1) What are checked and unchecked exception ?
Checked Exception:
- Checked exceptions are exceptions that must be declared in the method's signature using the `throws` keyword or handled using try-catch blocks.
- Examples: `IOException`, `SQLException`, `FileNotFoundException`.

Unchecked Exception:
- Unchecked exceptions (also known as runtime exceptions) do not need to be declared or handled explicitly in the method's signature.
- Examples: `NullPointerException`, `ArithmeticException`, `ArrayIndexOutOfBoundsException`.

checked exceptions are required to be handled or declared, while unchecked exceptions do not require such handling and often indicate programming errors.

2) How do you handle exceptions in Java?
Java uses try-catch blocks to handle exceptions. The code that might raise an exception is placed inside the try block, and corresponding exception handling code is placed inside the catch block.

3) What is an ArithmeticException in Java?
ArithmeticException is thrown when an arithmetic operation is attempted with inappropriate operands, such as dividing by zero.

4) What is a NumberFormatException in Java?
NumberFormatException is thrown when a string cannot be parsed into a numerical format, such as trying to convert a non-numeric string to an integer.

5)What is an ArrayIndexOutOfBoundsException in Java?
ArrayIndexOutOfBoundsException is thrown when an attempt is made to access an element in an array with an index that is outside the valid range of the array.

6)How can you create a custom exception in Java?
To create a custom exception, you need to extend the Exception class (or its subclasses) and implement constructors based on your requirements.

7)What is multithreading in Java?
Multithreading is a mechanism in Java that allows the execution of multiple threads concurrently within a single process.

8)How do you implement multithreading using the Thread class?
To implement multithreading using the Thread class, you need to extend the Thread class and override the run() method with the code you want to run in the new thread. Then, you can create instances of your custom thread class and call their start() method to start the execution.

9) What is the difference between extending Thread class and implementing Runnable interface for multithreading?
When you extend the Thread class, you cannot extend any other class as Java doesn't support multiple inheritance. However, by implementing the Runnable interface, you can still extend another class to gain additional functionality.

10) How do you implement multithreading using the Runnable interface?
To implement multithreading using the Runnable interface, you need to create a class that implements the Runnable interface and override the run() method. Then, you can create instances of the class and pass them to Thread objects to start the execution.

11) How can you create separate threads for even and odd numbers using the Thread class?
To create separate threads for even and odd numbers, you can extend the Thread class and override the run() method to handle the printing of even and odd numbers accordingly.

12)Methods in thread class?

- start(): Starts the execution of the thread by calling its run() method.
- run(): Contains the code that defines the thread's task.
- sleep(long millis): Suspends the thread's execution for the specified duration.
- join(): Waits for the thread to finish its execution before moving on.
- yield(): Suggests to the thread scheduler that the current thread is willing to give up its CPU time.
- interrupt(): Interrupts the thread's execution, causing an InterruptedException if the thread is in a waiting state.
- isAlive(): Checks if the thread is still running.
- setName(String name): Sets the name of the thread.
- setPriority(int priority): Sets the priority of the thread.

13)What is thread priority?

- Thread priority is an integer value that determines the preference given to a thread by the thread scheduler.
- Priorities range from 1 (lowest) to 10 (highest) in Java.
- You can set the priority using the setPriority(int priority) method.

14) What is thread interrupt?

- interrupt() method is used to interrupt a thread's execution.
- It sets the interrupt status of the thread to true, causing blocking methods to throw an InterruptedException.
- The interrupted thread can decide how to respond to the interrupt, such as terminating gracefully or continuing execution.

15) Advantages of multithreading?

Improved Responsiveness: Multithreading allows applications to remain responsive and interactive even when performing time-consuming tasks.
Enhanced Performance: Multithreading can lead to better performance, especially on multi-core processors, by utilizing parallel processing.
Efficient Resource Utilization: Threads share the same memory space, reducing memory overhead and efficiently utilizing system resources.
Concurrency: Multithreading enables concurrent execution, allowing multiple tasks to proceed simultaneously.

Real-time Processing: Threads can handle time-critical operations and respond quickly to external events in real-time systems.

16)Thread life cycle

- New: Thread is created, but the start() method has not been called yet.
- Runnable: After calling start(), the thread is eligible for execution, waiting for CPU time.
- Running: When the CPU scheduler selects the thread for execution, it enters the running state.
- Blocked/Waiting: Thread enters this state when it is temporarily unable to run, waiting for a resource or condition.
- Terminated/Dead: Thread enters this state when its run() method completes or is explicitly terminated using stop() or interrupt().

