**Java**

JDK – JDK provides env to **develop** and **run** java application.

JRE – JRE provides env to **run** java application.

JVM – In JRE, JVM is responsible to run the program line by line. JVM is an interpreter.

**Interpreter** translates just one statement of the program at a time into machine code. **Compiler** scans the entire program and translates the whole of it into machine code at once.

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

**Platform**: Any hardware or software in which a program runs, is known as a platform. Since java has a runtime environment (JRE) and API, it is called as a Platform.

**Object-Oriented**:

Java is object-oriented program language. Everything in java is an object. Object oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

**Platform Independent:**

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

There are two types of platforms software-based and hardware-based. Java provides a software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

Java code can be run on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode.

**The JVM is specific to each platform and is responsible for translating the bytecode into machine code that can be executed by the host system.**

This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere(WORA).

**Operator Overloading**: If we create two or more members having the same name but different in number or type of parameter, it is known as opeartor overloading.

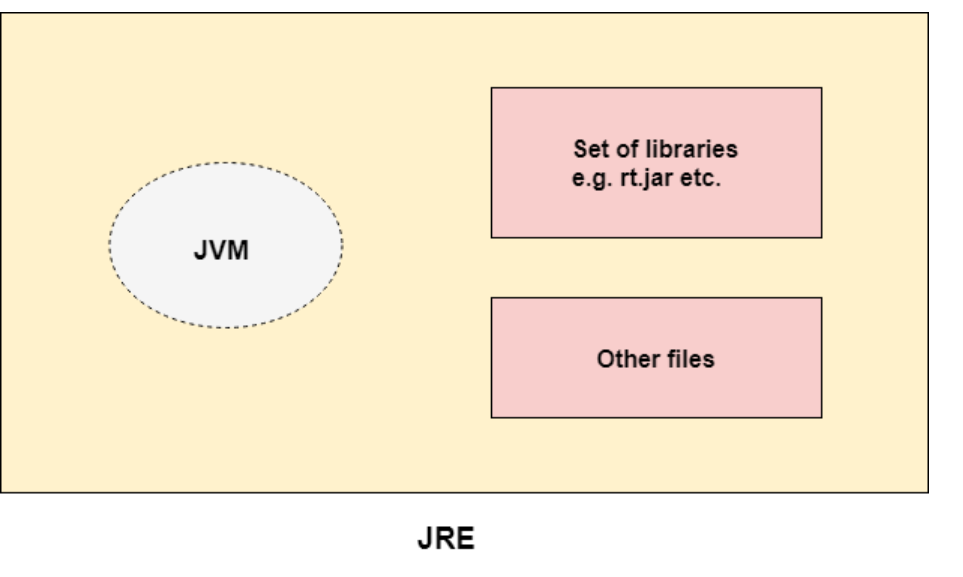
JVM

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.

JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each [OS](https://www.javatpoint.com/os-tutorial) is different from each other. However, Java is platform independent.

JRE

JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.



