exception handling , collections, Abstract Classes and Interfaces, Constructors, File IO and Serialization, Collections (List, Map, Set), Access Specifiers, Exceptions (Checked, Unchecked), Generics, Java Keywords (Static, Final, Volatile, Synchronized, Transient), JVM and Memory Management, Multithreading and Synchronization, JSP/ Servlets,JMS

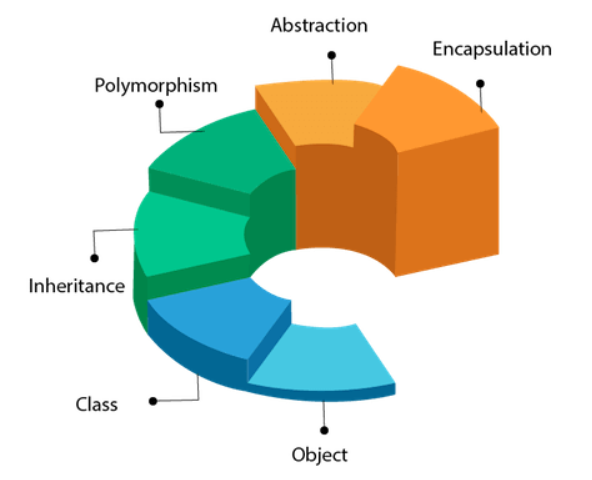
JDK: The JDK (Java Development Kit) is a software development kit that develops applications in Java. Along with JRE, the JDK also consists of various development tools (Java Debugger, JavaDoc, compilers, etc.)

JVM: The Java Virtual Machine (JVM) is a platform-independent abstract machine that has three notions in the form of specifications. This document describes the requirement of JVM implementation.  It is responsible for providing all the implementations to the JRE.

JRE: The Java Runtime Environment (JRE) is an implementation of JVM. It is a type of software package that provides class libraries of Java, JVM, and various other components for running the applications written in Java programming. JRE has a major responsibility for creating an environment for the execution of code.

Simula is considered the first object-oriented programming language

The main aim of object-oriented programming is to implement real-world entities



Advantage of OOPs over Procedure-oriented programming language

1. Makes development and maintenance easier
2. OOPs provides data hiding, whereas in procedural prog. Global data can be accessed from anywhere.
3. Provides the ability to simulate real-world event much more effectively.

**OBJECT**

Object means a real-world entity which has state and behaviour such as a pen, chair. An Object can be defined as an instance of a class. Objects can communicate without knowing the details of each other's data.

**Example:** A dog is an object because it has states like colour, name, breed, etc. as well as behaviours like wagging the tail, barking, eating, etc.

State: represents data of an obj.

Behaviour: represents behaviour of obj.

**CLASS**

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. Class in java contains fields, methods, constructors, blocks, nested class.

Instance variable: A variable which is created inside the class but outside the method is known as an instance variable. It gets memory at runtime when an object or instance is created.

Advantage of Method: Code Reusability, Code Optimization

3 ways to initialize object in Java: By reference variable, By method, By constructor example??

ways to create an object in java. They are: By new keyword, By newInstance(), method, By clone() method, By deserialization, By factory method etc.??

**METHODS**

It tells about behaviour of the object. Method declaration provides information about method attributes, such as visibility, return-type, name, and arguments.

**Access Specifier:**

1. *Public:* method is accessible by all classes when we use public specifier.
2. *Private:* The method is accessible only in the classes in which it is defined.
3. *Protected:* method is accessible within the package or subclass in the different package.
4. *Default:* when no access specifier is used, it is set as default. Visible only from same package.

Some **pre-defined methods** are length(), equals(), compareTo(), sqrt(), etc. When we call any of the predefined methods in our program, a series of codes related to the corresponding method runs in the background that is already stored in the library.

The method written by the user or programmer is known as a **user-defined method**. These methods are modified according to the requirement.

