1. **What is Java?** Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible
2. **What does it mean to be a “pass-by-value” language? How is this different from passing by reference?** Pass by value is often safer than pass by reference, because you cannot accidentally modify the parameters to your method/function. This makes the language simpler to use, since you don't have to worry about the variables you give to a function. You know they won't be changed while in pass-by-reference they will be changed, and this is often what you *expect*.
3. **What is the difference between the JRE, JDK, and JVM?** JDK is for development purpose whereas JRE is for running the java programs. JDK and JRE both contains JVM so that we can run our java program. JVM is the heart of java programming language and provides platform independence.
4. **What happens during the compilation process?** Java source code is compiled into bytecode when we use the javac compiler. The bytecode gets saved on the disk with the file extension .class . When the program is to be run, the bytecode is converted, using the just-in-time (JIT) compiler. The result is machine code which is then fed to the memory and is executed.
5. **What are the benefits of using Java?**

* Simple: Java was designed to be easy to use, write, compile, debug, and learn than other programming languages.
* Object-Oriented: Allows you to create modular programs and reusable code.
* Platform-Independent: Ability to move easily from one computer system to another.
* Distributed: Designed to make distributed computing easy with the networking capability that is inherently integrated into it.
* Secure: The Java language, compiler, interpreter, and runtime environment were each developed with security in mind.
* Allocation: Java has the feature of Stack allocation system. It helps the data to be stored and can be restored easily.
* Multithreaded: The capability for a program to perform several tasks simultaneously within a program.

1. **What is a constructor?** Constructor is a block of code that initializes the newly created object.
2. **What is the first line of a constructor? Myclass() {}**
3. **What are the primitive data types?** Primitive types are the most basic data types available within the Java language. There are 8: boolean , byte , char , short , int , long , float and double
4. **What is a no args constructor?**A constructor that does not accept any arguments.
5. **What is the default constructor?**
6. **Under what circumstances does a class have a default constructor? When there is no constructor defined**
7. **What are the scopes of a variable in java?** The scope of a variable defines the section of the code in which the variable is visible, **variables** that are defined within a block are not accessible outside that block.
8. **What are the different access modifiers? What does each one do?**

**Package == namespace (group classes with related purpose or functionality)**

**Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

**Default:** The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

**Protected:** The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

**Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

1. **What are the different control statements, and how are they different?**

**If/Else statements**

**\* While loops**

**\* Do-While loops**

**\* For loops**

**\* Enhance For loops**

**\* Switch statements**

1. **How do you create an Array in java?** int intArray[];
2. **What is varargs?** A method that takes variable number of arguments is called a variable-arity method, or simply a varargs method.
3. **What is the first line of a java application? Import / public class main {}**
4. **What are packages and imports?** A package is simply a container that groups related types (Java classes, interfaces, enumerations and annotations)
5. **What is a static import?** allows members (fields and methods) which have been scoped within their container class as public static , to be used in Java code without specifying the class in which the field has been defined.
6. **What does the static modifier do?**

The static modifier creates a class variable. That is to say, this variable does not belong to any instance of the class. It exists from the moment the program starts running, and you do not need to create an instance of the class to access it.

1. **What are Strings?**

String class in Java is a final class (cannot be extended)

String in Java is immutable, they cannot be changed once created

Strings are stored in a special pool on the heap called “string pool”

There are 2 ways to create strings in Java

1) String s = “s” // GOOD

2) String s = new String(“s”) 🡪 bad

1. **What are some string methods?**
2. **What is the difference between String, StringBuilder, and StringBuffer?**

We can use mutable versions of Strings by using the StringBuilder or StringBuffer classes. Do NOT use StringBuffer unless working in a multithreaded environment.

1. **What is the string pool?** Strings are stored in a special pool in the heap
2. **What is the difference between the stack and the heap? What is stored in each**
3. **What goes on the stack?** method calls and local variable are stored on the stack
4. **What goes on the heap?** Strings, var references, objects are stored on the heap
5. **What is an exception?** An exception disrupts the normal flow of an application and can be reasonably recovered from
6. **What is the difference between an exception and an error?** An error disrupts the normal flow of an application and cannot be reasonably recovered from
7. **What are the different ways in which we can handle an exception? Try-catch block and throws throw**
8. **What are the differences between a checked exception and an unchecked exception?**

Checked exceptions are exceptions that extend exception but not runtime exception

A checked exception must be handled by the developer by: using a try-catch block or throwing the exception to the caller (Compile time)

Ex:

- fileNotFoundException

- SQLException

Unchecked exception are exceptions that extend RuntimeException. These do NOT have to be handled by the developer (Run time)

