1.What is Java?

* [Java](https://www.javatpoint.com/java-tutorial) is the high-level, [object-oriented](https://www.javatpoint.com/java-oops-concepts), secure programming language
* It is platform-independent
* Has high performance
* Is Multithreaded and portable programming language.

**Explain JDK, JRE and JVM?**

|  |  |  |
| --- | --- | --- |
| **JDK vs JRE vs JVM** | | |
| **JDK** | **JRE** | **JVM** |
| It stands for Java Development Kit. | It stands for Java Runtime Environment. | It stands for Java Virtual Machine. |
| It is the tool necessary to compile, document and package Java programs. | JRE refers to a runtime environment in which Java bytecode can be executed. | It is an abstract machine. It is a specification that provides a run-time environment in which Java bytecode can be executed. |
| It contains JRE + development tools. | It’s an implementation of the JVM which physically exists. | JVM follows three notations: Specification, **Implementation,**and **Runtime Instance**. |

2. What do you understand by Java virtual machine?

* [Java Virtual Machine](https://www.javatpoint.com/jvm-java-virtual-machine) is a virtual machine that enables the computer to run the Java program.
* JVM acts like a run-time engine which calls the main method present in the Java code
* The Java code is compiled by JVM to be a Bytecode which is machine independent and close to the native code
  + - .Java ------🡪 .class

3. What is JIT compiler?

* **Just-In-Time(JIT) compiler:** It is used to improve the performance.
* JIT compiles parts of the bytecode that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.

4. What gives Java its 'write once and run anywhere' nature?

The bytecode. Java compiler converts the Java programs into the class file (Byte Code) which is the intermediate language between source code and machine code. This bytecode is not platform specific and can be executed on any computer.

5. What is classloader?

Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

1. **Bootstrap ClassLoader**: This is the first classloader which is the superclass of Extension classloader. It loads the *rt.jar* file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes, etc.
2. **Extension ClassLoader**: This is the child classloader of Bootstrap and parent classloader of System classloader. It loads the jar files located inside *$JAVA\_HOME/jre/lib/ext* directory.
3. **System/Application ClassLoader**: This is the child classloader of Extension classloader. It loads the class files from the classpath. By default, the classpath is set to the current directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

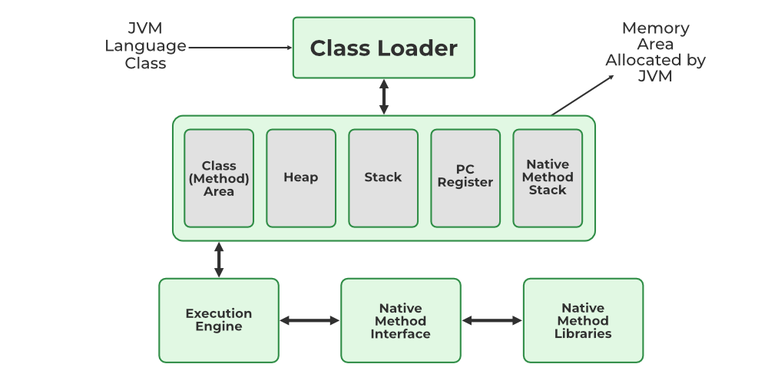
**6.** What are constructors in Java?

In Java, constructor refers to a block of code which is used to initialize an object. It must have the same name as that of the class. Also, it has no return type and it is automatically called when an object is created.

There are two types of constructors:

1. Default Constructor: In Java, a default constructor is the one which does not take any inputs. In other words, default constructors are the no argument constructors which will be created by default in case you no other constructor is defined by the user. Its main purpose is to initialize the instance variables with the default values. Also, it is majorly used for object creation.
2. Parameterized Constructor: The parameterized constructor in Java, is the constructor which is capable of initializing the instance variables with the provided values. In other words, the constructors which take the arguments are called parameterized constructors.

7. **What are Memory storages available with JVM?**



JVM consists of a few memory storages as mentioned below:

1. Class(Method) Area: stores class-level data of every class such as the runtime constant pool, field, and method data, and the code for methods.
2. Heap: Objects are created or objects are stored. It is used to allocate memory to objects during run time.
3. Stack: stores data and partial results which will be needed while returning value for method and performing dynamic linking
4. Program Counter Register: stores the address of the Java virtual machine instruction currently being executed.
5. Native Method Stack: stores all the native methods used in the application.

8. **What is singleton class in Java and how can we make a class singleton?**

Singleton class is a class whose only one instance can be created at any given time, in one JVM. A class can be made singleton by making its constructor private.