A method that belongs to a class rather than an instance of a class is known as a **static method**. The main advantage of a static method is that we can call it without creating an object. It can access static data members and also change the value of it. It is used to create an instance method. It is invoked by using the class name. Basically, static is used for a constant variable or a method that is same for every instance of a class. The best example of a static method is the **main()** method.

The method that does not has method body is known as **abstract method**. It always declared in the abstract class. It means the class itself must be abstract if it has abstract method. To create an abstract method, we use the keyword abstract.

**CONSTRUCTOR**

It is called when an instance of the class is created. Special type of method used to initialize the object. Compiler provides a default constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

The constructors in Java cannot be static because **if the constructors are marked as static, they cannot be called from the child class**; thus, the child class's object will not be created. The program will not be compiled and throw a compile-time error.

If we are declaring a constructor as abstract as we have to implement it in a child class, but we know **a constructor is called implicitly when the new keyword is used so it can't lack a body and also it cannot be called as a normal method**.

 A java constructor is internally final. Constructors cannot be inherited in Java therefore, there is no need to write final before constructors.

**Using the synchronized keyword with a constructor is a syntax error**. Synchronizing constructors doesn't make sense, because only the thread that creates an object should have access to it while it is being constructed.

It can also be overloaded like Java methods. Constructor **returns current class instance.**

**STATIC:**

* 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.
* A static method belongs to the class rather than the object of a class.
* A static method can be invoked without the need for creating an instance of a class.
* A static method can access static data member and can change the value of it.
* The static methods cannot be overridden because static methods are resolved at the compilation time of the java program and we know very well the method overriding is the runtime polymorphism.

Two main restrictions for the static method,

1. The static method cannot use non static data member or call non-static method directly.
2. this and super cannot be used in static context.

**this keyword is the reference of the current object** but without creating an object we can access the static method this will cause an error .

### **Q) Why is the Java main method static?**

Ans) It is because the object is not required to call a static method. If it were a non-static method, [JVM](https://www.javatpoint.com/jvm-java-virtual-machine) creates an object first then call main() method that will lead the problem of extra memory allocation. The main method is always static because static members are those methods that belong to the classes, not to an individual object. So if the main method will not be static then for every object, It is available. And that is not acceptable by JVM. JVM calls the main method based on the class name itself. Not by creating the object.

### **Q) Can we execute a program without main() method?**

Ans) No, one of the ways was the static block, but it was possible till JDK 1.6. Since JDK 1.7, it is not possible to execute a Java class without the [main method](https://www.javatpoint.com/java-main-method).

this keyword:

In Java, **this** is a reference variable that refers to the current object.

**Inheritance**

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviours of a parent object. ‘IS-A’ relationship is another name for inheritance.

The idea behind inheritance in Java is that you can create new classes 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.

Why use inheritance in java

For Method Overriding(so runtime polymorphism can be achieved).

For Code Reusability

The **extends** keyword indicates that you are making a new class that derives from an existing class.

There can be three types of inheritance in java: single, multilevel and hierarchical.

Multiple inheritance is not supported in Java through class.  
Java doesn’t support multiple inheritances in classes because it can lead to **diamond problem**. The ambiguity as the compiler doesn’t know which superclass method to execute if both the class B and C have same method.Ambiguity that arises when two classes B and C inherit from A, and class D inherits from both B and C.

When a class inherits another class, it is known as a single inheritance.

When there is a chain of inheritance, it is known as multilevel inheritance.

When two or more classes inherits from a single class, it is known as hierarchical inheritance.

# Aggregation

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

# Method Overloading

If a class has multiple methods having same name but different parameters.

Method overloading increases the readability of the program.

### **Q) Why Method Overloading is not possible by changing the return type of method only?**

In java, method overloading is not possible by changing the return type of the method only because of ambiguity. Compile time error occurs.  It is not possible to decide to execute which method based on the return type

**Method Overriding**

If subclass has same method as declared in parent class, it is known as method overriding.

Method overriding is used for runtime polymorphism.

