue.iterator();  while(itr.hasNext()){  System.out.println(itr.next());  }  queue.remove();  queue.poll();  System.out.println("after removing two elements:");  Iterator<String> itr2=queue.iterator();  while(itr2.hasNext()){  System.out.println(itr2.next());  }  }  } |

|  |  |
| --- | --- |
| *1*  *2*  *3*  *4*  *5*  *6*  *7*  *8*  *9*  *10*  *11*  *12*  *13*  *14*  *15*  *16*  *17*  *18*  *19*  *20*  *21*  *22*  *23*  *24*  *25*  *26*  *27*  *28*  *29*  *30*  *31*  *32*  *33*  *34*  *35*  *36*  *37*  *38*  *39*  *40*  *41*  *42*  *43*  *44*  *45*  *46*  *47*  *48*  *49*  *50*  *51*  *52*  *53*  *54*  *55*  *56*  *57*  *58*  *59*  *60*  *61*  *62*  *63*  *64*  *65*  *66*  *67*  *68*  *69*  *70*  *71*  *72* | *import java.util.ArrayList;*  *import java.util.List;*  *public class DoubleEndedQueueImpl {*  *private List<Integer> deque = new ArrayList<Integer>();*  *public void insertFront(int item){*  *//add element at the beginning of the queue*  *System.out.println("element added at front: "+item);*  *deque.add(0,item);*  *System.out.println(deque);*  *}*  *public void insertRear(int item){*  *//add element at the end of the queue*  *System.out.println("element added at rear: "+item);*  *deque.add(item);*  *System.out.println(deque);*  *}*  *public void removeFront(){*  *if(deque.isEmpty()){*  *System.out.println("Deque underflow, unable to remove.");*  *return;*  *}*    *//remove an item from the beginning of the queue*  *int rem = deque.remove(0);*  *System.out.println("element removed from front: "+rem);*  *System.out.println(deque);*  *}*    *public void removeRear(){*  *if(deque.isEmpty()){*  *System.out.println("Deque underflow, unable to remove.");*  *return;*  *}*    *//remove an item from the beginning of the queue*  *int rem = deque.remove(deque.size()-1);*  *System.out.println("element removed from front: "+rem);*  *System.out.println(deque);*  *}*        *public int peakFront(){*  *//gets the element from the front without removing it*  *int item = deque.get(0);*  *System.out.println("Element at first: "+item);*  *return item;*  *}*        *public int peakRear(){*  *//gets the element from the rear without removing it*  *int item = deque.get(deque.size()-1);*  *System.out.println("Element at rear: "+item);*  *return item;*  *}*    *public static void main(String a[]){*  *DoubleEndedQueueImpl deq = new DoubleEndedQueueImpl();*  *deq.insertFront(34);*  *deq.insertRear(45);*  *deq.removeFront();*  *deq.removeFront();*  *deq.removeFront();*  *deq.insertFront(21);*  *deq.insertFront(98);*  *deq.insertRear(5);*  *deq.insertFront(43);*  *deq.removeRear();*  *}*  *}* |

* ***JAVA HashMap***

*Java HashMap class implements the Map interface which allows us to store key and value pair, where keys should be unique. If you try to insert the duplicate key. It is easy to perform operations using key index like updation, delition, etc. HashMap class is found in the java.util package.*

*HashMap in Java is like the legacy Hashtable class, but it is not synchronized. It allows us to store the null elements as well, but there should be only one null key. Since Java 5, it is denoted as HashMap<K,V>, where K stands for key and V for value. It inherits the AbstractMap class and implements the Map interface.*

* *Java HashMap contains values based on the key.*
* *Java HashMap contains only unique keys.*
* *Java HashMap may have one null key and multiple null values.*
* *Java HashMap is non synchronized.*
* *Java HashMap maintains no order.*
* *The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.*

*As shown in the above figure, HashMap class extends AbstractMap class and implements Map interface.*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | package Edureka;    //Java program to illustrate  //Java.util.HashMap  import java.util.HashMap;  import java.util.Map;    public class Hashmap{  public static void main(String[] args){  HashMa<String, Integer> map = new HashMap<>();  print(map);  map.put("abc", 10);  map.put("mno", 30);  map.put("xyz", 20);    System.out.println("Size of map is" + map.size());    print(map);  if (map.containsKey("mno"))  {  Integer a = map.get("mno");  System.out.println("value for key \"mno\" is:- " + a);  }    map.clear();  print(map);  }    public static void print(Map<String, Integer> map){  if (map.isEmpty()){  System.out.println("map is empty");  }  else{  System.out.println(map);  }  }  } |