9. What is the difference between compile-time polymorphism and runtime polymorphism?

|  |  |  |
| --- | --- | --- |
| **SN** | **compile-time polymorphism** | **Runtime polymorphism** |
| 1 | In compile-time polymorphism, call to a method is resolved at compile-time. | In runtime polymorphism, call to an overridden method is resolved at runtime. |
| 2 | It is also known as static binding, early binding, or overloading. | It is also known as dynamic binding, late binding, overriding, or dynamic method dispatch. |
| 3 | Overloading is a way to achieve compile-time polymorphism in which, we can define multiple methods or constructors with different signatures. | Overriding is a way to achieve runtime polymorphism in which, we can redefine some particular method or variable in the derived class. By using overriding, we can give some specific implementation to the base class properties in the derived class. |
| 4 | It provides fast execution because the type of an object is determined at compile-time. | It provides slower execution as compare to compile-time because the type of an object is determined at run-time. |
| 5 | Compile-time polymorphism provides less flexibility because all the things are resolved at compile-time. | Run-time polymorphism provides more flexibility because all the things are resolved at runtime. |

10. **Explain different data types in Java.**

There are 2 types of data types in Java as mentioned below:

1. Primitive Data Type – 8
2. Non-Primitive Data Type or Object Data type

**Primitive Data Type : Eg: Boolean, byte, char, short, int,long, float, double**

**Non-Primitive Data Type:** Reference Data types will contain a memory address of the variable’s values because it is not able to directly store the values in the memory. Types of Non-Primitive are mentioned below:

* Strings
* Array
* Class
* Object
* Interface

11. **Differentiate between instance and local variables.**

| **Instance Variable** | **Local Variable** |
| --- | --- |
| Declared outside the method, directly invoked by the method. | Declared within the method. |
| Has a default value. | No default value |
| It can be used throughout the class. | The scope is limited to the method. |

12. What is the meaning of immutable regarding String?

The simple meaning of immutable is unmodifiable or unchangeable. In Java, String is immutable, i.e., once string object has been created, its value can't be changed.

13. Why are the objects immutable in java?

Because Java uses the concept of the string literal. Suppose there are five reference variables, all refer to one object "sachin". If one reference variable changes the value of the object, it will be affected by all the reference variables. That is why string objects are immutable in java.

  
14. How many ways can we create the string object?

1) String Literal

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

String s="welcome";

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. String objects are stored in a special memory area known as the **string constant pool**

String s1="Welcome";

String s2="Welcome";//It doesn't create a new instance

2) By new keyword

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

In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the constant string pool. The variable s will refer to the object in a heap (non-pool).

12.1  **== Operator**: Checks if two references point to the same object in memory.

 **equals() Method**: Checks if two objects are logically equal based on their content (often overridden in classes like String, Integer, and custom classes).

15. Why java uses the concept of the string literal?

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

16. What are the differences between String and StringBuffer?

|  |  |  |
| --- | --- | --- |
| **No.** | **String** | **StringBuffer** |
| 1) | The String class is immutable. | The StringBuffer class is mutable. |
| 2) | The String is slow and consumes more memory when you concat too many strings because every time it creates a new instance. | The StringBuffer is fast and consumes less memory when you cancat strings. |
| 3) | The String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method. | The StringBuffer class doesn't override the equals() method of Object class. |

17. What are the differences between StringBuffer and StringBuilder?

The differences between the StringBuffer and StringBuilder is given below.

|  |  |  |
| --- | --- | --- |
| **No.** | **StringBuffer** | **StringBuilder** |
| 1) | 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. |
| 2) | StringBuffer is *less efficient* than StringBuilder. | StringBuilder is *more efficient* than StringBuffer. |

18. How can we create an immutable class in Java?

We can create an immutable class by defining a final class having all of its members as final. Here, It have one final datamember, a parameterized constructor and getter method.

**public** **final** **class** Employee{

**final** String pancardNumber;

**public** Employee(String pancardNumber){

**this**.pancardNumber=pancardNumber;

}

**public** String getPancardNumber(){

**return** pancardNumber;

}

  }

* Final class
* Final Data members
* Paramerterized Constructor
* Only getters, no setters

19.  What is the difference between final, finally and finalize?

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **final** | **finally** | **finalize** |
| 1) | Final is used to apply restrictions on class, method, and variable. The final class can't be inherited, final method can't be overridden, and final variable value can't be changed. | Finally is used to place important code, it will be executed whether an exception is handled or not. | Finalize is used to perform clean up processing just before an object is garbage collected. |
| 2) | Final is a keyword. | Finally is a block. | Finalize is a method. |

