**What is JVM and is it platform independent?**

### Java Virtual Machine (JVM) is a specification that provides runtime environment in which java bytecode(.class files) can be executed.

### The JVM is not platform independent. Java Virtual Machine (JVM) provides the environment to execute the java file(. Class file). So at the end it's depends on your kernel , and kernel is differ from OS (Operating System) to OS. The JVM is used to both translate the bytecode into the machine language for a particular computer, and actually execute the corresponding machine-language instructions as well. Without the JVM, you can’t run a Java application.

### What is the difference between JDK and JVM?

### Java Development Kit (JDK) is for development purpose and JVM is a part of itto execute the java programs.

JDK provides all the tools, executables and binaries required to compile, debug and execute a Java Program. The execution part is handled by JVM to provide machine independence.

**What is the difference between JVM and JRE?**

Java Runtime Environment (JRE) is the implementation of JVM. JRE consists of JVM and java binaries and other classes to execute any program successfully. JRE doesn’t contain any development tools like java compiler, debugger etc. If you want to execute any java program, you should have JRE installed.

### Why Java is not pure Object Oriented language?

Java is not said to be pure object oriented because it support primitive types such as int, byte, short, long etc.

**What are the principle concepts of OOPs?**

* Abstraction
* Encapsulation
* Polymorphism
* Inheritance

**What is Java Package and which package is imported by default?**

A package is a namespace that organizes a set of related classes and interfaces.

A **Java package** organizes Java classes into namespaces providing a unique namespace for each type it contains.

Classes in the same package can access each other's package-private and protected members. Java packages can be stored in compressed files called JAR files, allowing classes to be downloaded faster as groups rather than individually.

In a Java source file, the package file's class or classes belong to is specified with the package keyword.

|  |  |
| --- | --- |
| java.lang | — basic language functionality and fundamental types |
| java.util | — collection data structure classes |
| java.io | — file operations |
| java.math | — multiprecisionarithmetics |
| java.nio | — the Non-blocking I/O framework for Java |
| java.net | — networking operations, sockets, DNS lookups. |
| java.security | — key generation, encryption and decryption |
| java.sql | — Java Database Connectivity(JDBC) to access databases |
| java.awt | — basic hierarchy of packages for native GUI components |
| javax.swing | — hierarchy of packages for platform-independent rich GUI components as well as Framework |
| java.text | — Provides classes and interfaces for handling text, dates, numbers, and messages in a manner independent of natural languages. |
| java.rmi | — Provides the RMI package. |
| java.time | — The main API for dates, times, instants, and durations. |

The java.lang.package is always imported by default as it is loaded internally by the JVM.

### What is overloading and overriding in java?

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

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 subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

**What is the difference between an Inner Class and a Sub-Class?**

Inner classes are in the same file, whereas subclasses can be in another file, maybe in another package. You cannot get an instance of an inner class without an instance of the class that contains it.

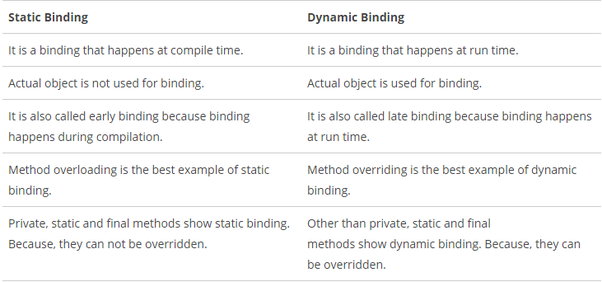
Inner classes have the methods they want, whereas subclasses have the methods of their parent class. Subclasses can of course define additional methods, but they'll always have those of their parent.

Sub-classes are used when you defines a "is-a" relationship.

### What is the difference between interface and abstract class?

### C:\Users\Admin\Desktop\pencil sketches\Abstract-Class.jpg

**What is static binding and dynamic binding?**

****

**Static 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.

**Example of static binding**

**class** Dog{

**private** **void** eat(){System.out.println("dog is eating...");}

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

   Dog d1=**new** Dog();

   d1.eat();

 }

}

**Dynamic Binding :**

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

### Example of dynamic binding

**class** Animal{

**void** eat(){System.out.println("animal is eating...");}

}

**class** Dog **extends** Animal{

**void** eat(){System.out.println("dog is eating...");}

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

   Animal a=**new** Dog();

   a.eat();

  }

}

Output: dog is eating...

|  |
| --- |
| In the above example object type cannot be determined by the compiler, because the instance of Dog is also an instance of Animal. So compiler doesn't know its type, only its base type. |

**What is data encapsulation and what’s its significance?**

Encapsulation is a concept in Object Oriented Programming for combining properties and methods in a single unit.

Encapsulation also serves data hiding purpose.

classPerson {

    privateString name;

    privateintage;

    publicString getName() {

        returnname;

    }

    publicString getAge() {

        returnage;

    }

}

the fields age and name can be only changed within the Person class.

The code won’t compile because the field name and age is marked as private. So we added two getter methods getName() and getAge() as public.

Encapsulation has a number of advantages that increase the reusability, flexibility and maintainability of the code.

* **Flexibility**: It’s more flexible and easy to change the encapsulated code with new requirements. For example, if the requirement for setting the age of a person changes, we can easily update the logic in the setter method setAge().
* **Reusability**: Encapsulated code can be reused throughout the application or across multiple applications. For example, the Person class can be reused whenever such type of object is required.
* **Maintainability**: Application code is encapsulated in separate units (classes, interfaces, methods, setters, getters, etc) so it’s easy to change or update a part of the application without affecting other parts, which reduces the time of maintenance.

**What is Java Bean Class?**

Defining class with public access, private members, public constructors, getter and setter is known as Java Bean Class.