Ex:

- ArithmeticException

- NullPointerException

- HibernateException

- NumberFormatException

- IllegalArgumentException

Errors can also be “handled” by using a try-catch block, but you should NEVER do this

1. **How many catch blocks can be used in a try catch? (min and max)?**

**A try catch can have multiple catch blocks**

1. **What does finally do?**

**The finally block is always executed**

1. **Can finally be skipped? How? Yes**
2. **How do I create a custom exception?**

**Extend the Exception class**

1. **What is autoboxing?**

Sometimes, Java will automatically wrap a primitive value as a wrapper class object. This is called "autoboxing"

1. **What is auto-unboxing?**

Java will also unwrap a Wrapper Class object and convert into back into a primitive for us. This is called auto-unboxing.

1. **What is a wrapper class?**

A Wrapper class is a class whose object wraps or contains a primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store a primitive data types. **object representation of primitives**

1. **What are the different wrapper classes? Char, Byte, Int, Long, Boolean, Float, Double**
2. **What is garbage collection?**

Java garbage collection is the process by which Java programs perform automatic memory management.

1. **How do I perform garbage collection?** Using *System.gc()* method
2. **What is the difference between final, finally, and finalize?**

Final prevents overriding prevents extending of a class, prevents us from changing where a reference point

Finally allows us to create a block of code that is always executed (barring the cases “where” it doesn’t)

Finalize() => a method that is invoked before an object is garbage collected, can only be invoked once per object.

1. **What is the Reflection API?**

**Reflection** is an **API** which is used to examine or modify the behavior of methods, classes, interfaces at runtime.

1. **What is a Lambda expression?**

Lambda expression allows us to create a reference to a function. We can then pass this reference to another function.

We are essentially passing an implementation of the interface's abstract method to another function as an argument.

1. **What is a functional interface? Contain only 1 abstract method.**

Interfaces define methods that an implementing class must implement

- any class that implements an interface must implement all of the unimplemented methods on that interfaces.

- Interfaces do NOT implement other interfaces.

- Interfaces can extend other interfaces.

- cannot be instantiated

Interfaces:

Methods on interfaces cannot be final or private

Methods on interfaces can have implementations if we use the “default” or “ ” keywords

A class can implement as many interfaces as you want it to

A class can only extend 1 class

An interface can extend as many interfaces as the developer wants it to

An interface does not implement others interfaces

All methods in interface are *public, abstract*

All variables in interface are *public, static, final*

1. **What are the pillars of object oriented programming? Explain them.**

**Inheritance**

**Polymorphism**

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

Abstract class are designed to be extended (Not implemented)

Can have abstract or concrete methods

Cannot be instantiated

Any concrete class that extends an abstract class must implement the unimplemented methods contained in that abstract class.

Abstract classes can extend other abstract classes

Abstract classes do NOT have to have any abstract methods.

Abstract method: a method with no implementation

An **abstract class** allows **you to** create functionality that subclasses can implement or override. An **interface** only allows **you to** define functionality, not implement it.

1. **How many interfaces can a class implement in Java?** As many as you want
2. **How many interfaces can an interface implement in Java?** Can’t
3. **How many interfaces can interface extend in Java?** As many as you want
4. **Can an abstract class implement an interface in Java?** Yes
5. **How many classes can a class extend in Java?** Only 1
6. **How many abstract classes can a class extend in Java?** 1
7. **What are the different non-access modifiers? What do they do?**
8. **What are the differences between FileinputStream, FileReader, and BufferedReader (as well as their counterparts)?**

A FileOutputStream allows us to write to a file in bytes. It is a stream.

A FileWriter allows us to write characters to a file.

A BufferedWriter allows us to write sequences of characters(i.e. strings) to a

file. Note that BufferedWriter is synchronized (so thread-safe), so it is

slower than FileWriter.

A FileInputStream is a stream that reads bytes from a file

A FileReader reads characters from a file

A BufferedReader reads sequences of characters (strings) from a file. Like BufferedWriter, it is synchronized, so it is thread safe.

1. **What is the Scanner class?**

The Scanner class is often used to take user input. Note, however, that the

Scanner class's use isn't just reserved for taking user input; at its core,

the Scanner class is efficient at parsing strings and other values.

1. **What are some Scanner methods?**
2. **What is serialization and deserialization?**

Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory.

1. **How do I serialize an object?**

**Implement Serializable interface**

1. **What is a marker interface? What is an example of a marker interface?**

It is an empty interface (no field or methods). Examples of marker interface are Serializable, Clonnable and Remote interface. All these interfaces are empty interfaces.