20. Collections Framework:

**Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.



21 . **Define a Java Class.**

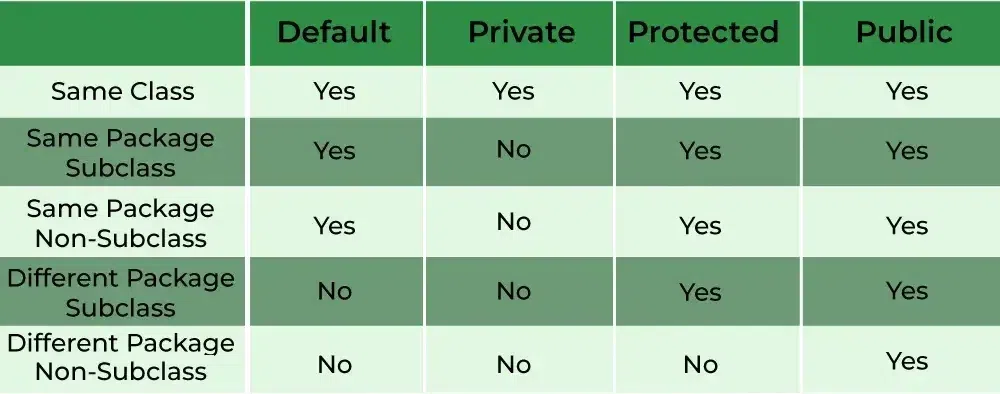
A class in Java is a blueprint which includes all your data.  A class contains fields (variables) and methods to describe the behavior of an object.

22. **What is an object in Java and how is it created?**

An object is a real-world entity that has a state and behavior. An object has three characteristics: State , Behavior , Identity. An object is created using the ‘new’ keyword.

ClassName obj = new ClassName();

23. Access Modifiers



24. ifference between ArrayList and LinkedList?

|  |  |  |
| --- | --- | --- |
| **No.** | **ArrayList** | **LinkedList** |
| 1) | ArrayList uses a dynamic array. | LinkedList uses a doubly linked list. |
| 2) | ArrayList is not efficient for manipulation because too much is required. | LinkedList is efficient for manipulation. |
| 3) | ArrayList is better to store and fetch data. | LinkedList is better to manipulate data. |
| 4) | ArrayList provides random access. | LinkedList does not provide random access. |
| 5) | ArrayList takes less memory overhead as it stores only object | LinkedList takes more memory overhead, as it stores the object as well as the address of that object. |

25. **Main concepts of OOPs in Java?**

Object-Oriented Programming or OOPs is a programming style that is associated with concepts like:

1. *Inheritance:*Inheritance is a process where one class acquires the properties of another.
2. *Encapsulation:*Encapsulation in Java is a mechanism of wrapping up the data and code together as a single unit.
3. *Abstraction:*Abstraction is the methodology of hiding the implementation details from the user and only providing the functionality to the users.
4. *Polymorphism:*Polymorphism is the ability of a variable, function or object to take multiple forms.

26. Iterator and ListIterator

|  |  |  |
| --- | --- | --- |
| **No.** | **Iterator** | **ListIterator** |
| 1) | The Iterator traverses the elements in the forward direction only. | ListIterator traverses the elements in backward and forward directions both. |
| 2) | The Iterator can be used in List, Set, and Queue. | ListIterator can be used in List only. |
| 3) | The Iterator can only perform remove operation while traversing the collection. | ListIterator can perform ?add,? ?remove,? and ?set? operation while traversing the collection. |

27. What is the difference between Iterator and Enumeration?

|  |  |  |
| --- | --- | --- |
| **No.** | **Iterator** | **Enumeration** |
| 1) | The Iterator can traverse legacy and non-legacy elements. | Enumeration can traverse only legacy elements. |
| 2) | The Iterator is fail-fast. | Enumeration is not fail-fast. |
| 3) | The Iterator is slower than Enumeration. | Enumeration is faster than Iterator. |
| 4) | The Iterator can perform remove operation while traversing the collection. | The Enumeration can perform only traverse operation on the collection. |

28. What is the difference between HashMap and Hashtable?

|  |  |  |
| --- | --- | --- |
| **No.** | **HashMap** | **Hashtable** |
| 1) | HashMap is not synchronized. | Hashtable is synchronized. |
| 2) | HashMap can contain one null key and multiple null values. | Hashtable cannot contain any null key or null value. |
| 3) | HashMap is not ?thread-safe,? so it is useful for non-threaded applications. | Hashtable is thread-safe, and it can be shared between various threads. |
| 4) | 4) HashMap inherits the AbstractMap class | Hashtable inherits the Dictionary class. |