* ***JAVA LinkedHashMap***

*Java LinkedHashMap class is Hashtable and Linked list implementation of the Map interface, with predictable iteration order. It inherits HashMap class and implements the Map interface.*

* *Java LinkedHashMap contains values based on the key.*
* *Java LinkedHashMap contains unique elements.*
* *Java LinkedHashMap may have one null key and multiple null values.*
* *Java LinkedHashMap is non synchronized.*
* *Java LinkedHashMap maintains insertion order.*
* *The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.*

|  |
| --- |
| * 1. ***import****java.util.\*;*   2. ***class****Book {*   3. ***int****id;    String name,author,publisher;*   4. ***int****quantity;*   5. ***public****Book(****int****id, String name, String author, String publisher,****int****quantity) {*   6. ***this****.id = id;*   7. ***this****.name = name;*   8. ***this****.author = author;*   9. ***this****.publisher = publisher;****this****.quantity = quantity;*   10. *}*   11. *}*   12. ***public******class****MapExample {*   13. ***public******static******void****main(String[] args) {*   14. *//Creating map of Books*   15. *Map<Integer,Book> map=****new****LinkedHashMap<Integer,Book>();*   16. *//Creating Books*   17. *Book b1=****new****Book(101,"Let us C","Yashwant Kanetkar","BPB",8);*   18. *Book b2=****new****Book(102,"Data Communications & Networking","Forouzan",*   19. *"Mc Graw Hill",4);*   20. *Book b3=****new****Book(103,"Operating System","Galvin","Wiley",6);*   21. *//Adding Books to map*   22. *map.put(2,b2);*   23. *map.put(1,b1);*   24. *map.put(3,b3);*   25. *//Traversing map*   26. ***for****(Map.Entry<Integer, Book> entry:map.entrySet()){*   ***int****key=entry.getKey();*  *Book b=entry.getValue();*  *System.out.println(key+" Details:");*  *System.out.println*  *(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);*   * 1. *}*   2. *}*   3. *}* |

* ***JAVA TreeMap***

*Java TreeMap class is a red-black tree based implementation. It provides an efficient means of storing key-value pairs in sorted order.*

*The important points about Java TreeMap class are:*

* *Java TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.*
* *Java TreeMap contains only unique elements.*
* *Java TreeMap cannot have a null key but can have multiple null values.*
* *Java TreeMap is non synchronized.*
* *Java TreeMap maintains ascending order.*

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | *1*  *2*  *3*  *4*  *5*  *6*  *7*  *8*  *9*  *10*  *11*  *12*  *13*  *14*  *15*  *16*  *17*  *18*  *19*  *20*  *21*  *22*  *23*  *24*  *25*  *26*  *27*  *28*  *29*  *30*  *31*  *32*  *33*  *34*  *35*  *36*  *37*  *38*  *39* | 1. ***import****java.util.\*;* 2. ***class****Book {* 3. ***int****id;* 4. *String name,author,publisher;* 5. ***int****quantity;* 6. ***public****Book(****int****id, String name, String author, String publisher,****int****quantity) {* 7. ***this****.id = id;* 8. ***this****.name = name;* 9. ***this****.author = author;* 10. ***this****.publisher = publisher;* 11. ***this****.quantity = quantity;* 12. *}* 13. *}* 14. ***public******class****MapExample {* 15. ***public******static******void****main(String[] args) {* 16. *//Creating map of Books* 17. *Map<Integer,Book> map=****new****TreeMap<Integer,Book>();* 18. *//Creating Books* 19. *Book b1=****new****Book(101,"Let us C","Yashwant Kanetkar","BPB",8);* 20. *Book b2=****new****Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);* 21. *Book b3=****new****Book(103,"Operating System","Galvin","Wiley",6);* 22. *//Adding Books to map* 23. *map.put(2,b2);* 24. *map.put(1,b1);* 25. *map.put(3,b3);* 27. *//Traversing map* 28. ***for****(Map.Entry<Integer, Book> entry:map.entrySet()){* 29. ***int****key=entry.getKey();* 30. *Book b=entry.getValue();* 31. *System.out.println(key+" Details:");* 32. *System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);* 33. *}* 34. *}* 35. *}* | |

* ***JAVA HashTable***

*Java Hashtable class implements a hashtable, which maps keys to values. It inherits Dictionary class and implements the Map interface.*

* *A Hashtable is an array of a list. Each list is known as a bucket. The position of the bucket is identified by calling the hashcode() method. A Hashtable contains values based on the key.*
* *Java Hashtable class contains unique elements.*
* *Java Hashtable class doesn't allow null key or value.*
* *Java Hashtable class is synchronized.*
* *The initial default capacity of Hashtable class is 11 whereas loadFactor is 0.75.*
* ***JAVA EnumMap***