**JavaBeans** are classes that encapsulate many objects into a single object (the bean).

**What are access modifiers?**

### Java provides access control through public, private and protected access modifier keywords. When none of these are used, it’s called default access modifier.

**What’s the benefit of using inheritance?**

Inheritance can also make application code more flexible to change because classes that inherit from a common  superclass can be used interchangeably. If the return type of a method is superclass  
  
Reusability - facility to use public methods of base class without rewriting the same  
Extensibility - extending the base class logic as per business logic of the derived class  
Data hiding - base class can decide to keep some data private so that it cannot be altered by the derived class  
Overriding-With inheritance, we will be able to override the methods of the base class so that meaningful  implementation of the base class method can be designed in the derived class.

**Why multiple inheritance is not supported in java?**  
Java supports multiple inheritance but not through classes, it supports only through its interfaces. The reason for not supporting multiple inheritance is to avoid the conflict and complexity arises due to it and keep Java a Simple Object Oriented Language

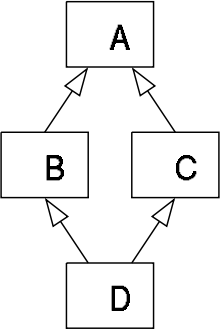
### What is the diamond problem in inheritance?

|  |
| --- |
| class A {      void display()      {          //some code      }  }    class B : public A{      void display()      {          //some code      }  }    class C : public A{      void display()      {          //some code      }  }    class D : public B, public C{      //contains two display() functions  } |

Suppose there are four classes A, B, C and D. Class B and C inherit class A. Now class B and C contains one copy of all the functions and data members of class A.

Class D is derived from Class B and C. Now class D contains two copies of all the functions and data members of class A. One copy comes from class B and another copy comes from class C.

Let’s say class A have a function with name display(). So class D have two display() functions as I have explained above. If we call display() function using class D object then ambiguity occurs because compiler gets confused that whether it should call display() that came from class B or from class C. If you will compile above program then it will show error.



This kind of problem is called diamond problem .

### What is break and continue statement?

### We can use break statement to terminate for, while, or do-while loop. We can use break statement in switch statement to exit the switch case. We can use break with label to terminate the nested loops.

* The continue statement skips the current iteration of a for, while or do-while loop. We can use continue statement with label to skip the current iteration of outermost loop.

**What is nested class?**

* In Java nested classes are classes that are defined inside another class.
* The purpose of a nested class is to clearly group the nested class with its surrounding class, signaling that these two classes are to be used together.

**How are this() and super() used with Constructor?**

**This() :**

* used to point the current class instance
* used to call constructor of same class
* private variable cannot be accessed using this()
* it canbe used with variables and methods

**super() :**

* used to call constructor of parent class
* it must be the first statement in the body of constructor
* using this constructor we can access the private variables in the super class.

**What is Serialization and Deserialization?**

**Serialization** is a process of converting an object into a sequence of bytes which can be persisted to a disk or database or can be sent through streams. The reverse process of creating object from sequence of bytes is called **deserialization**.

Java provides **Serializable** API encapsulated under java.io package for serializing and deserializing objects which include,

* java.io.serializable
* java.io.Externalizable
* ObjectInputStream
* and ObjectOutputStream etc.

**What is difference between Heap and Stack Memory?**

* Stack is used for static memory allocation and Heap for dynamic memory allocation, both stored in the computer's RAM .
* The main difference between heap and stack is that stack memory is used to store local variables and function call while heap memory is used to store objects in Java. No matter, where the object is created in code e.g. as a member variable, local variable or class variable, they are always created inside heap space in Java.
* If there is no memory left in the stack for storing function call or local variable, JVM will throw java.lang.StackOverFlowError, while if there is no more heap space for creating an object, JVM will throw java.lang.OutOfMemoryError: Java Heap Space.
* Another difference between stack and heap is that size of stack memory is a lot lesser than the size of  heap memory in Java.
* Variables stored in stacks are only visible to the owner Thread while objects created in the heap are visible to all thread

**Can we override private method in Java?**  
No, we cannot override private methods in Java as if we declare any variable ,method as private that variable or method will be visible for that class only   
  
**Can we declare the main method of our class as private?**

In java, main method must be public static in order to run any application correctly. If main method is declared as private, developer won’t get any compilation error however, it will not get executed and will give a runtime error.

**Can a class have multiple constructors?**

Yes, a class can have multiple constructors with different parameters. Which constructor gets used for object creation depends on the arguments passed while creating the objects.

**Can variables be used in Java without initialization?**

In Java, if a variable is used in a code without prior initialization by a valid value, program doesn’t compile and gives an error as no default value is assigned to variables in Java.

**Is it possible to override the main method?**

NO, because main is a static method. A static method can't be overridden in Java.

### What is the difference between Abstraction and Encapsulation?

* In Java, process of abstraction is done using interfaces, classes, abstract classes, fields, methods and variables. Everything is an abstraction.
* Encapsulation is the process of hiding information details and protecting data and behavior of an object from misuse by other objects. In Java, encapsulation is done using access modifiers (public, protected, private) with classes, interfaces, setters, getters.
* Abstraction provides a general structure of a class and leaves the details for the implementers.
* Encapsulation is to create and define the permissions and restrictions of an object and its member variables and methods.
* Abstraction is more about ‘**What**‘ a class can do. [**Idea**]
* Encapsulation is more about ‘**How**‘ to achieve that functionality. [**Implementation***].*

**ARRAY**

**What do you mean by an Array? How to create?**

An array is a very common type of data structure wherein all elements must be of the same data type. Once defined, the size of an array is fixed and cannot increase to accommodate more elements.The first element of an array starts with index zero.