29. **Difference between List and Set:**

| **List** | **Set** |
| --- | --- |
| 1. The List is an indexed sequence. | 1. The Set is an non-indexed sequence. |
| 2. List allows duplicate elements | 2. Set doesn’t allow duplicate elements. |
| 3. Elements by their position can be accessed. | 3. Position access to elements is not allowed. |
| 4. Multiple null elements can be stored. | 4. Null element can store only once. |
| 5. List implementations are ArrayList, LinkedList, Vector, Stack | 5. Set implementations are HashSet, LinkedHashSet. |

30. **What is the difference between List, set, and map in java?**

| **List** | **Set** | **Map** |
| --- | --- | --- |
| The list interface allows duplicate elements | The set does not allow duplicate elements. | The map does not allow duplicate elements |
| The list maintains insertion order. | The set does not maintain any insertion order. | The map also does not maintain any insertion order. |
| We can add any number of null values. | But in the set almost only one null value. | The map allows a single null key at most and any number of null values. |
| The list implementation classes are Array List and LinkedList. | Set implementation classes are HashSet, LinkedHashSet, and TreeSet. | Map implementation classes are HashMap, HashTable, TreeMap, ConcurrentHashMap, and LinkedHashMap. |

31. Difference between ArrayList and Vector

| **ArrayList** | **Vector** |
| --- | --- |
| ArrayList is not Synchronized | The vector is synchronized. |
| The size of ArrayList is incremented up to 50% of the current array size if the number of elements exceeds its capacity. | The size of ArrayList is incremented up to 100% of the current array size if the number of elements exceeds its capacity. |
| ArrayList is fast because it is non-Synchronized. | Vector is slower because it’s synchronized. |
| The iterator interface is used to traverse the elements | An iterator interface or Enumeration can be used to traverse the vector. |

32. **Differentiate between HashSet and HashMap**

| **HashSet** | **HashMap** |
| --- | --- |
| HashSet implements the Set interface | HashMap implements the Map interface |
| No Duplicates are allowed | Yes duplicates values are allowed but no duplicate key is allowed |
| Dummy values are allowed in HashSet. | No Dummy values are allowed in HashMap. |
| A single Object is required during an add operation | 2 Objects are required during an add operation |
| Speed is comparatively slower than HashMap | Speed is comparatively faster than HashSet because of hashing technique has been used here. |
| Have a single null value | Single null key and any number of null values |
| Add() method is used for the insertion | The put () method is used for insertion. |

33. this() And super()

A screenshot of a computer

Description automatically generated

34. Comparable vs Comparator

| **Comparable** | **Comparator** |
| --- | --- |
| The Comparable interface provides a single sorting sequence. | The Comparator interface provides multiple sorting sequences. |
| The actual class is modified by a comparable interface | The actual class is not modified by the Comparator interface. |
| compareTo() method is used to sort elements. | compare() method is used to sort elements. |
| Comparable is present in the package java.lang | Comparator is present in the package java.util |

35. FailFast vs FailSafe

| **Fail-Fast** | **Fail-Safe** |
| --- | --- |
| ConcurrentModificationException is thrown while modifying the object during the iteration process. | No Exception is thrown |
| Fail-Fast needs less memory during the process. | Fail-Safe iterator requires more memory during the process. |
| A clone Object is not created during the iteration process. | A clone Object or a Copy is created during the iteration process. |
| Fail-Fast does not allow modification during the process of iteration. | Fail-Safe allows modification during the process of iteration. |
| Fail-Fast is fast, | Fail-Safe is slightly slower than fail fast. |
| **Examples:**  ArrayList, Vector, HashMap, HashSet, etc. | **Examples:**  ConcurrentHashMap, CopyOnWriteArrayList, etc. |

36. **How does Hashmap internally Works?**

HashMap works on the principle of Hashing. It uses hashcode() and equals() method

**The internal workings of HashMap:**

* Hashing - Hashing is a process of converting an object into integer form by using the method hashCode(). hashCode() method of the object class returns the memory reference of an object in integer form.
* equals() - This method is used to check whether 2 objects are equal or not. HashMap uses equals() to compare the key to whether they are equal or not. If the equals() method return true, they are equal otherwise not equal.
* Buckets - A bucket is an element of the HashMap array. It is used to store nodes. Two or more nodes can have the same bucket. In that case, a link list structure is used to connect the nodes.



