**JAVA FUNDAMENTALS:**

1. **What is JVM and Is it platform independent?**

Java Virtual Machine (JVM) is the execution engine of java programming language. JVM is responsible for converting byte code into machine readable code. JVM is not platform independent, thats why you have different JVM for different operating systems. We can customize JVM with Java Options, such as allocating minimum and maximum memory to JVM. It’s called virtual because it provides an interface that doesn’t depend on the underlying OS.

1. **What is difference between JDK and JVM and JRE?**

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.

The JVM performs following main tasks:

Loads code, Verifies code, Executes code

Provides runtime environment

JRE is an acronym for Java Runtime Environment . It is used to provide runtime environment. It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.

JDK is an acronym for Java Development Kit.It physically exists.It contains JRE + development tools.

1. **Why Java is not pure Object Oriented language?**

Java is not a pure object oriented language because it supports Primitive datatype such as int, byte, long? etc, to be used, which are not objects.

1. **What are principle concepts of OOPS?**

Abstraction, Encapsulation, Inheritance, Polymorphism

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

A java package is a group of similar types of classes, interfaces and sub-packages.Package in java can be categorized in two form, built-in package and user-defined package.java.lang package is default impored package.

1. **What is Overloading and Overriding in Java?**

|  |  |
| --- | --- |
| **Overloading** | **Overriding** |
| Method overloading is used to increase readability of the program. | Method overriding is used to provide the specific implementation of the method that is already provided by its super class. |
| Method overloading is performed within class. | Method overriding occurs *in two classes* that have IS-A (inheritance) relationship. |
| In case of method overloading, *parameter must be different*. | In case of method overriding, *parameter must be same*. |
| Method overloading is the example of *compile time polymorphism*. | Method overriding is the example of *run time polymorphism*. |
| In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter. | *Return type must be same or covariant* in method overriding. |

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

A class would be called subclass if it extends some another class. This child class would be called subclass and that parent class would be called as super-class. On the other hand, nested classes are those which are defined inside a class.

1. **What is the difference between abstract class and interface?**

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| Abstract class can **have abstract and non-abstract**methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.. |
| Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| The **abstract keyword** is used to declare abstract class | The **interface keyword** is used to declare interface. |

1. **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.

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

1. **What is Data Encapsulation and what’s its significance?**

**Data encapsulation**, also known as **data** hiding, is the mechanism whereby the implementation details of a class are kept hidden from the user. The user can only perform a restricted set of operations on the hidden members of the class by executing special functions commonly called methods.

1. **What is Java Bean Class?**

JavaBeans are classes that encapsulate many [objects](https://en.wikipedia.org/wiki/Object_(computer_science)) into a single object (the bean). They are serializable, have a [zero-argument constructor](https://en.wikipedia.org/wiki/Nullary_constructor), and allow access to properties using [getter and setter methods](https://en.wikipedia.org/wiki/Mutator_method).

1. **What are Access modifiers?**

Access modifiers are keywords in object-oriented languages that set the accessibility of classes, methods, and other members. Access modifiers are a specific part of programming language syntax used to facilitate the encapsulation of components.

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

Reusability,facility to use public methods of base class without rewriting the same. 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.

1. **Why multiple Inheritance not supported in Java?**

Java doesn’t allow multiple inheritance to avoid the ambiguity caused by it. One of the example of such problem is the diamond problem that occurs in multiple inheritance.

1. **What is the diamond problem in inheritance?**

The "diamond problem" ("deadly diamond of death") is an ambiguity that arises when two classes B and C inherit from A, and class D inherits from both B and C. If there is a method in A that B and C have [overridden](https://en.wikipedia.org/wiki/Method_overriding_(programming)), and D does not override it, then which version of the method does D inherit: that of B, or that of C?, this is situation that is called as diamond problem.

1. **What is the difference between break and continue statement?**

An unlabeled break statement terminates the innermost switch, for, while, or do-while statement, but a labeled break terminates an outer statement.

The continue statement skips the current iteration of a for, while , or do-while loop. The unlabeled form skips to the end of the innermost loop's body and evaluates the boolean expression that controls the loop.

1. **What is nested class?**

**Java inner class** or nested class is a class which is declared inside the class or interface. We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable. Additionally, it can access all the members of outer class including private data members and methods.

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

Constructors use this to refer to another constructor in the same class with a different parameter list.

Constructors use super to invoke the super class's constructor. If a constructor uses super, it must use it in the first line; otherwise, the compiler will complain.