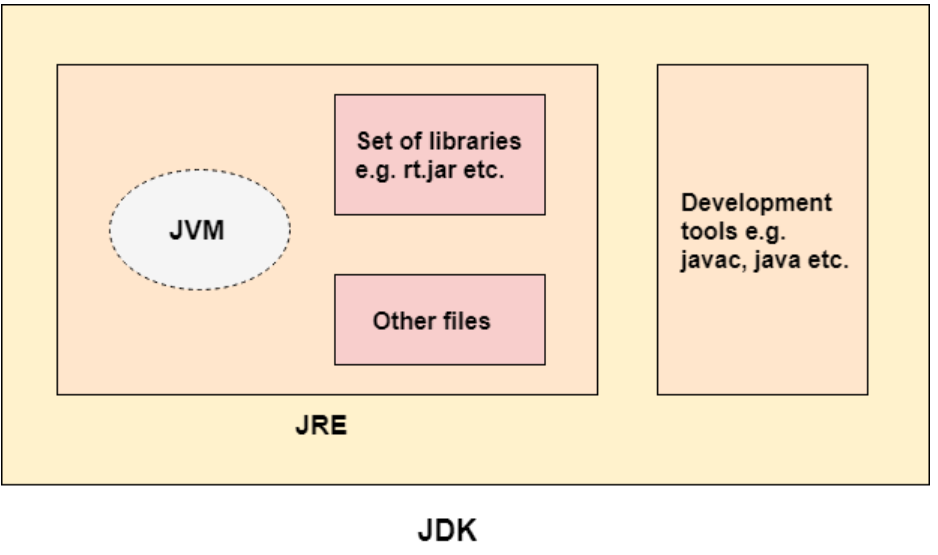
**JDK**

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and [applets](https://www.javatpoint.com/java-applet). It physically exists. It contains JRE + development tools.

JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

* Standard Edition Java Platform
* Enterprise Edition Java Platform
* Micro Edition Java Platform

The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.



Java Variables

A variable is a container which holds the value while the [Java program](https://www.javatpoint.com/simple-program-of-java) is executed. A variable is assigned with a data type.

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

**Variable** is name of reserved area allocated in memory. In other words, it is a name of memory location. It is a combination of "vary + able" that means its value can be changed.

### **Types of Variables**

There are three types of variables in [Java](https://www.javatpoint.com/java-tutorial):

* local variable
* instance variable
* static variable

#### **1) Local Variable**

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

#### **2) Instance Variable**

A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as [static](https://www.javatpoint.com/static-keyword-in-java).

It is called instance variable because its value is instance specific and is not shared among instances.

#### **3) Static variable**

A variable which is declared as static is called static variable. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

Data Types in Java

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

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



# Operators in Java

**Operator** in [Java](https://www.javatpoint.com/java-tutorial) is a symbol which is used to perform operations. For example: +, -, \*, / etc.

There are many types of operators in Java which are given below:

* Unary Operator,
* Arithmetic Operator,
* Shift Operator,
* Relational Operator,
* Bitwise Operator,
* Logical Operator,
* Ternary Operator and
* Assignment Operator.

## **Java Operator Precedence**

|  |  |  |
| --- | --- | --- |
| **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 main () method

# The main() is the starting point for JVM to start execution of a Java program. Without the main() method, JVM will not execute the program. The syntax of the main() method is:

# Java Main Method

**public:** It is an access specifier. We should use a public keyword before the main() method so that JVM can identify the execution point of the program. If we use private, protected, and default before the main() method, it will not be visible to JVM.

**static:** You can make a method static by using the keyword static. We should call the main() method without creating an object. Static methods are the method which invokes without creating the objects, so we do not need any object to call the main() method.

**void:** In Java, every method has the return type. Void keyword acknowledges the compiler that main() method does not return any value.

**main():** It is a default signature which is predefined in the JVM. It is called by JVM to execute a program line by line and end the execution after completion of this method. We can also overload the main() method.

### **Execution Process**

First, JVM executes the static block, then it executes static methods, and then it creates the object needed by the program. Finally, it executes the instance methods. JVM executes a static block on the highest priority basis. It means JVM first goes to static block even before it looks for the main() method in the program.

### **Q) Does constructor return any value?**

Yes, it is the current class instance (You cannot use return type yet it returns a value).

# Java static keyword

The **static keyword** in [Java](https://www.javatpoint.com/java-tutorial) is used for memory management mainly. We can apply static keyword with [variables](https://www.javatpoint.com/java-variables), methods, blocks and [nested classes](https://www.javatpoint.com/java-inner-class). The static keyword belongs to the class than an instance of the class.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block
4. Nested class

## **1) Java static variable**

If you declare any variable as static, it is known as a static variable.

* The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
* The static variable gets memory only once in the class area at the time of class loading.