*Java EnumMap class is the specialized Map implementation for enum keys. It inherits Enum and AbstractMap classes.*

* ***JAVA EnumSet***

*Java EnumSet class is the specialized Set implementation for use with enum types. It inherits AbstractSet class and implements the Set interface.*

* ***JAVA Vector***

***Vector****is like the*dynamic array*which can grow or shrink its size. Unlike array, we can store n-number of elements in it as there is no size limit. It is a part of Java Collection framework since Java 1.2. It is found in the java.util package and implements the*List*interface, so we can use all the methods of List interface here.*

*It is recommended to use the Vector class in the thread-safe implementation only. If you don't need to use the thread-safe implementation, you should use the ArrayList, the ArrayList will perform better in such case.*

*The Iterators returned by the Vector class are*fail-fast*. In case of concurrent modification, it fails and throws the ConcurrentModificationException.*

*It is similar to the ArrayList, but with two differences-*

* *Vector is synchronized.*
* *Java Vector contains many legacy methods that are not the part of a collections framework.*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | / Java code illustrating Vector Constructors  import java.util.\*;  public class Main{      public static void main(String[] args)      {              // create default vector              Vector v1 = new Vector();          // create a vector of given Size              Vector v2 = new Vector(20);          // create a vector of given Size and Increment              Vector v3 = new Vector(30,10);              v2.add(100);              v2.add(100);              v2.add(100);          // create a vector with given collection              Vector v4 = new Vector(v2);            System.out.println("Vector v1 of capacity " + v1.capacity());          System.out.println("Vector v2 of capacity " + v2.capacity());          System.out.println("Vector v3 of capacity " + v3.capacity());      System.out.println("Vector v4 of capacity " + v4.capacity());      } |

* ***JAVA Stack***

*The****stack****is a linear data structure that is used to store the collection of objects. It is based on****Last-In-First-Out****(LIFO).*[*Java collection*](https://www.javatpoint.com/collections-in-java)*framework provides many interfaces and classes to store the collection of objects. One of them is the****Stack class****that provides different operations such as push, pop, search, etc.*

*In this section, we will discuss the****Java Stack class,*** *its****methods,****and****implement****the stack data structure in a*[*Java program*](https://www.javatpoint.com/java-programs)*. But before moving to the Java Stack class have a quick view of how the stack works.*

*The stack data structure has the two most important operations that are****push****and****pop****. The push operation inserts an element into the stack and pop operation removes an element from the top of the stack. Let's see how they work on stack.*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  334  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | 1. **import** java.util.\*; 2. **public** **class** StackPushPopExample 3. { 4. **public** **static** **void** main(String args[]) 5. { 6. //creating an object of Stack class 7. Stack <Integer> stk = **new** Stack<>(); 8. System.out.println("stack: " + stk); 9. //pushing elements into the stack 10. pushelmnt(stk, 20); 11. pushelmnt(stk, 13); 12. pushelmnt(stk, 89); 13. pushelmnt(stk, 90); 14. pushelmnt(stk, 11); 15. pushelmnt(stk, 45); 16. pushelmnt(stk, 18); 17. //popping elements from the stack 18. popelmnt(stk); 19. popelmnt(stk); 20. //throws exception if the stack is empty 21. **try** 22. { 23. popelmnt(stk); 24. } 25. **catch** (EmptyStackException e) 26. { 27. System.out.println("empty stack"); 28. } 29. } 30. //performing push operation 31. **static** **void** pushelmnt(Stack stk, **int** x) 32. { 33. //invoking push() method 34. stk.push(**new** Integer(x)); 35. System.out.println("push -> " + x); 36. //prints modified stack 37. System.out.println("stack: " + stk); 38. } 39. //performing pop operation 40. **static** **void** popelmnt(Stack stk) 41. { 42. System.out.print("pop -> "); 43. //invoking pop() method 44. Integer x = (Integer) stk.pop(); 45. System.out.println(x); 46. //prints modified stack 47. System.out.println("stack: " + stk); 48. } 49. } |

***JAVA File class File reader, File writer***

***Still to come!!!***

***JAVA Generics***

***Still to come!!!***

***JAVA Algorithms & Data Structures***

## ***Data Structures in Java***

A data structure is a way of storing and organizing data in a computer so that it can be used efficiently. It provides a means to manage large amounts of data efficiently. And efficient data structures are key to designing efficient algorithms.