**A static method cannot be overridden**. 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.

# Super Keyword

It is a reference variable which is used to refer immediate parent class object.

# Final Keyword

It is used to restrict the user.

If you make any variable as final, you cannot change the value of final variable (It will be constant).

If you make any method as final, you cannot override it.

If you make any class as final, you cannot extend it.

# Polymorphism

We can perform a single action in different ways.

There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism.

**\*\*\*Runtime polymorphism** or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

If the reference variable of Parent class refers to the object of Child class, it is known as upcasting.

class A{}   
class B extends A{}   
A a=new B();//upcasting

When Subclass type refers to the object of Parent class, it is known as downcasting. If we perform it directly, compiler gives Compilation error. But if we use instanceof operator, downcasting is possible.

Connecting a method call to the method body is known as binding.

When type of the object is determined at compiled time(by the compiler), it is known as static binding. If there is any private, final or static method in a class, there is static binding

When type of the object is determined at run-time, it is known as dynamic binding.

The java **instanceof operator** is used to test whether the object is an instance of the specified type. If we apply the instanceof operator with any variable that has null value, it returns false.

### **Abstraction**

It is a process of hiding the implementation details and showing only functionality to the user. Abstraction lets you focus on what the object does instead of how it does it.

### **Abstract class in Java**

A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

A **factory method** is a method that returns the instance of the class.

If you are extending an abstract class that has an abstract method, you must either provide the implementation of the method or make this class abstract.

**Interface**It is a blueprint of a class. It is a mechanism to achieve abstraction multiple inheritance in Java. All the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. Class implements interface.

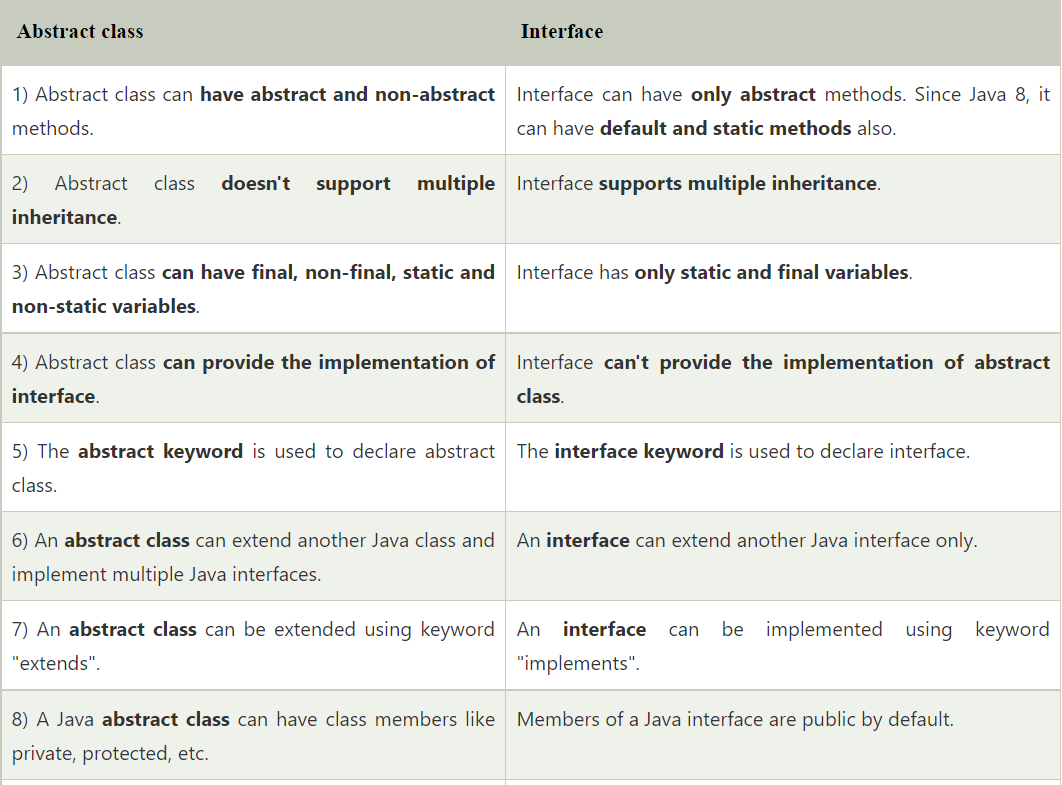
* Interfaces can extend multiple interfaces, and classes can implement multiple interfaces