# Java Type Casting

Type casting is when you assign a value of one primitive data type to another type.

In Java, there are two types of casting:

* **Widening Casting** (automatically) - converting a smaller type to a larger type size  
  byte -> short -> char -> int -> long -> float -> double
* **Narrowing Casting** (manually) - converting a larger type to a smaller size type  
  double -> float -> long -> int -> char -> short -> byte

## **Widening Casting**

Widening casting is done automatically when passing a smaller size type to a larger size type:

int myInt = 9;

double myDouble = myInt; // Automatic casting: int to double

## **Narrowing Casting**

Narrowing casting must be done manually by placing the type in parentheses in front of the value:

double myDouble = 9.78;

int myInt = (int) myDouble; // Manual casting: double to int

# Upcasting Vs Downcasting in Java

Typecasting is one of the most important concepts which basically deals with the conversion of one data type to another datatype implicitly or explicitly. In this article, the concept of the typecasting for objects is discussed.

Just like the datatypes, the objects can also be typecasted. However, in objects, there are only two types of objects (i.e.) parent object and child object. Therefore, typecasting of objects basically mean that one type of object (i.e.) child or parent to another. There are two types of typecasting. They are:

1. **Upcasting:** Upcasting is the [typecasting](https://www.geeksforgeeks.org/type-conversion-java-examples/) **of a child object to a parent object**. Upcasting can be done implicitly. Upcasting gives us the flexibility to access the parent class members but it is not possible to access all the child class members using this feature. Instead of all the members, we can access some specified members of the child class. For instance, we can access the overridden methods.
2. **Downcasting:** Similarly, downcasting means the typecasting of a **parent object to a child object**. Downcasting cannot be implicitly.

# Inheritance in Java

1. [Inheritance](https://www.javatpoint.com/inheritance-in-java)
2. [Types of Inheritance](https://www.javatpoint.com/inheritance-in-java#inheritancetypes)
3. [Why multiple inheritance is not possible in Java in case of class?](https://www.javatpoint.com/inheritance-in-java#inheritancenotmultiple)

**Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of [OOPs](https://www.javatpoint.com/java-oops-concepts) (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new [classes](https://www.javatpoint.com/object-and-class-in-java) that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a parent-child relationship.

### **Why use inheritance in java**

* For [Method Overriding](https://www.javatpoint.com/method-overriding-in-java) (so [runtime polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java) can be achieved).
* For Code Reusability.

## **Types of inheritance in java**

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.

#### **Note: Multiple inheritance is not supported in Java through class.**



# Aggregation in Java

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

Consider a situation, Employee object contains many informations such as id, name, emailId etc. It contains one more object named address, which contains its own informations such as city, state, country, zipcode etc. as given below.

### **Why use Aggregation?**

* For Code Reusability.
* Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.

# Java Polymorphism

# Method Overloading in Java

# If a [class](https://www.javatpoint.com/object-and-class-in-java) has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

## **Advantage of method overloading**

Method overloading increases the readability of the program.

### **Different ways to overload the method**

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

#### **In Java, Method Overloading is not possible by changing the return type of the method only.**

### **Can we overload java main() method?**

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But [JVM](https://www.javatpoint.com/jvm-java-virtual-machine) calls main() method which receives string array as arguments only.

#### **Java method overriding is mostly used in Runtime Polymorphism which we will learn in next pages.**

### **Can we override static method?**

No, a static method cannot be overridden. It can be proved by runtime polymorphism, so we will learn it later.

### **Why can we not override static method?**

It is because the static method is bound with class whereas instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area.

No, **we** cannot **override static methods** because **method overriding** is based on dynamic binding at runtime and the **static methods** are bonded using **static** binding at compile time.

### **Can we override java main method?**

No, because the main is a static method.