## Creating Arrays

You can create an array by using the new operator with the following syntax

### Syntax

arrayRefVar = new dataType[arraySize];

The above statement does two things −

* It creates an array using new dataType[arraySize].
* It assigns the reference of the newly created array to the variable arrayRefVar.

**Advantages and disadvantages of Array?**

**Advantages:**   
  
1. It is used to represent multiple data items of same type by using only single name.  
2. It can be used to implement other data structures like linked lists, stacks, queues, trees, graphs etc.  
3. 2D arrays are used to represent matrices.

**Disadvantages:**  
  
1. We must know in advance that how many elements are to be stored in array.  
2. Array is static structure. It means that array is of fixed size. The memory which is allocated to array can not be increased or reduced.  
3. Since array is of fixed size, if we allocate more memory than requirement then the memory space will be wasted. And if we allocate less memory than requirement, then it will create problem.  
4. The elements of array are stored in consecutive memory locations. So insertions and deletions are very difficult and time consuming.

**What is the meaning of anonymous array? Explain with an example?**

An array in Java **without any name** is anonymous array. It is an array just for creating and using instantly.

* We can create an array without name, such type of nameless arrays are called anonymous array.
* The main purpose of anonymous array is just for instant use (just for one time usage) .
* Anonymous array is passed as an argument of method

**Syntax:**

// anonymous int array

newint[] { 1, 2, 3, 4};

// anonymous char array

new char[] {'x', 'y', 'z');

// anonymous String array

new String[] {"Geeks", "for", "Geeks"};

// anonymous multidimensional array

newint[][] { {10, 20}, {30, 40, 50} };

|  |
| --- |
| // Java program to illustrate the  // concept of anonymous array  classTest {      publicstaticvoidmain(String[] args)      {            // anonymous array            sum(new int[]{ 1, 2, 3});      }      Public static void sum(int[] a)      {          Int total = 0;            // using for-each loop          for(int i : a)              total = total + i;            System.out.println("The sum is:"+ total);      }  Output= 6 |

**What are “jagged” arrays in java?**

Jagged array is array of arrays such that member arrays can be of different sizes, i.e., we can create a 2-D arrays but with variable number of columns in each row. These type of arrays are also known as Jagged arrays.

**How to copy an array into another array?**

There are mainly four different ways to copy all elements of one array into another array in Java.

1. Object.clone(): Object class provides clone() method and since array in java is also an Object, you can use this method to achieve full array copy. This method will not suit you if you want partial copy of the array.
2. System.arraycopy(): Systemclass arraycopy() is the best way to do partial copy of an array. It provides you an easy way to specify the total number of elements to copy and the source and destination array index positions. For example System.arraycopy(source, 3, destination, 2, 5)will copy 5 elements from source to destination, beginning from 3rd index of source to 2nd index of destination.
3. Arrays.copyOf(): If you want to copy first few elements of an array or full copy of array, you can use this method. Obviously it’s not versatile like System.arraycopy() but it’s also not confusing and easy to use. This method internally use System arraycopy() method.
4. Arrays.copyOfRange(): If you want few elements of an array to be copied, where starting index is not 0, you can use this method to copy partial array. Again this method is also using System arraycopy method itself.

**What is the difference between ArrayIndexOutfOBounds and ArrayStoreException?**

ArrayIndexOutOfBoundsException occurs when your code tries to access an invalid index for a given array e.g. negative index or higher index than length - 1.

ArrayStoreException occurs when you have stored an element of type other than type of array.

**How to check array contains value or not?**

Using List:

|  |
| --- |
| **Public static boolean** useList(String[] arr, String targetValue) {  **Return** Arrays.asList(arr).contains(targetValue);  } |

Using Set:

|  |
| --- |
| **Public static boolean** useSet(String[] arr, String targetValue) {  Set<String> set = **new** HashSet<String>(Arrays.asList(arr));  **Return** set.contains(targetValue);  } |

Using a simple loop:

|  |
| --- |
| **Public static boolean** useLoop(String[] arr, String targetValue) {  **for**(String s: arr){  **if**(s.equals(targetValue))  **returntrue**;  }  **returnfalse**;  } |

Using Arrays.binarySearch():  
binarySearch() can ONLY be used on sorted arrays. If the array is sorted, you can use the following code to search the target element:

|  |
| --- |
| **publicstaticboolean**useArraysBinarySearch(String[] arr, String targetValue) {  **int** a = Arrays.binarySearch(arr, targetValue);  **if**(a > 0)  **returntrue**;  **else**  **returnfalse**;  } |

**String:**

**What is String in Java?**

String is a Class in java and defined in java.lang package. It’s not a primitive data type like. int and long. String class represents character Strings. String is used in almost all the Java applications and there are some interesting facts we should know about String. String in immutable and final in Java and JVM uses String Pool to store all the String objects. Some other interesting things about String is the way we can instantiate a String object using double quotes and overloading of “+” operator for concatenation.

**What are different ways to create String Object?**

There are various ways you can create a **String Object** in Java:

## Using String literal

You can create String objects with String **literal**

String str="Hello!";

## Using new keyword

This is the common way to create a String object in java.

String str1= new String("Hello!");

## Using character array

You could also convert **character array** into String here

char ch[]={ 'H','e','l','l','o','!',};

String str1=new String(ch);

**What is String subSequence method?**

This method returns a new character sequence that is a subsequence of this sequence.