37. How HashMap works – In depth :

HashMap is a widely used data structure in Java that implements the Map interface. It stores key-value pairs, allowing fast retrieval of values based on their keys. Internally, HashMap uses a hash table, which is an array where each slot (or "bucket") can hold one or more entries.

**Basic Structure of HashMap**

* **Array of Buckets**: The core of a HashMap is an array where each element is referred to as a "bucket."
* **Entry (Node) Class**: Each bucket holds a linked list of nodes (or entries). Each node stores a key-value pair, along with a reference to the next node in the list (in case of collisions).

A diagram of a computer

Description automatically generated

**How HashMap Works**

**a. Storing Elements**

When you put a key-value pair into a HashMap, the following steps occur:

**1.Compute Hash**: The hashCode() method of the key object is called to compute an integer hash code. This hash code is then transformed using a process called "hashing" to find the index of the bucket where the entry should be stored.

int hash = key.hashCode();

 2. **Locate Bucket**: The computed index determines the bucket where the key-value pair should be stored.

 3. **Collision Check**: If the bucket is empty, the entry is placed there. If it's not empty, the HashMap checks if the key already exists:

* If the key exists, the old value is replaced.
* If the key doesn't exist, a new node is added to the bucket's linked list.

**B. Retrieving Elements**

When you retrieve a value using a key, the following steps occur:

1. **Compute Hash**: The hash code of the key is computed, and the bucket index is determined using the same hashing process as during insertion.
2. **Locate Bucket**: The computed index points to the bucket where the value should be found.
3. **Search for Key**: The HashMap iterates through the nodes in the bucket (if there are multiple due to collisions) and compares the key using the equals() method.
   * If the key is found, the corresponding value is returned.
   * If not, null is returned.

C. **Collision Resolution**

Collisions occur when two keys hash to the same bucket index. HashMap uses two main techniques to handle collisions:

1. **Separate Chaining** (Linked List):
   * Initially, a bucket is either empty or contains a single entry.
   * When a collision occurs, the bucket starts a linked list, where each node points to the next key-value pair that hashes to the same bucket.

**d. Rehashing and Resizing**

When the number of entries in the HashMap exceeds the product of the load factor and the current capacity, the HashMap resizes itself to accommodate more entries:

1. **New Capacity**: The array size is doubled.
2. **Rehashing**: All existing entries are rehashed to determine their new positions in the resized array.

This process ensures that the HashMap maintains efficient performance even as it grows.

**e. Load Factor**

The load factor is a measure of how full the HashMap can get before it needs resizing. The default load factor is 0.75, meaning the HashMap will resize when it is 75% full.

38. What is Hash Collision

* Two or more keys can generate same hash values sometimes. This is called a collision.
* A hash collision occurs when two different inputs (keys) produce the same hash value or hash code.

**Why Hash Collisions Occur:**

* **Finite Hash Space**: Hash functions map input data of arbitrary size to fixed-size output (hash values). Since the output space is limited, different inputs might map to the same hash value.
* **Poor Hash Function Design**: Some hash functions distribute values unevenly, increasing the likelihood of collisions.

**Strategies to Handle and Fix Hash Collisions:**

**1. Separate Chaining (Open Hashing)**

* **Concept**: Store collided items in a linked list (or another structure) at each index of the hash table.
* **Implementation**: When a collision occurs, the new item is added to the list of items at the same hash index.

**2. Open Addressing (Closed Hashing)**

* **Concept**: Instead of storing multiple items at one index, probe (search) for another available slot in the array when a collision occurs.
* **Techniques**:
  + **Linear Probing**: Check the next slot (index + 1) until an empty slot is found.
  + **Quadratic Probing**: Check slots with increasing intervals (index + 1², + 2², etc.).
  + **Double Hashing**: Use a second hash function to determine the probe sequence.

**3.Dynamic Resizing and Rehashing**

* **Concept**: When the load factor (number of items divided by table size) exceeds a certain threshold, resize the hash table and rehash all existing keys to new indices.

39. What is the abstraction?

Abstraction is a process of hiding the implementation details and showing only functionality to the user. It displays just the essential things to the user and hides the internal information, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery. Abstraction enables you to focus on what the object does instead of how it does it.

40. What is the difference between abstraction and encapsulation?

Abstraction hides the implementation details whereas encapsulation wraps code and data into a single unit.

41. What is the abstract class?

A class that is declared as abstract is known as an abstract class. It needs to be extended and its method implemented. It cannot be instantiated. It can have abstract methods, non-abstract methods, constructors, and static methods. It can also have the final methods which will force the subclass not to change the body of the method.