1. **What is Serialization and Deserialization**

**Serialization** is the process of converting an object into a stream of bytes in order to store the object or transmit it to memory, a database, or a file. Its main purpose is to save the state of an object in order to be able to recreate it when needed. The reverse process is called **deserialization**.

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

Java Heap space is used by java runtime to allocate memory to Objects and JRE classes. Whenever we create any object, it’s always created in the Heap space.

Garbage Collection runs on the heap memory to free the memory used by objects that doesn’t have any reference. Any object created in the heap space has global access and can be referenced from anywhere of the application.

1. **How garbage collection is done in Java?**

The garbage collector will look for objects which aren't being used anymore, and gets rid of them, freeing up the memory so other new objects can use that piece of memory.

1. **How destructors are defined in Java?**

A destructor is a special method called automatically during the destruction of an object. Actions executed in the destructor include the following:

* Recovering the heap space allocated during the lifetime of an object
* Closing file or database connections
* Releasing network resources
* Releasing resource locks

**ARRAY:**

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

Array is an indexed collection of homogeneous elements, they are fixed in size, to create an array we have to declare, instantiate and initialize the java array together by:

int a[]={33,3,4,5};//declaration, instantiation and initialization

1. **Advantages and disadvantages of Array?**

**Advantages:**

Code Optimization**:** It makes the code optimized, we can retrieve or sort the data easily.

Random access: We can get any data located at any index position.

**Disadvantages:**Size Limit: We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.

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

An array without name is known as anonymous [array in java](https://www.thecrazyprogrammer.com/2015/05/array-in-java-1-d.html).

* Anonymous array is passed as an argument of method.
* Anonymous array is created and initialized in the same line.

class AnonymousArray

{

static void print(int a[])

{

  for(int i=0;i<a.length;++i)

   System.out.print(a[i]+" ");

}

static void print(int a[][])

{

  for(int i=0;i<a.length;++i)

  {

   for(int j=0;j<a[i].length;++j)

    System.out.print(a[i][j]+" ");

   System.out.println("");

  }

}

public static void main(String...s)

{

  //1d anonymous array

  print(new int[]{10,20,30,40});

  System.out.println("n");

  //2d anonymous array

  print(new int[][]{{10,20},{30,40},{50,60}});

}

}

1. **What are “jagged” arrays in java?**

It is a new feature supported by Java. In Jagged arrays, each row, in a two-dimensional array, may contain different lengths. Let us design a two-dimensional array with 4 rows where the first row contains 4 elements, the second row with 1 element, the third row with 2 elements and the fourth row with 3 elements.

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

public class A {

public static void main(String args[]) {

int a[] = { 1, 2, 3, 4, 5, 6 };

int b[] = new int[a.length];

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

b[i] = a[i];

}

}

}

1. **What is difference between ArrayIndexOutfOBounds and ArrayStoreException?**

Public class ArrayIndexOutOfBoundsException extends IndexOutOfBoundsException

Thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

public class ArrayStoreException extends RuntimeException

Thrown to indicate that an attempt has been made to store the wrong type of object into an array of objects.

Object x[] = new String[3];  
x[0] = new Integer(0);

1. **Where does array stored in memory?**

It is stored in the heap. Array is an object, and objects are stored at heap.

**STRING:**

1. **What is String in Java?**

String is built in final class in java.lang package, object of String is used represent group of characters enclosed in “ ”. String object is immutable.

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

By literal – String s1= “Hello World”;

By new keyword- String s2= new String (“Hello World”);

1. **What is String subSequence method?**

Java 1.4 introduced CharSequence interface and String implements this interface, this is the only reason for the implementation of subSequence method in String class. Internally it invokes the [String substring](https://www.journaldev.com/807/java-string-substring) method.

String subSequence method returns a character sequence that is a subsequence of this sequence. An invocation of this method of the form str.subSequence(begin, end) behaves in exactly the same way as the invocation of str.substring(begin, end).

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

char chararray[]=string.toCharArray();//the arrray contains all the characters in the string .

String chartostring = Character.toString(chararrayname);

1. **How to convert String to byte array and vice versa?**

String input = "Hello World";

      byte[] bytes = input.getBytes(Charset.forName("UTF-8"));

System.out.println("Input : " + input);

      System.out.println("Input [Byte Format] : " + bytes);

String s = new String(bytes, Charset.forName("UTF-8")); // for UTF-8 encoding

System.out.println("String created from byte array in UTF-8 encoding: " + s);

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

String : It is an immutable class, which provides lot of utility methods for String object.