## Syntax

Here is the syntax of this method −

publicCharSequencesubSequence(intbeginIndex, intendIndex)

## Parameters

Here is the detail of parameters −

* **beginIndex** − the begin index, inclusive.
* **endIndex** − the end index, exclusive

**How to convert String to char and vice versa?**

We have following two ways for char to String conversion.  
Method1:UsingtoString()method  
Method2: Using valueOf() method

**Difference between String, StringBuffer and StringBuilder?**

* StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.
* StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
* StringBuffer is less efficient than StringBuilder
* StringBuilder is *more efficient* than StringBuffer.
* String is **immutable**, if you try to alter their values, another object gets created, whereas StringBuffer and StringBuilder are **mutable** so they can change their values.

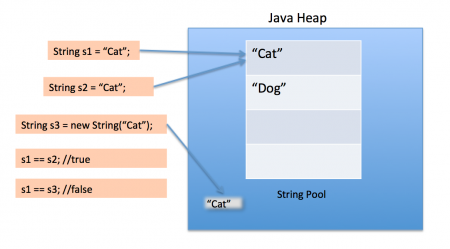
**Why String is immutable or final in Java?**

The string is Immutable in Java because String objects are cached in String pool. Since cached String literals are shared between multiple clients there is always a risk, where one client's action would affect all another client. For example, if one client changes the value of String "Test" to "TEST", all other clients will also see that value as explained in the first example. Since caching of String objects was important from performance reason this risk was avoided by making String class Immutable. At the same time, *String was made final*so that no one can compromise invariant of String class e.g. Immutability, Caching, hashcode calculation etc by extending and overriding behaviors. Another reason of *why String class is immutable* could die due to HashMap.

**What is String Pool?**

**String Pool in java** is a pool of Strings stored in **Java Heap Memory**.

Here is a diagram which clearly explains how String Pool is maintained in java heap space and what happens when we use different ways to create Strings.

[](https://www.journaldev.com/wp-content/uploads/2012/11/String-Pool-Java1.png)

**What does String intern() method do?**

The **java string intern() method** returns the interned **string**. It returns the canonical representation of **string**. It **can** be used to return **string** from pool memory, if it is created by new keyword.

**Why String is popular HashMap key in Java?**

Since **String** is immutable, its hashcode is cached at the time of creation and it doesn't need to be calculated again. This makes it a great candidate for **key** in a Map and its processing is fast than other **HashMap key** objects. This is **why String**is mostly used Object as **HashMap keys**.

**What is Java Collections Framework? List out some benefits of Collections framework?**

**Java Collections Framework** is a set of reusable data structures and algorithms which are designed to free programmers from implementing data structures themselves so that they can focus on business logics.

The Java Collections Framework provides common data structures implementations which are enough for general-purpose such as list, set, map, queue, tree, etc. These collections are high-performance, high-quality, and easy to use with very good documentation.

In addition, the Java Collections Framework provides useful and robust algorithms such as searching and sorting on collections, and the interoperability between collections and arrays.

The Java Collections Framework is a standard API which is used extensively in Java programming. It’s the standard and perfect API for manipulating collections because of the following primary benefits:

* ***Reduce programming effort:*** with the reusable and useful data structures and algorithms, the programmers do not have to re-invent the wheel, thus they can devote their time on developing application’s business.
* ***Increase program speed and quality:***the concrete collections implemented by Java Collections Framework are built for high performance and high quality, thus programmers can take this advantage into their programs.
* ***Foster software reuse:*** due to the Java Collections Framework is built into the JDK, code written using collections framework can be re-used every where among applications, libraries and APIs. That cuts development cost and increases interoperability among Java programs.

**What is the benefit of Generics in Collections Framework?**

Java 1.5 came with Generics and all collection interfaces and implementations use it heavily. Generics allow us to provide the type of Object that a collection can contain, so if you try to add any element of other type it throws compile time error.

This avoids ClassCastException at Runtime because you will get the error at compilation. Also Generics make code clean since we don’t need to use casting and **instanceof** operator. It also adds up to runtime benefit because the bytecode instructions that do type checking are not generated.

**What are the basic interfaces of Java Collections Framework?**

Collection is the root of the collection hierarchy. A collection represents a group of objects known as its elements. The Java platform doesn’t provide any direct implementations of this interface.

**Set** is a collection that cannot contain duplicate elements. This interfacemodels the mathematical set abstraction and is used to represent sets, such as the deck of cards.

**List** is an ordered collection and can contain duplicate elements. You can access any element from it’s index. List is more like array with dynamic length.

A **Map** is an object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value.

Some other interfaces are Queue, Dequeue, Iterator, SortedSet, SortedMap and ListIterator.

**What are common algorithms implemented in Collections Framework?**

Java Collections Framework also provided polymorphic algorithms to operate on the collections. These are all reusable piece of functionality provided to make job easier when working with collections. All operation as usual defied static to call directly from Collections class. Below are algorithms list:

* Sorting
* Shuffling
* Routine Data Manipulation
* Searching
* Composition
* Finding Extreme Values

**Why Collection doesn’t extend Cloneable and Serializable interfaces?**

Collection is an interface that specifies a group of objects known as *elements*. The details of how the group of elements is maintained is left up to the concrete implementations of Collection. For example, some Collection implementations like List allow duplicate elements whereas other implementations like Set don't.

A lot of the Collection implementations have a public clone method. However, it does't really make sense to include it in all implementations of Collection. This is because Collection is an abstract representation. What matters is the implementation. The semantics and the implications of either cloning or serializing come into play when dealing with the *actual* implementation; that is, **the concrete implementation should decide how it should be cloned or serialized**, or even *if* it can be cloned or serialized. In some cases, depending on what the actual backing-implementation is, cloning and serialization may not make much sense. So mandating cloning and serialization in *all* implementations is actually less flexible and more restrictive. The specific implementation should make the decision as to whether it can be cloned or serialized.