## Q) Multiple inheritance is not supported through class in java, but it is possible by an interface, why?

As we have explained in the inheritance chapter, multiple inheritance is not supported in the case of [class](https://www.javatpoint.com/object-and-class-in-java) because of ambiguity. However, it is supported in case of an interface because there is no ambiguity. It is because its implementation is provided by the implementation class.

## Q) What is marker or tagged interface?

An interface which has no member is known as a marker or tagged interface, for example, [Serializable](https://www.javatpoint.com/serialization-in-java), Cloneable, Remote, etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

****

**Packages**

A java package is a group of similar types of classes, interfaces and sub-packages. There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Java package provides access protection.  
Java package removes naming collision.

# Access Modifiers

specifies the accessibility or scope of a field, method, constructor, or class.

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.
5. private makes a member accessible only from within its own class
6. protected makes a member accessible only to classes in the same package or subclass of the class
7. default access is very similar to protected (make sure you spot the difference) default access makes a member accessible only to classes in the same package.
8. public means that all other classes regardless of the package that they belong to, can access the member (assuming the class itself is visible)
9. final makes it impossible to extend a class, when applied to a method it prevents a method from being overridden in a subclass, when applied to a variable it makes it impossible to reinitialise a variable once it has been initialised
10. abstract declares a method that has not been implemented.
11. transient indicates that a variable is not part of the persistent state of an object.
12. volatile indicates that a thread must reconcile its working copy of the field with the master copy every time it accesses the variable.

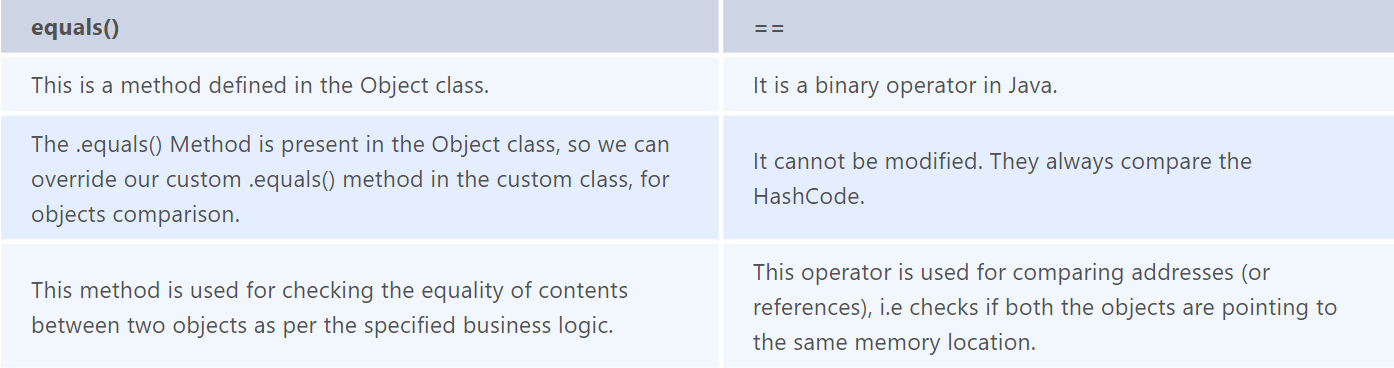
non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc.