42. What is a marker interface?

A Marker interface can be defined as the interface which has no data member and member functions. For example, Serializable, Cloneable are marker interfaces. The marker interface can be declared as follows.

**public** **interface** Serializable{

}

43. What are the differences between abstract class and interface?

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| An abstract class can have a method body (non-abstract methods). | The interface has only abstract methods. |
| An abstract class can have instance variables. | An interface cannot have instance variables. |
| An abstract class can have the constructor. | The interface cannot have the constructor. |
| An abstract class can have static methods. | The interface cannot have static methods. |
| You can extend one abstract class. | You can implement multiple interfaces. |
| The abstract class **can provide the implementation of the interface**. | The Interface **can't provide the implementation of the abstract class**. |
| The **abstract keyword** is used to declare an abstract class. | The **interface keyword** is used to declare an interface. |
| An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| An **abstract class** can be extended using keyword **extends** | An **interface class** can be implemented using keyword **implements** |
| A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| **Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |

44.  How to make a read-only class in Java?

A class can be made read-only by making all of the fields private. The read-only class will have only getter methods which return the private property of the class to the main method. We cannot modify this property because there is no setter method available in the class. Consider the following example.

**public** **class** Student{

**private** String college="AKG";

**public** String getCollege(){

**return** college;

}

}

45. A How many types of exception can occur in a Java program?

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception.

* **Checked Exception:** Checked exceptions are the one which are checked at compile-time. For example, SQLException, ClassNotFoundException, etc.
* **Unchecked Exception:** Unchecked exceptions are the one which are handled at runtime because they can not be checked at compile-time. For example, ArithmaticException, NullPointerException, ArrayIndexOutOfBoundsException, etc.
* **Error:** Error cause the program to exit since they are not recoverable. For Example, OutOfMemoryError, AssertionError, etc.

46. What is Exception Handling?

Exception Handling is a mechanism that is used to handle runtime errors. It is used primarily to handle checked exceptions. Exception handling maintains the normal flow of the program. There are mainly two types of exceptions: checked and unchecked. Here, the error is considered as the unchecked exception.

47. What is the difference between Checked Exception and Unchecked Exception?

1) 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.

2) Unchecked Exception

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

**48.** hierarchy of Java Exception classes



49. How do you create a custom exception in Java?

Java custom exceptions are used to customize the exception according to user need. Using the custom exception, we can have your own exception and message.  To create custom exception, we need to extend Exception class.

**public** **class** WrongFileNameException **extends** Exception {

**public** WrongFileNameException(String errorMessage) {

**super**(errorMessage);

    }

}

50. What is the difference between throw and throws?

|  |  |
| --- | --- |
| **throw keyword** | **throws keyword** |
| 1) The **throw** keyword is used to throw an exception explicitly. | The **throws** keyword is used to declare an exception. |
| 2) The checked exceptions cannot be propagated with throw only. | The checked exception can be propagated with throws |
| 3) The **throw** keyword is followed by an instance. | The **throws** keyword is followed by class. |
| 4) The **throw** keyword is used within the method. | The **throws** keyword is used with the method signature. |
| 5) You cannot throw multiple exceptions. | You can declare multiple exceptions, e.g., public void method()throws IOException, SQLException. |

51. What is Garbage Collection?

Garbage collection is a process of reclaiming the unused runtime objects. It is performed for memory management. In other words, we can say that It is the process of removing unused objects from the memory to free up space and make this space available for Java Virtual Machine

52. HashMap vs TreeMap

A screenshot of a computer

Description automatically generated

53. Advantages of Encapsulation in Java?

There are the following advantages of Encapsulation in Java?

* By providing only the setter or getter method, you can make the class read-only or write-only. In other words, you can skip the getter or setter methods.
* It provides you the control over the data. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.
* It is a way to achieve data hiding in Java because other class will not be able to access the data through the private data members.
* The encapsulate class is easy to test. So, it is better for unit testing.
* The standard IDE's are providing the facility to generate the getters and setters. So, it is easy and fast to create an encapsulated class in Java.

54. Diff b/w String, StringBuilder and String Buffer

A blue and white table with white text

Description automatically generated

55. Array vs ArrayList

A screenshot of a computer

Description automatically generated

56. **What is Polymorphism?**

Polymorphism is briefly described as “one interface, many implementations”. Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts – specifically, to allow an entity such as a variable, a function, or an object to have more than one form. There are two types of polymorphism:A group of different devices

Description automatically generated with medium confidence

1. Compile time polymorphism
2. Run time polymorphism

Compile time polymorphism is method overloading whereas Runtime time polymorphism is done using inheritance and interface.