**What is difference between Array and ArrayList?**

**Resizable:**   Array is static in size that is fixed length data structure, One cannot change the length after creating the Array object.  
ArrayList is dynamic in size. Each ArrayList object  has instance variable *capacity*which indicates the size of the ArrayList. As elements are added to an ArrayList its capacity grows automatically.

**Performance:** Performance of Array and ArrayList depends on the operation you are performing :  
  
**resize() opertation** *:* Automatic resize of ArrayList will slow down the performance as it will use temporary array to copy elements from the old array to new array.  
ArrayList is internally backed by Array during resizing  as it calls the native implemented method System.arrayCopy(src,srcPos,dest,destPos,length) .  
  
**add() or get() operation** *:* adding an element or retrieving an element from the array or arraylist object has almost same  performance , as for ArrayList object these operations  run in constant time.

**Primitives :**  ArrayList cannot contains primitive data types (like int , float , double) it can only contains Object while Array can contain both primitive data types as well as objects.

**Iterating the values :** We can use iterator  to iterate through ArrayList . The iterators returned by the ArrayList class's iterator and listiterator method are fail-fast.  We can use for loop or for each loop to iterate through array .

**Type-Safety :**In Java , one can ensure Type Safety through Generics. while Array is a homogeneous data structure , thus it will contain objects of specific class or primitives of specific  data type. In array if one try to store the different data type other than the specified while creating the array object , ArrayStoreException is thrown.

**Length :**Length of the ArrayList is provided by the size() method while Each array object has the length variable which returns the length of the array.  
**Adding elements :** We can insert elements into the arraylist object using the add() method while  in array we insert elements using the assignment operator.  
**Multi-dimensional :**Array can be multi dimensional , while ArrayList is always single dimensional.

**What is difference between Enumeration and Iterator interface?**

**remove() method**

This is the main difference between *Enumeration* and *Iterator* interface. *Enumeration* only traverses the *Collection*object. You can’t do any modifications to *Collection* while traversing the *Collection* using *Enumeration*. Whereas *Iterator* interface allows us to remove an element while traversing the *Collection*object. *Iterator* has *remove()* method which is not there in the *Enumeration* interface. Below is the list of *Enumeration* and *Iterator*methods.

|  |  |
| --- | --- |
| **Iterator** | **Enumeration** |
| hasNext() | hasMoreElements() |
| next() | nextElement() |
| remove() | (Not Available) |

**Legacy Interface**

*Enumeration* is a legacy interface used to traverse only the legacy classes like *Vector*, *HashTable* and *Stack*. Whereas *Iterator* is not a legacy code which is used to traverse most of the classes in the collection framework.For example, *ArrayList*, *LinkedList*, *HashSet*, *LinkedHashSet*, *TreeSet*, *HashMap*, *LinkedHashMap*, *TreeMap* etc.

**Safe And Secure**

As *Iterator* is fail-fast in nature and doesn’t allow modification of a collection by other threads while iterating, it is considered as safe and secure than *Enumeration*.

### Enumeration Vs Iterator In Java :

|  |  |
| --- | --- |
| **Enumeration** | **Iterator** |
| Using Enumeration, you can only traverse the collection. You can’t do any modifications to collection while traversing it. | Using Iterator, you can remove an element of the collection while traversing it. |
| Enumeration is introduced in JDK 1.0 | Iterator is introduced from JDK 1.2 |
| Enumeration is used to traverse the legacy classes like Vector, Stack and HashTable. | Iterator is used to iterate most of the classes in the collection framework like ArrayList, HashSet, HashMap, LinkedList etc. |
| Methods : hasMoreElements() and nextElement() | Methods : hasNext(), next() and remove() |
| Enumeration is fail-safe in nature. | Iterator is fail-fast in nature. |
| Enumeration is not safe and secured due to it’s fail-safe nature. | Iterator is safer and secured than Enumeration. |

**What is difference between Stack and Queue?**

Stack and Queue both are the non-primitive data structures. The main differences between stack and queue are that stack uses LIFO (last in first out) method to access and add data elements whereas Queue uses FIFO (First in first out) method to access and add data elements.

| **BASIS FOR COMPARISON** | **STACK** | **QUEUE** |
| --- | --- | --- |
| Working principle | LIFO (Last in First out) | FIFO (First in First out) |
| Structure | Same end is used to insert and delete elements. | One end is used for insertion, i.e., rear end and another end is used for deletion of elements, i.e., front end. |
| Number of pointers used | One | Two (In simple queue case) |
| Operations performed | Push and Pop | Enqueue and dequeue |
| Examination of empty condition | Top == -1 | Front == -1 || Front == Rear + 1 |
| Examination of full condition | Top == Max - 1 | Rear == Max – 1 |
| Variants | It does not have variants. | It has variants like circular queue, priority queue, doubly ended queue. |
| Implementation | Simpler | Comparatively complex |

**What is difference between Comparable and Comparator interface?**

* Comparable provides **single sorting sequence**. In other words, we can sort the collection on the basis of single element such as id or name or price etc.
* Comparator provides **multiple sorting sequence**. In other words, we can sort the collection on the basis of multiple elements such as id, name and price etc.
* Comparable **affects the original class** i.e. actual class is modified
* Comparator **doesn't affect the original class** i.e. actual class is not modified.
* Comparable provides **compareTo() method** to sort elements.
* Comparator provides **compare() method** to sort elements.
* Comparable is found in **java.lang** package.
* Comparator is found in **java.util** package.
* We can sort the list elements of Comparable type by **Collections.sort(List)** method.
* We can sort the list elements of Comparator type by **Collections.sort(List,Comparator)** method.