Comparator interface

A comparator interface is used to order the objects of user-defined classes. A comparator object is capable of comparing two objects of the same class**.**

### **Data Encapsulation**

* Data Encapsulation is an Object-Oriented Programming concept of hiding the data attributes and their behaviours in a single unit.
* It helps developers to follow modularity while developing software by ensuring that each object is independent of other objects by having its own methods, attributes, and functionalities.
* It is used for the security of the private properties of an object and hence serves the purpose of data hiding  
  String str1 = new String("InterviewBit");  
  String str2 = "InterviewBit";  
  System.out.println(str1 == str2); // false  
  System.out.println(str1.equals(str2)); //true



**Shallow copy** - The shallow copy only creates a new reference and points to the same object.   
 Rectangle obj2 = obj1;

**Deep Copy** - In a deep copy, we create a new object and copy the old object value to the new object.  
Rectangle obj3 = **new** Rectangle();  
Obj3.length = obj1.length;  
Obj3.breadth = obj1.breadth;

The **clone()** will do this deep copy internally and return a new object. And to do this we need to write only 1 line of code. That is - **Rectangle obj2 = obj1.clone();**

In Java, a string is basically immutable i.e. it cannot be modified. After its declaration, it continues to stay in the string pool as long as it is not removed in the form of garbage. In other words, a string resides in the heap section of the memory for an unregulated and unspecified time interval after string value processing is executed

**MIN\_PRIORITY:** It has an integer value assigned with 1.  
**MAX\_PRIORITY:** It has an integer value assigned with 10.  
**NORM\_PRIORITY:**It has an integer value assigned with 5.

In Java, Thread with MAX\_PRIORITY gets the first chance to execute. But the default priority for any thread is NORM\_PRIORITY assigned by JVM.

* The ‘**throw**’ keyword is used to manually throw the exception to the calling method.
* And the ‘**throws**’ keyword is used in the function definition to inform the calling method that this method throws the exception. So if you are calling, then you have to handle the exception.

It is because the 0 index array avoids the extra arithmetic operation to calculate the memory address.  
 If starts with 0 index 🡪 **[Base Address + (index \* no\_of\_bytes)]** If starts with 1 index 🡪 **[Base Address + ((index-1) \* no\_of\_bytes)]**

ArrayList will allocate the new ArrayList of Size (current size + half of the current size). And add the old elements into the new when an element is added to array list.

### **Java thread lifecycle** Java thread life cycle is as follows:

* **New** – When the instance of the thread is created and the start() method has not been invoked, the thread is considered to be alive and hence in the NEW state.
* **Runnable** – Once the start() method is invoked, before the run() method is called by JVM, the thread is said to be in RUNNABLE (ready to run) state. This state can also be entered from the Waiting or Sleeping state of the thread.
* **Running** – When the run() method has been invoked and the thread starts its execution, the thread is said to be in a RUNNING state.
* **Non-Runnable (Blocked/Waiting)** – When the thread is not able to run despite the fact of its aliveness, the thread is said to be in a NON-RUNNABLE state. Ideally, after some time of its aliveness, the thread should go to a runnable state.
  + A thread is said to be in a Blocked state if it wants to enter synchronized code but it is unable to as another thread is operating in that synchronized block on the same object. The first thread has to wait until the other thread exits the synchronized block.
  + A thread is said to be in a Waiting state if it is waiting for the signal to execute from another thread, i.e it waits for work until the signal is received.
* **Terminated** – Once the run() method execution is completed, the thread is said to enter the TERMINATED step and is considered to not be alive.

**IMPORTANT**System.out.println('b' + 'i' + 't'); //319  
Ascii values are taken and added as the characters are in single quote.

**HEAP VS STACK**

Stack is used to store the order of method execution and local variables while the heap memory stores the objects and it uses dynamic memory allocation and deallocation.

Whenever we create objects it occupies space in the heap memory while the reference of that object creates in the stack.

**Callbyvalue Callbyreference**

Call by Value means calling a method with a parameter as value. Through this, the argument value is passed to the parameter.

While Call by Reference means calling a method with a parameter as a reference. Through this, the argument reference is passed to the parameter.