## **Difference between method Overloading and Method Overriding in java**

|  |  |
| --- | --- |
| Method Overloading | Method Overriding |
| 1)Method overloading is used to increase the readability of the program.  2)Method overloading is the example of compile time polymorphism. | 1)Method overriding is used to provide the specific implementation of the method that is already provided by its super class.2)Method overriding is the example of run time polymorphism. |

# Access Modifiers in Java

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the **class**. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the **package**. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the **package** and outside the package through **child class**. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is **everywhere**. It can be accessed from within the class, outside the class, within the package and outside the package.

# Java String

In [Java](https://www.javatpoint.com/java-tutorial), string is basically an object that represents sequence of char values. An [array](https://www.javatpoint.com/array-in-java) of characters works same as Java string. For example:

1. **char**[] ch={'j','a','v','a','t','p','o','i','n','t'};
2. String s=**new** String(ch);

Is same as

String s = “javatpoint”;

**Java String** class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The java.lang.String class implements Serializable, Comparable and CharSequence [interfaces](https://www.javatpoint.com/interface-in-java).



## **CharSequence Interface**

The CharSequence interface is used to represent the sequence of characters. String, [StringBuffer](https://www.javatpoint.com/StringBuffer-class) and [StringBuilder](https://www.javatpoint.com/StringBuilder-class) classes implement it. It means, we can create strings in java by using these three classes.



The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.

We will discuss immutable string later. Let's first understand what is String in Java and how to create the String object.

### What is String in java

Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

### How to create a string object?

There are two ways to create String object:

1. By string literal
2. By new keyword

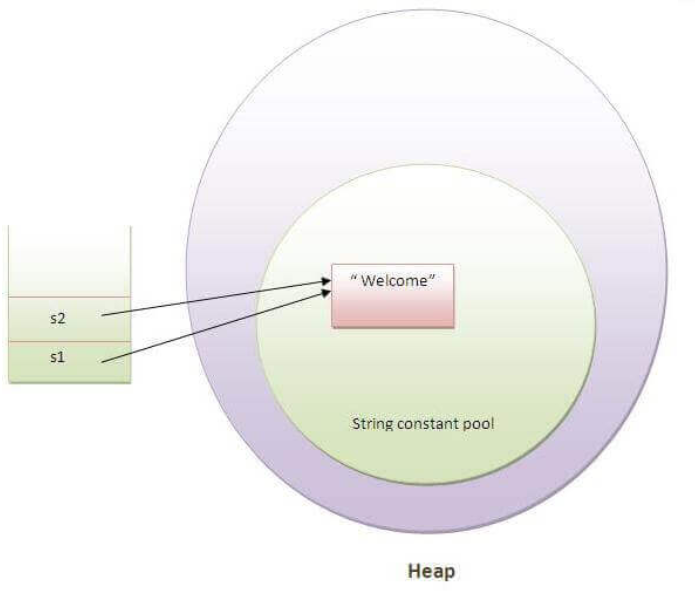
### 1) String Literal

Java String literal is created by using double quotes. For Example:

String s="welcome";