**Multi-Threading**

**What is the difference between Process and Thread?**

* A program in execution is often referred as process. A thread is a subset(part) of the process.
* A process consists of multiple threads. A thread is a smallest part of the process that can execute concurrently with other parts(threads) of the process.
* A process is sometime referred as task. A thread is often referred as lightweight process.
* A process has it’s own address space. A thread uses the process’s address space and share it with the other threads of that process.
* A thread can communicate with other thread(of the same process) directly by using methods like wait(), notify(),notifyAll(). A process can communicate with other process by using inter-process communication.
* New threads are easily created. However the creation of new processes require duplication of the parent process.
* Threads have control over the other threads of the same process. A process does not have control over the sibling process, it has control over its child processes only.

**What are the benefits of multi-threaded programming?**

* More efficient CPU use
* Better system reliability
* Improved performance on multiprocessor computers

**What is difference between user Thread and daemon Thread?**

|  |  |
| --- | --- |
| **User Threads** | **Daemon Threads** |
| JVM waits for user threads to finish their work. It will not exit until all user threads finish their work. | JVM will not wait for daemon threads to finish their work. It will exit as soon as all user threads finish their work. |
| User threads are foreground threads. | Daemon threads are background threads. |
| User threads are high priority threads. | Daemon threads are low priority threads. |
| User threads are created by the application. | Daemon threads, in most of time, are created by the JVM. |
| User threads are mainly designed to do some specific task. | Daemon threads are designed to support the user threads. |
| JVM will not force the user threads to terminate. It will wait for user threads to terminate themselves. | JVM will force the daemon threads to terminate if all user threads have finished their work. |

**What are different states in lifecycle of Thread?**

1. New
2. Runnable
3. Blocked
4. Waiting
5. Timed Waiting
6. Terminated

**Why Thread sleep() and yield() methods are static?**

* Both yield and sleep are declared on java.lang.Thread class.
* Both sleep() and yield() are static methods and operate on current thread. It doesn't matter which thread's object you used to call this method, both these methods will always operate on current thread.
* Sleep as well as Yield is used to relinquish CPU from current thread, but at same time it doesn't release any lock held by the thread. If you also want to release locks along with releasing CPU, you should be using wait() method instead. See difference between sleep() and wait() method for more details.

**What is Deadlock? How to analyze and avoid deadlock situation?**

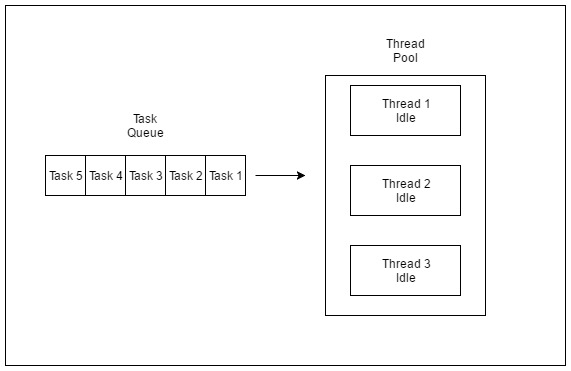
Deadlock is a programming situation where two or more threads are blocked forever, this situation arises with at least two threads and two or more resources.

To analyze a deadlock, we need to look at the java thread dump of the application, we need to look out for the threads with state as BLOCKED and then the resources it’s waiting to lock, every resource has a unique ID using which we can find which thread is already holding the lock on the object.

**What is Thread Pool? How can we create Thread Pool in Java?**

**A thread pool reuses previously created threads to execute current tasks and offers a solution to the problem of thread cycle overhead and resource thrashing.** Since the thread is already existing when the request arrives, the delay introduced by thread creation is eliminated, making the application more responsive.

* Java provides the Executor framework which is centered around the Executor interface, its sub-interface –**ExecutorService** and the class-**ThreadPoolExecutor**, which implements both of these interfaces. By using the executor, one only has to implement the Runnable objects and send them to the executor to execute.
* They allow you to take advantage of threading, but focus on the tasks that you want the thread to perform, instead of thread mechanics.
* To use thread pools, we first create a object of ExecutorService and pass a set of tasks to it. ThreadPoolExecutor class allows to set the core and maximum pool size.Therunnables that are run by a particular thread are executed sequentially.

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/tpinit.jpg)

*Thread Pool Initialization with size = 3 threads. Task Queue = 5 Runnable Objects*

**Exception Handling:**

**What is Exception & Exception Handling in Java?**

Exception is an error event that can happen during the execution of a program and disrupts its normal flow. Java provides a robust and object oriented way to handle exception scenarios, known as **Java Exception Handling**.

**What are the Exception Handling Keywords in Java?**

Java provides specific keywords for exception handling purposes,

**1.throw**– We know that if any exception occurs, an exception object is getting created and then Java runtime starts processing to handle them. Sometime we might want to generate exception explicitly in our code, for example in a user authentication program we should throw exception to client if the password is null. **throw** keyword is used to throw exception to the runtime to handle it.

**2. throws** – When we are throwing any exception in a method and not handling it, then we need to use**throws**keyword in method signature to let caller program know the exceptions that might be thrown by the method. The caller method might handle these exceptions or propagate it to it’s caller method using throws keyword. We can provide multiple exceptions in the throws clause and it can be used with main() method also.

**3. try-catch**– We use try-catch block for exception handling in our code. try is the start of the block and catch is at the end of try block to handle the exceptions. We can have multiple catch blocks with a try and try-catch block can be nested also. catch block requires a parameter that should be of type Exception.

**4. finally** – finally block is optional and can be used only with try-catch block. Since exception halts the process of execution, we might have some resources open that will not get closed, so we can use finally block. finally block gets executed always, whether exception occurred or not.