In call by value, the modification done to the parameter passed does not reflect in the caller's scope while in the call by reference, the modification done to the parameter passed are persistent and changes are reflected in the caller's scope.

### **How to implement a queue using stack?**

A queue can be implemented using **two stacks**. Let q be the queue andstack1 and stack2 be the 2 stacks for implementing q. We know that stack supports push, pop, and peek operations and using these operations, we need to emulate the operations of the queue - enqueue and dequeue. Hence, queue q can be implemented in two methods (Both the methods use auxillary space complexity of O(n)):

**1. By making enqueue operation costly:**

* Here, the oldest element is always at the top of stack1 which ensures dequeue operation occurs in O(1) time complexity.
* To place the element at top of stack1, stack2 is used.
* **Pseudocode:**
  + **Enqueue:** Here time complexity will be O(n)

enqueue(q, data):

While stack1 is not empty:

Push everything from stack1 to stack2.

Push data to stack1

Push everything back to stack1.

* **Dequeue:** Here time complexity will be O(1)

deQueue(q):

If stack1 is empty then error else

Pop an item from stack1 and return it

**2. By making the dequeue operation costly:**

* Here, for enqueue operation, the new element is pushed at the top of stack1. Here, the enqueue operation time complexity is O(1).
* In dequeue, if stack2 is empty, all elements from stack1 are moved to stack2 and top of stack2 is the result. Basically, reversing the list by pushing to a stack and returning the first enqueued element. This operation of pushing all elements to a new stack takes O(n) complexity.
* **Pseudocode:**
  + **Enqueue:**Time complexity: O(1)

enqueue(q, data):

Push data to stack1

* **Dequeue:**Time complexity: O(n)

dequeue(q):

If both stacks are empty then raise error.

If stack2 is empty:

While stack1 is not empty:

push everything from stack1 to stack2.

Pop the element from stack2 and return it.

### **How do you implement stack using queues?**

* A stack can be implemented using two queues. We know that a queue supports enqueue and dequeue operations. Using these operations, we need to develop push, pop operations.
* Let stack be ‘s’ and queues used to implement be ‘q1’ and ‘q2’. Then, stack ‘s’ can be implemented in two ways:

**1. By making push operation costly:**

* This method ensures that the newly entered element is always at the front of ‘q1’ so that pop operation just dequeues from ‘q1’.
* ‘q2’ is used as auxillary queue to put every new element in front of ‘q1’ while ensuring pop happens in O(1) complexity.
* **Pseudocode:**
  + Push element to stack s: Here push takes O(n) time complexity.

push(s, data):

Enqueue data to q2

Dequeue elements one by one from q1 and enqueue to q2.

Swap the names of q1 and q2

* Pop element from stack s: Takes O(1) time complexity.

pop(s):

dequeue from q1 and return it.

**2. By making pop operation costly:**

* In push operation, the element is enqueued to q1.
* In pop operation, all the elements from q1 except the last remaining element, are pushed to q2 if it is empty. That last element remaining of q1 is dequeued and returned.
* **Pseudocode:**
  + Push element to stack s: Here push takes O(1) time complexity.

push(s,data):

Enqueue data to q1

* Pop element from stack s: Takes O(n) time complexity.

pop(s):

Step1: Dequeue every elements except the last element from q1 and enqueue to q2.

Step2: Dequeue the last item of q1, the dequeued item is stored in result variable.

Step3: Swap the names of q1 and q2 (for getting updated data after dequeue)

Step4: Return the result.

**Nested Class**

* The scope of a nested class is bounded by the scope of its enclosing class. Thus in below example, class *NestedClass* does not exist independently of class *OuterClass*.
* A nested class has access to the members, including private members, of the class in which it is nested. But the enclosing class does not have access to the members  
  of the nested class.
* A nested class is also a member of its enclosing class.
* As a member of its enclosing class, a nested class can be declared *private*, *public*, *protected*, or *package private*(default).