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

1. String s1="Welcome";
2. String s2="Welcome";//It doesn't create a new instance

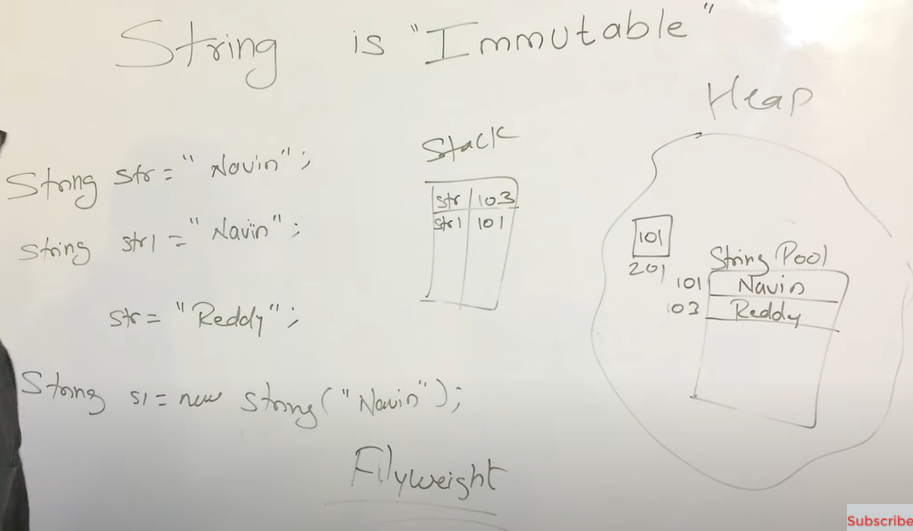


In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool, that is why it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.

#### **Note: String objects are stored in a special memory area known as the "string constant pool".**

Why Java uses the concept of String literal?

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).



### 2) By new keyword

Diagram

Description automatically generated

1. String s=**new** String("Welcome");//creates two objects and one reference variable

In such case, [JVM](https://www.javatpoint.com/jvm-java-virtual-machine) will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

# Immutable String in Java

In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable.

Once string object is created its data or state can't be changed but a new string object is created.

Let's try to understand the immutability concept by the example given below:

1. **class** Testimmutablestring{
2. **public** **static** **void** main(String args[]){
3. String s="Sachin";
4. s.concat(" Tendulkar");//concat() method appends the string at the end
5. System.out.println(s);//will print Sachin because strings are immutable objects
6. }
7. }

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

Output:Sachin

Now it can be understood by the diagram given below. Here Sachin is not changed but a new object is created with sachintendulkar. That is why string is known as immutable.



As you can see in the above figure that two objects are created but s reference variable still refers to "Sachin" not to "Sachin Tendulkar".

But if we explicitely assign it to the reference variable, it will refer to "Sachin Tendulkar" object.For example:

1. **class** Testimmutablestring1{
2. **public** **static** **void** main(String args[]){
3. String s="Sachin";
4. s=s.concat(" Tendulkar");
5. System.out.println(s);
6. }
7. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Testimmutablestring1)

Output:Sachin Tendulkar

In such case, s points to the "Sachin Tendulkar". Please notice that still sachin object is not modified.