57. **What is inheritance in Java?**

Inheritance in Java is the concept where the properties of one class can be inherited by the other. It helps to reuse the code and establish a relationship between different classes. Inheritance is performed between two types of classes:

1. Parent class (Super or Base class)
2. Child class (Subclass or Derived class)

A class which inherits the properties is known as Child Class whereas a class whose properties are inherited is known as Parent class.

**58. What is an association?**

Association is a relationship where all object have their own lifecycle and there is no owner. Let’s take the example of Teacher and Student. Multiple students can associate with a single teacher and a single student can associate with multiple teachers but there is no ownership between the objects and both have their own lifecycle. These relationships can be one to one, one to many, many to one and many to many.

**59. What do you mean by aggregation?**

An aggregation is a specialized form of Association where all object has their own lifecycle but there is ownership and child object can not belong to another parent object. Let’s take an example of Department and teacher. A single teacher can not belong to multiple departments, but if we delete the department teacher object will not destroy.

**60. What is composition in Java?**

Composition is again a specialized form of Aggregation and we can call this as a “death” relationship. It is a strong type of Aggregation. Child object does not have their lifecycle and if parent object deletes all child object will also be deleted. Let’s take again an example of a relationship between House and rooms. House can contain multiple rooms there is no independent life of room and any room can not belongs to two different houses if we delete the house room will automatically delete.

61.difference between aggregation and composition?

Aggregation represents the weak relationship whereas composition represents the strong relationship. For example, the bike has an indicator (aggregation), but the bike has an engine (composition).

62. What is multithreading?

Multithreading is a process of executing multiple threads simultaneously. Multithreading is used to obtain the multitasking. It consumes less memory and gives the fast and efficient performance.

* Threads share the same address space.
* The thread is lightweight.

63. What do you understand by inter-thread communication?

* The process of communication between synchronized threads is termed as inter-thread communication.
* Inter-thread communication is used to avoid thread polling in Java.
* The thread is paused running in its critical section, and another thread is allowed to enter (or lock) in the same critical section to be executed.
* It can be obtained by wait(), notify(), and notifyAll() methods.

64.  lifecycle of a Thread?

A thread can have one of the following states during its lifetime:

1. **New:** In this state, a Thread class object is created using a new operator, but the thread is not alive. Thread doesn't start until we call the start() method.
2. **Runnable:** In this state, the thread is ready to run after calling the start() method. However, the thread is not yet selected by the thread scheduler.
3. **Running:** In this state, the thread scheduler picks the thread from the ready state, and the thread is running.
4. **Waiting/Blocked:** In this state, a thread is not running but still alive, or it is waiting for the other thread to finish.
5. **Dead/Terminated:** A thread is in terminated or dead state when the run() method exits.



65. Differentiate between the Thread class and Runnable interface for creating a Thread?

The Thread can be created by using two ways.

* By extending the Thread class
* By implementing the Runnable interface

However, the primary differences between both the ways are given below:

* By extending the Thread class, we cannot extend any other class, as Java does not allow multiple inheritances while implementing the Runnable interface; we can also extend other base class(if required).
* By extending the Thread class, each of thread creates the unique object and associates with it while implementing the Runnable interface; multiple threads share the same object
* Thread class provides various inbuilt methods such as getPriority(), isAlive and many more while the Runnable interface provides a single method, i.e., run().

66. What does join() method?

The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

67. What is the difference between wait() and sleep() method?

|  |  |
| --- | --- |
| **wait()** | **sleep()** |
| 1) The wait() method is defined in Object class. | The sleep() method is defined in Thread class. |
| 2) The wait() method releases the lock. | The sleep() method doesn't release the lock. |

68. What is the synchronization?

Synchronization is the capability to control the access of multiple threads to any shared resource. It is used:

* To prevent thread interference.
* To prevent consistency problem.

When the multiple threads try to do the same task, there is a possibility of an erroneous result, hence to remove this issue, Java uses the process of synchronization which allows only one thread to be executed at a time. Synchronization can be achieved in three ways:

* by the synchronized method
* by synchronized block
* by static synchronization

69. What is the deadlock?

Deadlock is a situation in which every thread is waiting for a resource which is held by some other waiting thread. In this situation, Neither of the thread executes nor it gets the chance to be executed. Instead, there exists a universal waiting state among all the threads. Deadlock is a very complicated situation which can break our code at runtime.

**Ways to avoid the deadlock condition in Java:**

* **Avoid Nested lock:** Nested lock is the common reason for deadlock as deadlock occurs when we provide locks to various threads so we should give one lock to only one thread at some particular time.
* **Avoid unnecessary locks:** we must avoid the locks which are not required.
* **Using thread join:** Thread join helps to wait for a thread until another thread doesn't finish its execution so we can avoid deadlock by maximum use of join method.