StringBuffer: it is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.

StringBuffer is less efficient than StringBuilder.

StringBuilder: is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.

StringBuilder is more efficient than StringBuffer.

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

String is immutable for several reasons, here is a summary:

* **Security**: parameters are typically represented as String in network connections, database connection urls, usernames/passwords etc. If it were mutable, these parameters could be easily changed.
* **Synchronization** and concurrency: making String immutable automatically makes them thread safe thereby solving the synchronization issues.
* **Caching**: when compiler optimizes your String objects, it sees that if two objects have same value (a="test", and b="test") and thus you need only one string object (for both a and b, these two will point to the same object).

**Class loading**: String is used as arguments for class loading. If mutable, it could result in wrong class being loaded (because mutable objects change their state).

1. **What is String Pool?**

String Pool in Java corresponds to an allocation of memory in Java heap memory. It consists of a collection of String objects, which are shared and reused among several String object references for same String content.

Note : This capability is gained through the [immutability](http://www.codedjava.com/2017/09/immutable-objects-in-java_19.html) nature of Java String.

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

According to [String#intern()](https://docs.oracle.com/javase/9/docs/api/java/lang/String.html" \l "intern--), intern method is supposed to return the String from the String pool if the String is found in String pool, otherwise a new string object will be added in String pool and the reference of this String is returned.

1. **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.

**COLLECTION:**

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

 A collection is an object that groups multiple elements into a single unit.

Collections framework provides unified architecture for manipulating and representing collections.

Benefits of Collections Framework:   
1.Improves program quality and speed  
2.Increases the chances of reusability of software  
3.Decreases programming effort

1. 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.

1. What are the basic interfaces of Java Collections Framework?

[Collection](http://crunchify.com/category/java-tutorials/) 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 [interface](http://crunchify.com/what-is-an-interface-in-java-how-its-used-java-tutorial-example-attached/) models the mathematical set [abstraction](http://crunchify.com/what-is-an-abstract-class-and-abstract-method-in-java-when-should-i-use-it/) 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 its 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

1. What are common algorithms implemented in Collections Framework?

common algorithms implemented in Collections Framework are :

Sorting

Shuffling

Routine Data Manipulation

Searching

Composition

Finding Extreme Values

1. Why Collection doesn’t extend Cloneable and Serializable interfaces?

Collection interface specifies group of Objects known as elements. How the elements are 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.  
The semantics and the implications of either cloning or serializing come into play when dealing with the actual implementation; so concrete implementation should decide how it should be cloned or serialized, or even if it can be cloned or serialized.  
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.

1. What is difference between Array and ArrayList? When will you use Array over ArrayList?

Arrays can contain primitive or Objects whereas ArrayList can contain only Objects.  
Arrays are fixed size whereas ArrayList size is dynamic.  
Arrays doesn’t provide a lot of features like ArrayList, such as addAll, removeAll, iterator etc.

Although ArrayList is the obvious choice when we work on list, there are few times when array are good to use.

* If the size of list is fixed and mostly used to store and traverse them.
* For list of primitive data types, although Collections use autoboxing to reduce the coding effort but still it makes them slow when working on fixed size primitive data types.
* If you are working on fixed multi-dimensional situation, using [][] is far more easier than List<List<>>

1. What are similarities and difference between ArrayList, Linked List and Vector?

ArrayList and Vector are similar classes in many ways.

1. Both are index based and backed up by an array internally.
2. Both maintains the order of insertion and we can get the elements in the order of insertion.
3. The iterator implementations of ArrayList and Vector both are fail-fast by design.
4. ArrayList and Vector both allows null values and random access to element using index number.

These are the differences between ArrayList and Vector.

1. Vector is synchronized whereas ArrayList is not synchronized. However if you are looking for modification of list while iterating, you should use CopyOnWriteArrayList.
2. ArrayList is faster than Vector because it doesn’t have any overhead because of synchronization.
3. ArrayList is more versatile because we can get synchronized list or read-only list from it easily using Collections utility class.

8. Why Map interface doesn’t extend Collection interface?

Although Map interface and it’s implementations are part of Collections Framework, Map are not collections and collections are not Map. Hence it doesn’t make sense for Map to extend Collection or vice versa.  
If Map extends Collection interface, then where are the elements? Map contains key-value pairs and it provides methods to retrieve list of Keys or values as Collection but it doesn’t fit into the “group of elements” paradigm.