17)What is dead lock?

- Deadlock occurs when two or more threads are blocked, each waiting for a resource held by another thread.
- The threads are stuck in a loop, unable to proceed, resulting in a deadlock situation.

18)What is deamon process?

A daemon process is a background process that runs in the background without direct interaction with the user or the foreground processes of the operating system

19) What is synchronization?

Synchronization in Java prevents multiple threads from interfering with shared resources to avoid data inconsistencies and race conditions.

20)What is the issue with non-synchronized withdrawal operation from a shared bank account in a multithreaded environment?

In a non-synchronized withdrawal operation, multiple threads can access and modify the account balance simultaneously, leading to data inconsistency and incorrect results. This situation is known as a race condition.

21) How do you ensure thread safety while performing the withdrawal operation in a shared bank account?
To ensure thread safety, you can use the synchronized keyword in the method that performs the withdrawal operation. This allows only one thread to access the method at a time, preventing race conditions and ensuring data consistency.

22) Apart from using the synchronized keyword for the entire method, how else can you achieve synchronization in Java?
You can use synchronized blocks, which allow you to specify a specific block of code to be executed by only one thread at a time. This gives more fine-grained control over synchronization.

23) What is inter thread communication?
It allows threads to communicate and synchronize their actions, often using wait(), notify(), and notifyAll() methods.

24) wait(), notify(), and notifyAll():
These methods are used for inter-thread communication. wait() makes a thread wait, while notify() wakes up one waiting thread, and notifyAll() wakes up all waiting threads.

25) What is the Producer-Consumer problem, and how is it related to inter thread communication?
The Producer-Consumer problem involves two types of threads, producers, and consumers. Producers produce data and add it to a shared buffer, while consumers consume data from the buffer. Inter Thread communication is essential in this scenario to ensure that the buffer is not accessed by both threads simultaneously, which could lead to data corruption.

26)How can you ensure that even and odd threads print natural numbers alternately without any overlap?
You can achieve this by using inter thread communication mechanisms like wait() and notify() or notifyAll(). The even thread can print an even number and then call wait() to pause its execution, and the odd thread

can print the odd number and call notify() or notifyAll() to wake up the even thread.

27)What are some common String operations that you can perform in Java?
Some common String operations include concatenation, substring extraction, character extraction, length calculation, case conversion, searching, and replacing.

28)What is the purpose of StringBuffer in Java, and how is it different from String?
StringBuffer is used to create mutable strings, meaning the contents can be modified after the object creation. In contrast, String objects are immutable, and their values cannot be changed once created.

29)What is string builder
StringBuilder is a class in Java used for mutable sequences of characters, allowing efficient string manipulations.

30)What is the difference between StringBuffer and StringBuilder in Java?
Both StringBuffer and StringBuilder are used to create mutable strings, but the main difference is that StringBuffer is thread-safe (synchronized), while StringBuilder is not. If you are working in a single-threaded environment, StringBuilder is generally preferred due to its higher performance.

31)What is string pooling?
- String pooling is a technique where multiple String objects with the same content share the same memory location.
- It saves memory by reusing existing strings instead of creating new objects for the same value.

32)What is valueOf method?
valueOf() is a static method in wrapper classes that converts a primitive data type or a string representation to its corresponding wrapper object.

33)What are wrapper classes?

- Wrapper classes are classes in Java that provide a way to convert primitive data types into objects and vice versa.
- They allow using primitive data types as objects when needed, as they are part of the Java Collections Framework.

34)What is autoboxing and auto unboxing?

- Autoboxing is the process of automatically converting a primitive type to its corresponding wrapper class object.
- Auto unboxing is the process of automatically converting a wrapper class object back to its primitive type.

35)What is an Enumeration in Java, and how is it different from other collection types?

Enumeration is an older interface used to iterate through elements of legacy collections like Vector and Hashtable. It provides methods like hasMoreElements() and nextElement() to traverse the collection. Enumerations are read-only and do not support removal of elements.

36)What is annotation?

Annotations are a form of metadata in Java that provide additional information about code elements, like classes, methods, etc., used by compilers, runtime, and other tools.

37)How do you define a custom annotation in Java?