**Explain Java Exception Hierarchy?**

When any exception is raised an **exception object** is getting created. Java Exceptions are hierarchical and inheritance is used to categorize different types of exceptions. Throwable is the parent class of Java Exceptions Hierarchy and it has two child objects – Error and Exception. Exceptions are further divided into checked exceptions and runtime exception.

**1. Errors**: Errors are exceptional scenarios that are out of scope of application and it’s not possible to anticipate and recover from them, for example hardware failure, JVM crash or out of memory error. That’s why we have a separate hierarchy of errors and we should not try to handle these situations. Some of the common Errors are OutOfMemoryError and StackOverflowError.

**2. Checked Exceptions**: Checked Exceptions are exceptional scenarios that we can anticipate in a program and try to recover from it, for example FileNotFoundException. We should catch this exception and provide useful message to user and log it properly for debugging purpose. Exception is the parent class of all Checked Exceptions and if we are throwing a checked exception, we must catch it in the same method or we have to propagate it to the caller using throws keyword.

**3. Runtime Exception**: Runtime Exceptions are cause by bad programming, for example trying to retrieve an element from the Array. We should check the length of array first before trying to retrieve the element otherwise it might throw ArrayIndexOutOfBoundException at runtime. RuntimeException is the parent class of all runtime exceptions. If we are throwing any runtime exception in a method, it’s not required to specify them in the method signature throws clause. Runtime exceptions can be avoided with better programming.



**What are important methods of Java Exception Class?**

Some of the useful methods of Throwable class are;

**1. public String getMessage()**– This method returns the message String of Throwable and the message can be provided while creating the exception through it’s constructor.

**2. public String getLocalizedMessage()**– This method is provided so that subclasses can override it to provide locale specific message to the calling program. Throwable class implementation of this method simply use getMessage() method to return the exception message.

**3. public synchronized ThrowablegetCause()**– This method returns the cause of the exception or null id the cause is unknown.

**4. public String toString()**– This method returns the information about Throwable in String format, the returned String contains the name of Throwable class and localized message.

**5. public void printStackTrace()**– This method prints the stack trace information to the standard error stream, this method is overloaded and we can pass PrintStream or PrintWriter as argument to write the stack trace information to the file or stream.

**What is difference between Checked and Unchecked Exception in Java?**

* Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

* Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions e.g.ArithmeticException,NullPointerException,ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

**What is difference between throw and throws keyword in Java?**

|  |  |
| --- | --- |
| Throw | throws |
| Java throw keyword is used to explicitly throw an exception. | Java throw keyword is used to explicitly throw an exception. |
| Checked exception cannot be propagated using throw only. | Checked exception can be propagated with throws. |
| Throw is followed by an instance. | Throws is followed by class. |
| Throw is used within the method. | Throws is used with the method signature. |
| You cannot throw multiple exceptions. | You can declare multiple exceptions e.g. public void method()throws IOException,SQLException. |

**How to write custom exception in Java?**

If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.

Let's see a simple example of java custom exception.

class InvalidAgeException extends Exception{

 InvalidAgeException(String s){

  super(s);

 }

}

class TestCustomException1{

   static void validate(int age)throws InvalidAgeException{

     if(age<18)

      throw new InvalidAgeException("not valid");

     else

      System.out.println("welcome to vote");

   }

   public static void main(String args[]){

      try{

      validate(13);

      }catch(Exception m){System.out.println("Exception occured: "+m);}

      System.out.println("rest of the code...");

  }

}

**What are different scenarios causing “Exception in thread main”?**

Some of the common main thread exception scenarios are:

· **Exception in thread main java.lang.UnsupportedClassVersionError**: This exception comes when your java class is compiled from another JDK version and you are trying to run it from another java version.

· **Exception in thread main java.lang.NoClassDefFoundError**: There are two variants of this exception. The first one is where you provide the class full name with .class extension. The second scenario is when Class is not found.

· **Exception in thread main java.lang.NoSuchMethodError: main**: This exception comes when you are trying to run a class that doesn’t have main method.

· **Exception in thread “main” java.lang.ArithmeticException**: Whenever any exception is thrown from main method, it prints the exception is console. The first part explains that exception is thrown from main method, second part prints the exception class name and then after a colon, it prints the exception message.

**What is difference between final, finally and finalize in Java?**

* final and finally are keywords in java whereas finalize is a method.
* final keyword can be used with class variables so that they can’t be reassigned, with class to avoid extending by classes and with methods to avoid overriding by subclasses,
* finally keyword is used with try-catch block to provide statements that will always gets executed even if some exception arises, usually finally is used to close resources.
* finalize() method is executed by Garbage Collector before the object is destroyed, it’s great way to make sure all the global resources are closed.

Out of the three, only finally is related to java exception handling.

**What happens when exception is thrown by main method?**

When exception is thrown by main() method, Java Runtime terminates the program and print the exception message and stack trace in system console.

**Annotations:**

**What are annotations? What are their typical use cases?**

Annotations are metadata bound to elements of the source code of a program and have no effect on the operation of the code they operate.

Their typical uses cases are:

· **Information for the compiler** – with annotations, the compiler can detect errors or suppress warnings

· **Compile-time and deployment-time processing** – software tools can process annotations and generate code, configuration files, etc.