9. What is difference between Enumeration and Iterator interface?

Enumeration is twice as fast as Iterator and uses very less memory. Enumeration is very basic and fits to basic needs. But Iterator is much safer as compared to Enumeration because it always denies other threads to modify the collection object which is being iterated by it.  
Iterator takes the place of Enumeration in the Java Collections Framework. Iterators allow the caller to remove elements from the underlying collection that is not possible with Enumeration. Iterator method names have been improved to make it’s functionality clear.

10.What is difference between Stack and Queue?

Both Queue and Stack are used to store data before processing them.java.util.Queue is an interface whose implementation classes are present in java concurrent package. Queue allows retrieval of element in First-In-First-Out (FIFO) order but it’s not always the case. There is also Deque interface that allows elements to be retrieved from both end of the queue.  
Stack is similar to queue except that it allows elements to be retrieved in Last-In-First-Out (LIFO) order.  
Stack is a class that extends Vector whereas Queue is an interface.

11.What is difference between Comparable and Comparator interface?

  Java provides Comparable interface which should be implemented by any custom class if we want to use Arrays or Collections sorting methods. Comparable interface has compareTo(T obj) method which is used by sorting methods. We should override this method in such a way that it returns a negative integer, zero, or a positive integer if “this” object is less than, equal to, or greater than the object passed as argument.

But, in most real life scenarios, we want sorting based on different parameters. For example, as a CEO, I would like to sort the employees based on Salary, an HR would like to sort them based on the age. This is the situation where we need to use Comparator interface because Comparable.compareTo(Object o) method implementation can sort based on one field only and we can’t chose the field on which we want to sort the Object.

Comparator interface compare(Object o1, Object o2) method need to be implemented that takes two Object argument, it should be implemented in such a way that it returns negative int if first argument is less than the second one and returns zero if they are equal and positive int if first argument is greater than second one.

Check this post for use of Comparable and Comparator interface to[sort objects](https://www.journaldev.com/780/comparable-and-comparator-in-java-example).

12.How HashMap works in Java?

HashMap stores key-value pair in Map.Entry static nested class implementation. HashMap works on hashing algorithm and uses hashCode() and equals() method in put and get methods.

When we call put method by passing key-value pair, HashMap uses Key hashCode() with hashing to find out the index to store the key-value pair. The Entry is stored in the LinkedList, so if there are already existing entry, it uses equals() method to check if the passed key already exists, if yes it overwrites the value else it creates a new entry and store this key-value Entry.

When we call get method by passing Key, again it uses the hashCode() to find the index in the array and then use equals() method to find the correct Entry and return it’s value.

13.What are different Collection views provided by Map interface?

Map interface provides three collection views:

1. **Set<K> keySet()**: Returns a Set view of the keys contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator’s own remove operation), the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the Iterator.remove, Set.remove, removeAll, retainAll, and clear operations. It does not support the add or addAll operations.
2. **Collection<V> values()**: Returns a Collection view of the values contained in this map. The collection is backed by the map, so changes to the map are reflected in the collection, and vice-versa. If the map is modified while an iteration over the collection is in progress (except through the iterator’s own remove operation), the results of the iteration are undefined. The collection supports element removal, which removes the corresponding mapping from the map, via the Iterator.remove, Collection.remove, removeAll, retainAll and clear operations. It does not support the add or addAll operations.
3. **Set<Map.Entry<K, V>> entrySet()**: Returns a Set view of the mappings contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa. If the map is modified while an iteration over the set is in progress (except through the iterator’s own remove operation, or through the setValue operation on a map entry returned by the iterator) the results of the iteration are undefined. The set supports element removal, which removes the corresponding mapping from the map, via the Iterator.remove, Set.remove, removeAll, retainAll and clear operations. It does not support the add or addAll operations.

**MULTI-THREADING :**

1. What is the difference between Process and Thread?

|  |  |
| --- | --- |
| PROCESS | THREAD |
| A process is a collection of one or more threads and associated system resources | Threads are light-weight processes within a process . |
| Process can be divided into multiple threads | Threads cannot be sub divided. |
| Each process has its own memory space | Threads of the same process share a common memory space |
| It is difficult to create a process | It is easy to create a thread. |

2.What are the benefits of multi-threaded programming?