To create a custom annotation, you define an interface with the @interface keyword. You can then specify elements (values) that can be used when annotating code elements.

38)What is the work flow of annotation behind the scene?

During compilation, the compiler processes annotations and generates additional information about annotated elements in the compiled class file.

At runtime, the Java runtime environment can access this information and perform specific actions based on the annotations.

39) How do you represent a file in Java, and what properties can you obtain from a File object?
In Java, you can represent a file using the File class. Some properties that you can obtain from a File object include file name, file path, file size, last modified timestamp, etc.

40) What are different byte streams in java?
Byte streams handle binary data. Example InputStream, OutputStream, FileInputStream, FileOutputStream, ByteArrayInputStream, ByteArrayOutputStream

41)What is the purpose of the FilenameFilter interface in Java?
The FilenameFilter interface is used to filter filenames when listing files in a directory. It allows you to include or exclude files based on specific criteria.

42) What are Byte Streams in Java, and how are they used for file handling?
Byte Streams in Java are used to handle binary data, such as reading and writing raw bytes from and to files. FileInputStream and FileOutputStream are examples of Byte Streams that can be used to copy contents from one file to another.

43) What are Character Streams in Java, and how are they different from Byte Streams?
Character Streams in Java are used to handle character data, such as reading and writing text-based data from and to files. Unlike Byte Streams, which deal with raw bytes, Character Streams work with characters and provide encoding and decoding capabilities.

44)What is the purpose of using BufferedInputStream in Java file handling?
BufferedInputStream is used to improve the performance of file reading by reducing the number of read operations and minimizing the overhead involved in reading data from the file.

45)How does BufferedOutputStream improve the performance of file writing in Java?
BufferedOutputStream improves file writing performance by reducing the number of write operations and buffering the data in memory before writing it to the file in larger chunks, thus minimizing the overhead.

46) What is the benefit of using BufferedReader and BufferedWriter in file handling?
BufferedReader and BufferedWriter are used to read and write text-based data more efficiently. They read and write data in larger chunks, reducing the number of IO operations and enhancing performance.

47)What is serialization and deserialization and why are they used?
Serialization is the process of converting an object into a stream of bytes so that it can be stored in memory, transmitted over the network, or saved to a file. Deserialization is the process of reconstructing the object from the serialized form. They are used to persist objects and exchange data between different Java applications.

48)What is the transient keyword?
- transient is a keyword used with instance variables in Java.
- It indicates that the variable should not be serialized during object serialization, and its value will not be persisted.

49)What is an array list?
ArrayList is a dynamic array-like data structure in Java that can resize itself to accommodate elements. It's part of the Java Collections Framework.

50) How do you create an ArrayList to store a list of objects in Java?
You can create an ArrayList to store objects by specifying the type of objects it will hold. For example, ArrayList<Employee> will create an ArrayList to store Employee objects.

51)What is a linked list
LinkedList is a linear data structure in Java, where each element is stored in a node with a reference to the next node, allowing for efficient insertion and deletion operations.

52)What is the difference between ArrayList and LinkedList in Java?
ArrayList is based on an array, while LinkedList is based on linked nodes. ArrayList provides faster access to elements using index, but LinkedList is more efficient in insertion and deletion operations.

53)What is a HashSet in Java, and what is its primary feature?
HashSet is a collection that stores unique elements and does not allow duplicates. It uses the hashing mechanism to quickly look up elements, making it ideal for scenarios where you need to maintain a unique set of items.

54)What is treeset
TreeSet is a sorted set in Java that implements the SortedSet interface. It stores elements in ascending order, based on natural ordering or a custom comparator.

55)How is TreeSet different from HashSet in Java?
TreeSet is similar to HashSet, but it stores elements in a sorted and ascending order. It is implemented using a red-black tree, which guarantees log(n) time complexity for insertion, deletion, and search operations.

56) What is a hash map?
HashMap is a class in Java that implements the Map interface. It stores key-value pairs and provides fast access and retrieval operations.

JAVA VIVA

Arrays in Java:

1. What is an array in Java?
   - Answer: An array is a data structure that stores a fixed-size sequential collection of elements of the same type.

2. How do you declare and initialize an array in Java?
   - Answer: You can declare and initialize an array in Java using the following syntax:
   ```
   dataType[] arrayName = new dataType[arraySize];
   ```

3. How do you access elements in an array?
   - Answer: You can access elements in an array using their index. The index of the first element is 0, and the index of the last element is `array.length - 1`.