· **Runtime processing** – annotations can be examined at runtime to customize the behavior of a program

**Describe some useful annotations from the standard library.**

There are several annotations in the *java.lang* and *java.lang.annotation* packages, the more common ones include but not limited to:

· *@Override –* marks that a method is meant to override an element declared in a superclass. If it fails to override the method correctly, the compiler will issue an error

· *@Deprecated*– indicates that element is deprecated and should not be used. The compiler will issue a warning if the program uses a method, class, or field marked with this annotation

· *@SuppressWarnings*– tells the compiler to suppress specific warnings. Most commonly used when interfacing with legacy code written before generics appeared

· *@FunctionalInterface* – introduced in Java 8, indicates that the type declaration is a functional interface and whose implementation can be provided using a Lambda Expression

**How can you create an annotation?**

Annotations are a form of an interface where the keyword *interface* is preceded by *@,* andwhose body contains *annotation type element* declarations that look very similar to methods:

|  |  |
| --- | --- |
|  | public@interfaceSimpleAnnotation {  String value();  int[] types();  } |

After the annotation is defined, yon can start using it in through your code:

|  |  |
| --- | --- |
|  | @SimpleAnnotation(value = "an element", types = 1)  public class Element {  @SimpleAnnotation(value = "an attribute", types = { 1, 2 })  public Element nextElement;  } |

Note that, when providing multiple values for array elements, you must enclose them in brackets.

Optionally, default values can be provided as long as they are constant expressions to the compiler:

|  |  |
| --- | --- |
|  | public @interface SimpleAnnotation {  String value() default "This is an element";  int[] types() default { 1, 2, 3 };  } |

Now, you can use the annotation without those elements:

|  |  |
| --- | --- |
|  | @SimpleAnnotation  public class Element {  // ...  } |

Or only some of them:

|  |  |
| --- | --- |
|  | @SimpleAnnotation(value = "an attribute")  public Element nextElement; |

**What object types can be returned from an annotation method declaration?**

The return type must be a primitive, *String*, *Class*, *Enum*, or an array of one of the previous types. Otherwise, the compiler will throw an error.

**Which program elements can be annotated?**

Annotations can be applied in several places throughout the source code. They can be applied to declarations of classes, constructors, and fields:

**What are meta-annotations?**

Are annotations that apply to other annotations.

All annotations that aren’t marked with *@Target,* or are marked with it but include *ANNOTATION\_TYPE* constant are also meta-annotations:

@Target(ElementType.ANNOTATION\_TYPE)

public @interface SimpleAnnotation {

    // ...

}

**What are repeating annotations?**

These are annotations that can be applied more than once to the same element declaration.

For compatibility reasons, since this feature was introduced in Java 8, repeating annotations are stored in a *container annotation* that is automatically generated by the Java compiler. For the compiler to do this, there are two steps to declared them.

First, we need to declare a repeatable annotation:

@Repeatable(Schedules.class)

public @interface Schedule {

    String time() default "morning";

}

Then, we define the containing annotation with a mandatory *value* element, and whose type must be an array of the repeatable annotation type:

public @interface Schedules {

    Schedule[] value();

}

Now, we can use @Schedule multiple times:

@Schedule

@Schedule(time = "afternoon")

@Schedule(time = "night")

voidscheduledMethod() {

}

**Regex**

**What is regex? why we go for regex?**

The Java Regex or Regular Expression is an API to define pattern for searching or manipulating strings.

It is widely used to define constraint on strings such as password and email validation. We will be able to test our own regular expressions by the Java Regex Tester Tool.

**What are the classes in Java that helps to deal with regular expressions?**

Java has a dedicated package named java.util.regex that has three classes which help to deal with regular expressions. Following is a brief description about them.

Pattern – represents compiled representation of a regex. You can get a new instance by using the static ‘compile’ method which accepts a regular expression as the first argument.

PatternSyntaxException – unchecked exception that occurs when there is a problem with the regular expression pattern’s syntax.

Matcher – engine that interprets the pattern and does match operations for an input string. You can get a new instance by using a Pattern object’s matcher method.

**What is a metacharacter?**

A metacharacter is a character that has a special meaning to a regular expression engine. This will not be counted as a regular character by the regex engine. Examples of metacharacters are ^, $, ., \*, +, etc.

**What are predefined character classes?**

Predefined character classes are useful shorthand notations available for commonly used regular expressions.

Predefined Character Class Description

. Any character

d 0-9

s Whitespace character

w A word character, ie [A-Za-z\_0-9]

D Non-digit character

W Non-word character

S Non-whitespace character

**Which is regex engine class?**

Matcher class is the engine of regex and it implements MatchResult(I), used perform matching operations on a character sequence.

**Which is compiler of regex?**

Pattern class the compiled version of regex, used to define a pattern for the regex engine.

Advantages of regex

With smart code completion, safe refactoring,and better support for Node.js, angular.

**ENUM**

**What is enum and why we go for enum?**

It is a data type that contains fixed set of constants.

It can be used for days of the week , directions etc. The java enum constants are static and final implicitly. It is available from JDK 1.5.

**Can Enum implement interface in Java?**

Yes, Enum can implement interface in Java. Since enum is a type, similar to class and interface, it can implement interface. This gives a lot of flexibility to use Enum as specialized implementation in some cases.

**Can Enum extends class in Java?**

No, Enum cannot extend class in Java. Since all Enum by default extend abstract base class java.lang.Enum, obviously they cannot extend another class, because Java doesn't support multiple inheritance for classes. Because of extending java.lang.Enum class, all enum gets methods like ordinal(), values() or valueOf().

**Can we declare Constructor inside Enum in Java?**

Yes, you can, but remember you can only declare either private or package-private constructor inside enum. public and protected constructors are not permitted inside enum.

**Can we override toString() method for Enum? What happens if we don't?**

Yes we can override toString in Enum, as like any other class it also extends java.lang.Object and has toString()method available, but even if you don't override, you will not going to regret much, because abstract base class of enum does that for you and return name, which is name of the enum instance itself. here is the code of toString() method from Enum class

**How do you create Enum without any instance?**

yes, we can create Enum without any instance in Java, say for creating a utility class.