### Why string objects are immutable in java?

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Because java uses the concept of string literal. Suppose there are 5 reference variables,  all referes to one object "sachin". **If one reference variable changes the value of the object,**  **it will be affected to all the reference variables**. That is why string objects are immutable in java. Java StringBuffer class Java StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is  same as String class except it is mutable i.e. it can be changed. **Note: Java StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order.**Java StringBuilder class Java StringBuilder class is used to create mutable (modifiable) string. The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5. Exception Handling in Java 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 this page, we will learn about Java exceptions, its type and the difference between checked and  unchecked exceptions. **What is Exception in Java** **Dictionary Meaning:** Exception is an abnormal condition.  In Java, an exception is an event that disrupts the normal flow of the program.  It is an object which is thrown at runtime. **What is Exception Handling** Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException,  IOException, SQLException, RemoteException, etc. Advantage of Exception Handling The core advantage of exception handling is **to maintain the normal flow of the application**.  An exception normally disrupts the normal flow of the application that is why we use exception handling. **Hierarchy of Java Exception classes** The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:  hierarchy of exception handling Types of Java Exceptions There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:   1. Checked Exception 2. Unchecked Exception 3. Error  **Difference between Checked and Unchecked Exceptions**1) Checked Exception The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time. 2) Unchecked Exception The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime. 3) Error Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.  There are 5 keywords which are used in handling exceptions in Java.   |  |  | | --- | --- | | **Keyword** | **Description** | | 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. | | 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. | | finally | The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not. | | throw | The "throw" keyword is used to throw an exception. | | 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. |  **Objects and Classes in Java**An object in Java is the physical as well as a logical entity, whereas,a class in Java is a logical entity only.An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. A class is a group of objects which have common properties.  It is a template or blueprint from which objects are created.  It is a logical entity. It can't be physical.  A class in Java can contain:   * **Fields** * **Methods** * **Constructors** * **Blocks** * **Nested class and interface**  **Static Binding and Dynamic Binding** Connecting a method call to the method body is known as binding.  There are two types of binding   1. Static Binding (also known as Early Binding). 2. Dynamic Binding (also known as Late Binding).   Static vs. Dynamic Binding in java **Collections in Java** The **Collection in Java** is a framework that provides an architecture to store and manipulate the  group of objects. **What is Collection in Java** A Collection represents a single unit of objects, i.e., a group. **What is a framework in Java**  * It provides readymade architecture. * It represents a set of classes and interfaces. * It is optional.  **What is Collection framework** The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:   1. Interfaces and its implementations, i.e., classes and 2. Algorithm   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).  Hierarchy of Java Collection framework **Collection Interface** The Collection interface is the interface which is implemented by all the classes in the collection  framework. It declares the methods that every collection will have. In other words, we can say that the  Collection interface builds the foundation on which the collection framework depends. **List Interface** List interface is the child interface of Collection interface. It inhibits a list type data structure  in which we can store the ordered collection of objects. It can have duplicate values.  List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack. **ArrayList** The ArrayList class implements the List interface. It uses a dynamic array to store the duplicate  element of different data types. The ArrayList class maintains the insertion order and is  non-synchronized. The elements stored in the ArrayList class can be randomly accessed. **LinkedList** LinkedList implements the Collection interface. It uses a doubly linked list internally to store  the elements. It can store the duplicate elements. It maintains the insertion order and is not  synchronized. In LinkedList, the manipulation is fast because no shifting is required.   |  |  | | --- | --- | | **ArrayList** | **LinkedList** | | 1) ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly linked list** to store the elements. | | 2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the bits are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. | | 3) An ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. | | 4) ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |  **Vector** Vector uses a dynamic array to store the data elements. It is similar to ArrayList. However, It is  synchronized and contains many methods that are not the part of Collection framework. **Stack** The stack is the subclass of Vector. It implements the last-in-first-out data structure, i.e., Stack.  The stack contains all of the methods of Vector class. **Queue Interface** Queue interface maintains the first-in-first-out order. It can be defined as an ordered list that is  used to hold the elements which are about to be processed. There are various classes like  PriorityQueue, Deque, and ArrayDeque which implements the Queue interface. **PriorityQueue** The PriorityQueue class implements the Queue interface. It holds the elements or objects which are  to be processed by their priorities. PriorityQueue doesn't allow null values to be stored in the queue. **Deque Interface** Deque interface extends the Queue interface. In Deque, we can remove and add the elements from  both the side. Deque stands for a double-ended queue which enables us to perform the  operations at both the ends. **ArrayDeque** ArrayDeque class implements the Deque interface. It facilitates us to use the Deque.  