*An algorithm is a step procedure to solve logical and mathematical problems. A recipe is a good example of an algorithm because it says what must be done, step by step. In computing, an algorithm is a precise list of operations that could be done by a Turing machine.*

*Let’s explore the two major categories of algorithms in Java, which are:*

* *Sorting Algorithms in Java*
* *Searching Algorithms in Java*

***Linear Search***

*Linear search is a very simple algorithm that starts searching for an element or a value from the beginning of an array until the required element is not found. It compares the element to be searched with all the elements in an array, if the match is found, then it returns the index of the element else it returns -1. This algorithm can be implemented on the unsorted list.*

*Here’s pseudocode representing Linear Search in Java:*

|  |  |
| --- | --- |
| *1*  *2*  *3*  *4*  *5*  *6*  *7*  *8*  *9*  *10* | *procedure linear\_search (a[] , value)*  *for i = 0 to n-1*  *if a[i] = value then*  *print "Found "*  *return i*  *end if*  *print "Not found"*  *end for*    *end linear\_search* |

***Binary Search***

*This kind of search uses the Divide and Conquer methodology and requires the data set to be sorted beforehand.*

*It divides the input collection into equal halves, and with each iteration compares the goal element with the element in the middle.*

*If the element is found, the search ends. Else, we continue looking for the element by dividing and selecting the appropriate partition of the array, based on if the goal element is smaller or bigger than the middle element.*

*This is why it's important to have a sorted collection for Binary Search.*

*The search terminates when the firstIndex (our pointer) goes past lastIndex (last element), which implies we have searched the whole array and the element is not present.*

*There are two ways to implement this algorithm - iterative and recursive.*

*There shouldn't be a difference regarding time and space complexity between these two implementations, though this doesn't hold true to all languages.*

*Here’s pseudocode representing Binary Search in Java:*

|  |  |
| --- | --- |
| *1*  *2*  *3*  *4*  *5*  *6*  *7*  *8*  *9*  *10*  *11*  *12*  *13*  *14*  *15*  *16*  *17*  *18*  *19* | *Procedure binary\_search*  *a; sorted array*  *n; size of array*  *x; value to be searched*  *lowerBound = 1*  *upperBound = n*  *while x not found*  *if upperBound < lowerBound*  *EXIT: x does not exists.*  *set midPoint = lowerBound + ( upperBound - lowerBound ) / 2*  *if A[midPoint] < x set lowerBound = midPoint + 1 if A[midPoint] > x*  *set upperBound = midPoint - 1*  *if A[midPoint] = x*  *EXIT: x found at location midPoint*  *end while*  *end procedure* |

***Fibonacci Search***

#### **Explanation:**

*Fibonacci search employs divide and conquer approach wherein we unequally split element as per the Fibonacci series. This search requires the array to be sorted.*

*Unlike in Binary Search where we divide the elements into equal halves to reduce the array range - In Fibonacci search we try to use addition or subtraction to get a smaller range.*

*Remember the formula for Fibonacci series is:*

*Fibo(N)=Fibo(N−1)+Fibo(N−2)Fibo(N)=Fibo(N−1)+Fibo(N−2)*

*The first two numbers in this series are Fibo(0) = 0 and Fibo(1) = 1. So as per this formula, the series looks like this*0, 1, 1, 2, 3, 5, 8, 13, 21...

*Interesting observations to note here is that:*

*Fibo(N-2) is approximately 1/3rd of Fibo(N)*

*Fibo(N-1) is approximately 2/3rd of Fibo(N)*

*So when we use fibonacci series numbers to partition the range it gets split in the same ratio as above.*

1. *package com.search.demo;*
2. *public class FibonacciSearch {*
3. *static int[] a = {10,20,30,40,50,60,70,80,90,100};*
4. *static int required = 70;*
5. *static int m = 2;*
6. *static int p = 0;*
7. *static int q = 0;*
8. */\*\**
9. *@param args*
10. *\*/*
11. *public static void main(String[] args) {*
12. *// TODO Auto-generated method stub*
13. *FibonacciSearch fs = new FibonacciSearch();*
14. *fs.findm();*
15. *fibSearch(required);*
16. *}*
17. *private void findm(){*
18. *//here you have to find Fm which matches size of searching array, or which is close to it.*
19. *int n = a.length;*
20. *int fibCurrent = 1;*
21. *int fibPrev1 = 1;*
22. *int fibPrev2 = 0;*
23. *while(n > fibCurrent){*
24. *fibPrev2 = fibPrev1;*
25. *fibPrev1 = fibCurrent;*
26. *fibCurrent = fibPrev1 + fibPrev2;*
27. *m++;*
28. *}*
29. *p = m-1;*
30. *q = m-2;*
31. *}*
32. *public static int fibSearch(int no){*
33. *for(;;){*
34. *if(m == 0){*
35. *System.out.println("not found");*
36. *return -1;*
37. *}*
38. *int j = f(p);*
39. *if(no == a[j]){*
40. *System.out.println("found at "+p);*
41. *}else if(no < a[j]){*
42. *m = p;*
43. *p = m - 1;*
44. *q = m - 2;*
45. *}else if(no > a[j]){*
46. *m = q; // as per the step 6..*
47. *p = m-1;*
48. *q = m-2;*
49. *}*
50. *}*
51. *//return m;*
52. *}*
53. *public static int f(int val){*
54. *if(val == 2 || val == 1 || val == 0){*
55. *return 1;*
56. *}*
57. *return (f(val-1) + f(val-2));*
58. *}*
59. *}*