1. **Responsiveness**. Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user.
2. **Resource sharing**. Processes may only share resources through techniques such as shared memory or message passing. Such techniques must be explicitly arranged by the programmer. However, threads share the memory and the resources of the process to which they belong by default.
3. **Economy.** Allocating memory and resources for process creation is costly. Because threads share the resources of the process to which they belong, it is more economical to create and context-switch threads. Empirically gauging the difference in overhead can be difficult, but in general, it is much more time consuming to create and manage processes than threads.
4. **Scalability**. The benefits of multithreading can be greatly increased in a multiprocessor architecture, where threads may be running in parallel on different processors. A single-threaded process can only run on one processor, regardless how many are available. Multithreading on a multiCPU machine increases parallelism.

3. What is difference between user Thread and daemon Thread?

When we create a Thread in java program, it’s known as user thread. A daemon thread runs in background and doesn’t prevent JVM from terminating. When there are no user threads running, JVM shutdown the program and quits. A child thread created from daemon thread is also a daemon thread.

4.How can we create a Thread in Java?

There are two ways to create a thread:

|  |  |
| --- | --- |
| BY extending Thread | By implementing runnable interface |
| **class** Test **extends** Thread  {  **public** **void** run(){  Sytem.out.println(" running...");   }  **public** **static** **void** main(String args[])  {  Test t1=**new** Test();  t1.start();    }  } | **class** Test2 **implements** Runnable  {   **public** **void** run(){  System.out.println(“running...");   }  **public** **static** **void** main(String args[])  {  Test2 m1=**new** Test2();  Thread t1 =**new** Thread(m1);  t1.start();    }  } |
| O/p: running… | O/p: running... |

5.What are different states in lifecycle of Thread?

During the life time of a thread, there are many states it can enter.

They include :

* New State-if you create an instance of Thread class but before the invocation of start() method.
* Runnable-After the invocation of start() method but the thread Scheduler is not selected to running
* Running- A Thread is running state if thread scheduler selected it
* Non-Runnable: This is the state when the thread is still alive but currently not eligible to run.
* Terminated- A terminated or dead state when its run() exits.

6.What is Thread Scheduler and Time Slicing?

**Thread scheduler** in java is the part of the JVM that decides which thread should run. Only one thread at a time can run in a single process.

The period of **time** for which a process is allowed to run in a pre-emptive multitasking system is generally called the **time slice** or quantum.

7.What is context-switching in multi-threading?

A **context switch** (also sometimes referred to as a process **switch** or a task **switch**) is the **switching** of the CPU (central processing unit) from one process or **thread** to another. A process (also sometimes referred to as a task) is an executing (i.e., running) instance of a program.

8.How does thread communicate with each other?

When threads share resources, communication between Threads is important to coordinate their efforts. Object class wait(), notify() and notifyAll() methods allows threads to communicate about the lock status of a resource.

9.Why thread communication methods wait (), notify() and notifyAll() are in Object class?

In Java every Object has a monitor and wait, notify methods are used to wait for the Object monitor or to notify other threads that Object monitor is free now. There is no monitor on threads in java and synchronization can be used with any Object, that’s why it’s part of Object class so that every class in java has these essential methods for inter thread communication.

10.Why wait(), notify() and notifyAll() methods have to be called from synchronized method or block?

When a Thread calls wait() on any Object, it must have the monitor on the Object that it will leave and goes in wait state until any other thread call notify() on this Object. Similarly when a thread calls notify() on any Object, it leaves the monitor on the Object and other waiting threads can get the monitor on the Object. Since all these methods require Thread to have the Object monitor, that can be achieved only by synchronization, they need to be called from synchronized method or block.

11.Why Thread sleep () and yield () methods are static?

Thread sleep() and yield() methods work on the currently executing thread. So there is no point in invoking these methods on some other threads that are in wait state. That’s why these methods are made static so that when this method is called statically, it works on the current executing thread and avoid confusion to the programmers who might think that they can invoke these methods on some non-running threads.

12.How can we achieve thread safety in Java?

There are several ways to achieve thread safety in java – synchronization, atomic concurrent classes, implementing concurrent Lock interface, using volatile keyword, using immutable classes and Thread safe classes.

13.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.

Avoid Nested Locks, Lock Only What is Required and Avoid waiting indefinitely are common ways to avoid deadlock situation.

14.What is Java Timer Class? How to schedule a task to run after specific interval?

java.util.Timer is a utility class that can be used to schedule a thread to be executed at certain time in future. Java Timer class can be used to schedule a task to be run one-time or to be run at regular intervals.

java.util.TimerTask is a abstract class that implements Runnable interface and we need to extend this class to create our own Timer Task that can be scheduled using java Timer class.