1. **How do I prevent some data from getting serialized?** Define data member as transient
2. **What is the difference between Collection and Collections?**

Collections is a group of methods which we can use in conjunction with classes that inherit from the collection interface.

Collection is an interface

Collection interface does NOT support primitive

1. **What are some of the subinterfaces of the Collection interface? List, Set, Queue**
2. **What are the various input/delete/get methods for List, Set, and Queue?**
3. **How are the List, Set, and Queue interfaces different from each other?**

List:

- supports random access

- supports duplicates

- maintains order

- unlike an array, a list has a dynamic size

- capacity is increased by 50% of current capacity as needed

ArrayList

Traversal is Fast

Adding new object is slow

LinkedList

Traveral is slower

Adding new objects is faster

Vector is a thread safe, implementation of an ArrayList

Stack follows a first in, last out policy.

Set:

- support random access

- do not support duplicates

- does not maintain the order of objects that have been added to it

- If you want a sorted set, use an implementation of the SortedSet Interface

1. **What is an iterator?**

'Iterator' is an interface which belongs to collection framework. It allows us to traverse the collection, access the data element and remove the data elements of the collection.

1. **How is an iterator different from a ListIterator?**

ListIterator can traverse in reverse direction

1. **What are generics? Why use them?**

Generics enable types (classes and interfaces) to be parameters when defining classes, interfaces and methods. type parameters provide a way for you to re-use the same code with different inputs

1. **Do generics support primitive data types?** No. Anything use Generics has to be converted in Objects
2. **What is the difference between comparator and comparable?**

- If sorting of objects needs to be based on natural order then use Comparable whereas if you sorting needs to be done on attributes of different objects, then use Comparator in Java.

- For using Comparable, Class needs to implement it whereas for using Comparator we don't need to make any change in the class.

Single sorting sequence vs multiple sorting sequence.

1. **What is the purpose of the Object class?**

The Object class defines the basic state and behavior that all objects must have, such as the ability to compare oneself to another object, to convert to a string, to wait on a condition variable, to notify other objects that a condition variable has changed, and to return the object's class.

1. **What are some methods that are inherited from the Object class? Equals(), getclass(), toString()**

**53)What is the difference between == and .equals? same**

1. **What is the purpose of hashcode?**

The purpose of the hashCode() method is to provide a numeric representation of an object's contents so as to provide an alternate mechanism to loosely identify it. By default the hashCode() returns an integer that represents the internal memory address of the object.

1. **What is the Map interface’s relationship to the Collection interface?**

Map is NOT a part of Collection interface

map <k, v>

- uses key-value pairs

- inorder to access anything in a map, you must have the key

- NOT iterable

1. **Explain the difference between Hashmap and Hashtable.**

HashMap and Hashtable store key/value pairs in a hash table. When using a Hashtable or HashMap, we specify an object that is used as a key, and the value that you want linked to that key. The key is then hashed, and the resulting hash code is used as the index at which the value is stored within the table.

HashMap is non synchronized. It is not-thread safe and can’t be shared between many threads

Hashtable is synchronized. It is thread-safe and can be shared with many threads.

1. **How do I iterate through a HashMap?**
2. **What is multithreading? Why do we use it?**

**Multithreading entails multiple threads running concurrently.**

A thread is a single use of execution

Thread have priorities, a thread’s priority can be high or low.

A low priority thread is referred to as a daemon thread (i.e. garbage collection)

A daemon thread does not prevent the JVM from exiting.

Each thread gets its own stack; the heap is shared across all threads.

1. **What are the different ways to create a thread? What is different about them?**

In Java, there are only 2 ways to create a thread. You can:

1) extend the thread class

2) implement the Runnable interface. This is useful when you want a class to be able to extend another class

1. **Why would I choose one method of creating a thread over another?**
2. **What are the JVM recognized states of a thread?**

There are 6 JVM recognized states of a thread:

NEW: a thread has been created, it’s NOT running

RUNNABLE: signifies that a thread is running/ being executed

WAITING: a thread has been passed for some indefinite amount of time

TIMED\_WAITING: a thread has been passed /is waiting for a definite amount of time

BLOCKED: a thread cannot continue its line of execution (e.g. because another thread won’t release a lock/ resource that is needed for this thread to continue running)

TERMINATED: a thread has finished its line of execution

1. **What are various methods that threads have?**
2. **What is synchronization?**

Synchronization is the process of allowing threads to execute one after another.

Synchronization in java is the capability to control the access of multiple threads to any shared resource.