### ***Sorting Algorithms***

*Sorting algorithms are used to rearrange the elements in an array or a given data structure either in an ascending or descending order. The comparison operator decides the new order of the elements.*

***Quick Sort-****Quicksort algorithm is a fast, recursive, non-stable sort algorithm which works by the divide and conquer principle. It picks an element as pivot and partitions the given array around that picked pivot.*

*Steps to implement Quick sort:*

1. *Pick a suitable “pivot point”.*
2. *Divide the lists into two lists based on this pivot element. Every element which is smaller than the pivot element is placed in the left list and every element which is larger is placed in the right list. If an element is equal to the pivot element then it can go in any list. This is called the partition operation.*
3. *Recursively sort each of the smaller lists.*

*Here’s pseudocode representing Quicksort Algorithm.*

|  |  |
| --- | --- |
| *1*  *2*  *3*  *4*  *5*  *6*  *7*  *8*  *9*  *10*  *11*  *12*  *13*  *14*  *15*  *16*  *17*  *18*  *19*  *20* | *QuickSort(A as array, low as int, high as int){*  *if (low < high){*  *pivot\_location = Partition(A,low,high)*  *Quicksort(A,low, pivot\_location)*  *Quicksort(A, pivot\_location + 1, high)*  *}*  *}*  *Partition(A as array, low as int, high as int){*  *pivot = A[low]*  *left = low*    *for i = low + 1 to high{*  *if (A[i] < pivot) then{*  *swap(A[i], A[left + 1])*  *left = left + 1*  *}*  *}*  *swap(pivot,A[left])*    *return (left)}* |

*In the above pseudocode, partition() function performs partition operation and Quicksort() function repeatedly calls partition function for each smaller list generated. The complexity of quicksort in the average case is Θ(n log(n)) and in the worst case is Θ(n2).*

***Bubble Sort***

*Bubble Sort, often referred to as sinking sort, is the simplest sorting algorithm. It repeatedly steps through the list to be sorted, compares each pair of adjacent elements and swaps them if they are in the wrong order. Bubble sort gets its name because it filters out the elements to the top of the array, like bubbles that float on water.*

*Here’s pseudocode representing Bubble Sort Algorithm (ascending sort context).*

|  |  |
| --- | --- |
| *1*  *2*  *3*  *4*  *5*  *6*  *7*  *8*  *9*  *10*  *11*  *12*  *13* | *a[] is an array of size N*  *begin BubbleSort(a[])*    *declare integer i, j*  *for i = 0 to N - 1*  *for j = 0 to N - i - 1*  *if a[j] > a[j+1] then*  *swap a[j], a[j+1]*  *end if*  *end for*  *return a*    *end BubbleSort* |

***JAVA Find factorial***

**Factorial Program** in Java: Factorial of n is the product of all positive descending integers. Factorial of n is denoted by n!. For example:

1. 4! = 4\*3\*2\*1 = 24
2. 5! = 5\*4\*3\*2\*1 = 120

Here, 4! is pronounced as "4 factorial", it is also called "4 bang" or "4 shriek".

The factorial is normally used in Combinations and Permutations (mathematics).

There are many ways to write the factorial program in java language. Let's see the 2 ways to write the factorial program in java.

1. *Factorial Program using loop*
2. *Factorial Program using recursion*
3. ***class****FactorialExample2{*
4. ***static******int****factorial(****int****n){*
5. ***if****(n == 0)*
6. ***return****1;*
7. ***else***
8. ***return****(n \* factorial(n-1));*
9. *}*
10. ***public******static******void****main(String args[]){*
11. ***int****i,fact=1;*
12. ***int****number=4;//It is the number to calculate factorial*
13. *fact = factorial(number);*
14. *System.out.println("Factorial of "+number+" is: "+fact);*
15. *}*
16. *}*