15. What is Thread Pool? How can we create Thread Pool in Java?

A thread pool manages the pool of worker threads, it contains a queue that keeps tasks waiting to get executed.

A thread pool manages the collection of Runnable threads and worker threads execute Runnable from the queue.

java.util.concurrent.Executors provides implementation of java.util.concurrent.Executor interface to create the thread pool in java.

**EXCEPTION HANDLING:**

1. What is Exception & Exception Handling in Java?

Exception is an error event that can happen during the execution of a program and disrupts it’s normal flow. Exception can arise from different kind of situations such as wrong data entered by user, hardware failure, network connection failure etc.

Whenever any error occurs while executing a java statement, an exception object is created and then JRE tries to find exception handler to handle the exception. If suitable exception handler is found then the exception object is passed to the handler code to process the exception, known as **catching the exception**. If no handler is found then application throws the exception to runtime environment and JRE terminates the program.

**Java Exception handling**framework is used to handle runtime errors only, compile time errors are not handled by exception handling framework.

1. What are the Exception Handling Keywords in Java?

There are four keywords used in java exception handling.

1. **throw**: Sometimes we explicitly want to create exception object and then throw it to halt the normal processing of the program.**throw** keyword is used to throw exception to the runtime to handle it.
2. **throws**: When we are throwing any checked 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 usingthrows 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.
5. Explain Java Exception Hierarchy?

Java Exceptions are hierarchical and [inheritance](https://www.journaldev.com/644/inheritance-java-example) is used to categorize different types of exceptions. Throwable is the parent class of Java Exceptions Hierarchy and it has two child objects – Error andException. Exceptions are further divided into checked exceptions and runtime exception.

**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.

**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.

**Runtime Exceptions** are caused 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.

4.What are important methods of Java Exception Class?

Exception and all of it’s subclasses doesn’t provide any specific methods and all of the methods are defined in the base class Throwable.

1. **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. **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. **synchronized Throwable getCause()** – This method returns the cause of the exception or null id the cause is unknown.
4. **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. **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.

5. What is difference between Checked and Unchecked Exception in Java?

1. Checked Exceptions should be handled in the code using try-catch block or else main() method should use throws keyword to let JRE know about these exception that might be thrown from the program. Unchecked Exceptions are not required to be handled in the program or to mention them in throws clause.
2. Exception is the super class of all checked exceptions whereasRuntimeException is the super class of all unchecked exceptions.
3. Checked exceptions are error scenarios that are not caused by program, for example FileNotFoundException in reading a file that is not present, whereas Unchecked exceptions are mostly caused by poor programming, for example NullPointerException when invoking a method on an object reference without making sure that it’s not null.

6. What is difference between throw and throws keyword in Java?

throws keyword is used with method signature to declare the exceptions that the method might throw whereas throw keyword is used to disrupt the flow of program and handing over the exception object to runtime to handle it.

7.How to write custom exception in Java?

We can extend Exception class or any of it’s subclasses to create our custom exception class. The custom exception class can have it’s own variables and methods that we can use to pass error codes or other exception related information to the exception handler.

8.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.

9.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.

10.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.

**REGULAR EXPRESSIONS:**

1. 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.

1. 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.

1. What is a metacharacter?

A metacharacter is a character that has a special meaning to a regular expression engine.

1. 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

1. 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.

1. Which is compiler of regex?

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

1. What are advantages of regex?

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

**ANNOTATIONS:**

1. 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.

Usage of annotations:

* Information for the compiler: Annotations can be used by the compiler to produce warnings or even errors based on differentrules.
* Documentation: Annotations can be used by software applications to measure the quality of the code like FindBugs or PMD do or generate reports automatically like Jenkins, Jira or Teamcity.
* Code generation: annotations can be used to generate code or XML ﬁles automatically using metadata information present in the code.
* Runtime processing: Annotations that are examined in runtime can be used for different objectives like unit testing (Junit),
* dependency injection (Spring), validation, logging (Log4J) ,data access (Hibernate) etc.

2. 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.

3.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 @interface SimpleAnnotation {      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; |

4.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.

Here’s an example code that successfully follows this principle:

|  |  |
| --- | --- |
|  | enum Complexity {      LOW, HIGH  }    public @interface ComplexAnnotation {      Class<? extends Object> value();        int[] types();        Complexity complexity();  } |

5.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:

* Methods and their parameters
* Local variables, including a loop and resource variables
* And even packages, through the *package-info.java* file
* Type casts
* Implements
* clause
* And *throws* clause

6.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.

7.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.