Unlike queue, we can add or delete the elements from both the ends.  ArrayDeque is faster than ArrayList and Stack and has no capacity restrictions. **Set Interface** Set Interface in Java is present in java.util package. It extends the Collection interface.  It represents the unordered set of elements which doesn't allow us to store the duplicate items.  We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet,  and TreeSet. **HashSet** HashSet class implements Set Interface. It represents the collection that uses a hash table for  storage. Hashing is used to store the elements in the HashSet. It contains unique items. **LinkedHashSet** LinkedHashSet class represents the LinkedList implementation of Set Interface. It extends the  HashSet class and implements Set interface. Like HashSet, It also contains unique elements.  It maintains the insertion order and permits null elements. **SortedSet Interface** SortedSet is the alternate of Set interface that provides a total ordering on its elements.  The elements of the SortedSet are arranged in the increasing (ascending) order.  The SortedSet provides the additional methods that inhibit the natural ordering of the elements. **TreeSet** Java TreeSet class implements the Set interface that uses a tree for storage. Like HashSet,  TreeSet also contains unique elements. However, the access and retrieval time of TreeSet  is quite fast. The elements in TreeSet stored in ascending order.  1)what is a databaase?  1.A database is an organized collection of data, generally stored and accessed electronically  from a computer system. Where databases are more complex they are often developed using formal design and modeling techniques.  2. A DATABASE is a collection of data stored in a format that can easily be accessed.  2)what is html?  Hypertext Markup Language is the standard markup language for documents designed to be displayed  in a web browser. It can be assisted by technologies such as Cascading Style Sheets and scripting languages such as JavaScript.  3) class A {    int i = 10;    public void printValue() {  System.out.println("Value-A");  }  }  class B extends A {    int i = 12;    public void printValue() {    System.out.println("Value-B");    }  }    public class Test {    public static void main(String[] args) {    A a = new B();    a.printValue();    System.out.println(a.i);  }  }  ans). If you create object of subclass with reference of super class like ( A a = new B();)  then subclass method and super class variable will be executed.  4) What is Object Oriented?  Object Oriented means organizing software as a combination of different types of objects that incorporates both data & behaviour..  5) what is OOP's ?  Object Oriented Programming is a methodology that simplify software development and maintaince by providing some rules..  6) What is a Variable?  Variable is a piece of memeory that can contain a data value..  7) What is a Array?  Array is an object which contains elements of a similar data type..  8) What is a String?  The CharSequence interface is used to represent the sequence of characters.  String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in java by using these three classes.  Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character '\0'.  9) What is a Class?  A Class , in the context of java, are templates that are used to create objects and to define object data types and methods..  10) What is a Object?  A object is a instance of Class..  11) What is Encapsulation?  It is a technique of making the fields in aclass private and providing access to the fields  via public methods. If a field is declared private ,  it cannot be accessed by anyone outside  the class, thereby hiding the fields within the class. For this reasson Encapsulation is also  referred to as data hiding..  Encapsulation is the mechanism that binds together code and data it manipulates  and keep both safe from outside interference and misuse..  You can implement Encapsultion using Access Modifiers (Public, Protected & Private)..  Encapsulation solves the problem in Implementation level..  For simplicity, encapsulation means hiding data using getters and setters.  12) What is PolyMorphisim?  Polymorphism in Java is a concept by which we can perform a single action in different ways. ... We can perform polymorphism in java by method overloading and method overriding.  If you overload a static method in Java, it is the example of compile time polymorphism.  The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.  Overriding is the example of run time Polymorphism.  Overloading is an example of compiler-time polymorphism and overriding is an example of run time polymorphism.  13) What is overloading?  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 in Java,  that allows a class to have more than one constructor having different argument lists.  14) What is overRiding?  Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes.  If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.  15) What is Abstarction?  Abstraction is a process of hiding the implementation details from the user.  ?nly the functionality will be provided to the user.  In Java, abstraction is achieved using abstract classes and interfaces. ..  16) What is Inheritance?  It is the mechanism in java by which one class is allow to inherit  the features(fields and methods) of another class. ...  The subclass can add its own fields and methods in addition to the superclass fields and methods.  17) Types of Inheritance?  Single Inheritance.  Multiple Inheritance (Through Interface)  Multilevel Inheritance.  Hierarchical Inheritance.  Hybrid Inheritance (Through Interface)    <img src ="https://simplesnippets.tech/wp-content/uploads/2018/04/java-types-of-inheritance.jpg" alt="Image not found"/>  18) What is a Abstarct class in java?  If in a class you declare a method as a abstarct , then you should keep the 'abstract' before the class, and in abstract class we can also have a methods without abstarct...    19) What is a Interface in java ?  Using Interface we can have 100% abstarction....  Every method writing in a Interface by default it is a absract..And interface is not a class...  If you implements the interface you have to implement all the methods in interface... |