70. What is Thread Scheduler in java?

In Java, when we create the threads, they are supervised with the help of a Thread Scheduler, which is the part of JVM. Thread scheduler is only responsible for deciding which thread should be executed. Thread scheduler uses two mechanisms for scheduling the threads: Preemptive and Time Slicing.

Java thread scheduler also works for deciding the following for a thread:

* It selects the priority of the thread.
* It determines the waiting time for a thread
* It checks the Nature of thread

71. Java 8 Features:

* Lambda Expressions:
  + Lambda expressions allow developers to write more concise and expressive code
  + by enabling the use of functional programming constructs.
  + They simplify the syntax for defining anonymous functions, making it easier to work with collections and perform operations like filtering, mapping, and reducing.
* Stream API:
  + The Stream API provides a new way to work with collections in Java. It allows developers to perform bulk operations on collections, such as filtering, mapping, sorting, and aggregating, using functional-style operations.
  + Streams enable more declarative and concise code, improving readability and reducing boilerplate code.
* Default and Static Methods in Interfaces:
  + Java 8 introduced the ability to define default and static methods in interfaces.
  + Default methods provide a way to add new methods to interfaces without breaking existing implementations
  + while static methods allow interfaces to have utility methods that can be called without an instance.
* Optional:
  + Optional is a new class introduced in Java 8 to represent optional values, reducing the need for null checks and improving code clarity and safety.
  + Optional provides methods to handle the presence or absence of a value in a more functional and idiomatic way.
  + This class helps in avoiding null pointer exceptions by providing methods to check the presence of a value before accessing it.
* Method References:
  + Method references allows to refer to methods or constructors using a concise syntax, improving code readability and reducing boilerplate code.
  + They complement the use of lambda expressions and provide an alternative way to achieve the same functionality.

72. Describe the more commonly found functional interfaces in the standard library.

Although many functional interfaces exist, these are the one's users most likely encounter:

* Function. Takes one argument and returns a result
* Consumer. Takes one argument and returns no result
* Supplier. Takes a not argument and returns a result
* Predicate. Takes one argument and returns a boolean

73. **What are the main components of a Stream?**

Components of the stream are:

* A data source
* Set of Intermediate Operations to process the data source
* Single Terminal Operation that produces the result

A diagram of a process flow

Description automatically generated

74. Most commonly used Intermediate operations

* map(Funtion<T, R>) ,
* distinct()  ,
* Filter(Predicate<T>)
* flatMap(mapper)

75. **Most common type of Terminal operations?**

* collect() - Collects single result from all elements of the stream sequence.
* reduce() - Produces a single result from all elements of the stream sequence
  + count() - Returns the number of elements on the stream.
  + min() - Returns the min element from the stream.
  + max() - Returns the max element from the stream.

76. FindFirst vs FindAny

| **findFirst()** | **findAny()** |
| --- | --- |
| Returns the first element in the Stream | Return any element from the Stream |
| Deterministic in nature | Non-deterministic in nature |

77. Collections vs Streams

| **Collections** | **Streams** |
| --- | --- |
| Data structure holds all the data elements | No data is stored. Have the capacity to process an infinite number of elements on demand |
| External Iteration | Internal Iteration |
| Can be processed any number of times | Traversed only once |
| Elements are easy to access | No direct way of accessing specific elements |
| Is a data store | Is an API to process the data |

78. **Given a list of integers, find out all the even numbers that exist in the list using Stream functions?**

List<Integer> list = Arrays.asList(10,15,8,49,25,98,32);  
 list.stream()  
 .filter(n -> n%2 == 0)  
 .forEach(System.out::println);

79. **How to use map to convert object into Uppercase in Java 8?**

List<String> nameLst = names.stream()  
 .map(String::toUpperCase)  
 .collect(Collectors.toList());

80. Anagram program in Java **“Mother In Law”** and **“Hitler Woman”**

81. How to reverse a string in Java

82. Count number of words in a given sentence – “Sister” s – 2,i-1,t-1,e-1,r-1

83. How to find second largest number in an integer array?

84. How to create a pyramid of numbers in Java

85. Find the Largest and Smallest no in an array

86. Find Missing Number in the array

87. Inheritance Program

88.Method Overloading and Method Overiding

89. Abstract class

90. Singleton class

91. Find duplicate elements in a list / remove duplicates in a list

92. sorting Using comparator and comparable