1. **What are the risks of synchronization?**

it increases waiting time of thread and effects performance of the system

1. **What is deadlock, livelock, and thread starvation?**

Thread starvation occurs when a “greedy” thread won’t release a resource that is needed by another thread, causing that thread to “starve”.

Deadlock vs. livelock

- Deadlock occurs when 2 threads must exchange resources but will only do so if the other thread hands over its resource 1st

- Livelock occurs when 2 threads must exchange resources but will only do so if the other thread accepts its resource 1st.

1. **What is the producer and consumer problem?**

**Make sure producer won’t try to add data to the buffer when buffer is full and consumer won’t try to remove data from the buffer when buffer is empty**

1. **What is JUnit?**

JUnit is a unit testing framework for Java programming language

1. **What are the annotations of JUnit?**

@BeforeClass – Run once before any of the test methods in the class, public static void.

@AfterClass – Run once after all the tests in the class have been run, public static void.

@Before – Run before @Test, public void.

@After – Run after @Test, public void.

@Test – This is the test method to run, public void

1. **What are the different assert methods of JUnit?**

Assert is a method useful in determining Pass or Fail status of a test case

* [Boolean](https://www.guru99.com/junit-assert.html#2)
* [Null object](https://www.guru99.com/junit-assert.html#3)
* [Identical](https://www.guru99.com/junit-assert.html#4)
* [Assert Equals](https://www.guru99.com/junit-assert.html#5)
* [Assert Array Equals](https://www.guru99.com/junit-assert.html#6)
* [Fail Message](https://www.guru99.com/junit-assert.html#7)

1. **How do I create a test case and test suite in JUnit?**

**To create test case: add a method annotated with @test.**

**To create test suite: annotate the class with @RunWith and @SuiteClass**

1. **What is Maven?**

Maven is a build automation tool

Maven defines a process for distributing our software/projects

Maven has a default lifecycle which carries out the process of building and deploying our projects for us

1. **What is the Maven lifecycle? Validate compile test package verify install deploy**
2. **What is the purpose of the pom.xml?**

contains information about the project and configuration details used by Maven to build the project

1. **What is the purpose of using Maven?**

1) Maven validates that our project is “correct”

- it checks for the existence of the pom.xml (POM = Project Object Model – xml config file) (mvn validate)

2) Maven compiles our source code for us (mvn compile)

3) Maven tests our source code (runs our tests) (mvn test)

4) Maven packages our project (e.g. jar, war) (mvn package)

5) Maven verifies by running checks on the results of our integration tests (mvn verify)

6) Maven installs our project to our local Maven repository (mvn install)

7) Maven deploys our projects to the Maven repository (mvn deploy)

Mvn clean package/target

8) Maven downloads dependencies for us (From the remote Maven repository) and caches from locally for us.

* Making the build process easy
* Providing a uniform build system
* Providing quality project information
* Providing guidelines for best practices development
* Allowing transparent migration to new features

1. **What is the difference between a Singleton and Factory?**

\* The Singleton design pattern is a design pattern in which only one instance

\* of a class exists for the duration of the application. In order to achieve

\* this design pattern, we create a static reference to an object of the class

\* and make the constructor private so that no one can create another

\* instance of this class. We then return this static instance each time an

\* instance of this class is requested. Thus, the same exact instance is used

\* each time an instance is requested.

A **Factory Pattern** or **Factory** Method **Pattern** says that just define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate. In other words, subclasses are responsible to create the instance of the class.

A **singleton** pattern ensures that you always get back the same instance **of** whatever type you are retrieving, whereas the **factory** pattern generally gives you a **different** instance **of** each type. The purpose **of** the **singleton** is where you want all calls to go through the same instance. The purpose of the factory is to create and return new instances.

Factory design pattern allow us to create objects without having to specify the exact (concrete) object

1. **What is the difference between a Java Bean and a POJO?**

**POJO plain old java object, doesn’t follow strict convention**

**- contains fields, constructors, getters and setters**

**JavaBean follows a more strict pattern than POJO.**

**- implement serializable interface**

**- contains getters setters, no args contructor**

**- overide equals() and hashcode() method**

**- overide the toString()**

1. **What is the difference between implicit and explicit casting?**
2. **When can you upcast a variable? Downcast a variable?**

Covariance entails setting a reference to the super to the instance of the child class.

Upcasting is casting to a supertype, while downcasting is casting to a subtype. Upcasting is always allowed, but downcasting involves a type check and can throw a ClassCastException.

1. **What is Git?**
2. **What are some of the benefits of using Git?**
3. **What is version control?**
4. **What are some basic Git commands?**
5. **What is the difference between git init and git clone?**