4. What is a jagged array?
   - Answer: A jagged array is an array of arrays where each sub-array can have a different length. It allows creating arrays with varying lengths.

Interfaces:

1. What is an interface in Java?
   - Answer: An interface in Java is a collection of abstract methods and constants. It provides a way to define a contract for classes that implement the interface.

2. Can an interface have instance variables?
   - Answer: No, an interface cannot have instance variables. It can only have constants (public static final variables).

3. Can a class implement multiple interfaces in Java?
   - Answer: Yes, Java supports multiple inheritance through interfaces. A class can implement multiple interfaces by separating them with commas.

4. Why can't we implement multiple inheritance at class level in Java?
 - Answer:In Java, multiple inheritance is not allowed at the class level to avoid the complications that arise from the Diamond Problem.

The Diamond Problem occurs when a class inherits from two or more classes that have a common superclass. If multiple inheritance were allowed at the class level, it would be possible to inherit conflicting or ambiguous implementations of methods and members from different parent classes.

Inner Classes:

1. What is an inner class in Java?
   - Answer: An inner class is a class defined within another class. It can access members of the outer class, including private members.

2. What are the types of inner classes in Java?
   - Answer: Java supports four types of inner classes: nested inner class, local inner class, anonymous inner class, and static nested class.

Anonymous Inner Class:

1. What is an anonymous inner class?
   - Answer: An anonymous inner class is a class without a name that is defined and instantiated at the same time. It is useful when you need to create a class with a single instance.

2. How do you create an anonymous inner class?
   - Answer: Anonymous inner classes can be created by extending a class or implementing an interface directly at the point of instantiation.

Static:

1. What does the "static" keyword mean in Java?
   - Answer: The "static" keyword is used to define a member (variable or method) that belongs to the class itself rather than to any instance of the class.

2. Can static methods access instance variables?
   - Answer: No, static methods cannot directly access instance variables because they are not associated with any instance. However, they can access static variables.

Static Nested Classes:

1. What is a static nested class in Java?
   - Answer: A static nested class is a nested class that is a static member of the outer class. It does not require an instance of the outer class to be instantiated.

2. How is a static nested class different from an inner class?
   - Answer: A static nested class is associated with the outer class itself, whereas an inner class is associated with an instance of the outer class.

Concrete Class:

1. What is a concrete class in Java?
   - Answer: A concrete class is a class that can be instantiated and can provide implementations for all its inherited abstract methods.

2. Can a concrete class be inherited by another class?
   - Answer: Yes, a concrete class can be inherited by another class using the "extends" keyword.

Static Methods:

1. What is a static method in Java?
   - Answer: A static

 method is a method that belongs to the class itself rather than to any instance of the class. It can be called using the class name.

2. Can a static method access non-static (instance) variables?
   - Answer: No, a static method cannot directly access non-static variables. It can only access static variables.

Functional Interface:

1. What is a functional interface in Java?
   - Answer: A functional interface is an interface that contains only one abstract method. It is used to implement functional programming concepts in Java.

2. What is the purpose of the `@FunctionalInterface` annotation?
   - Answer: The `@FunctionalInterface` annotation is used to ensure that the interface is intended to be a functional interface. It generates a compilation error if the interface doesn't meet the requirements.

Inheritance in Java:

1. What is inheritance in Java?
   - Answer: Inheritance is a mechanism in Java that allows a class to inherit properties and behaviors (methods and variables) from another class.

2. Why is multiple inheritance not possible at the class level in Java?
   - Answer: Multiple inheritance at the class level is not allowed in Java to avoid complications arising from conflicts that may occur when two or more superclasses have methods with the same signature.

Final Keyword:

1. What does the "final" keyword mean in Java?
   - Answer: The "final" keyword can be applied to classes, methods, and variables. It indicates that the entity is constant and cannot be modified or extended.

2. What is the difference between "final" and "finalize"?
   - Answer: "final" is a keyword used to define constants, mark classes as unmodifiable, and prevent method overriding. "finalize" is a method in the Object class that is called by the garbage collector before an object is garbage collected.

Abstract Class in Java:

1. What is an abstract class in Java?
   - Answer: An abstract class is a class that cannot be instantiated. It is designed to be a base class for other classes and may contain abstract methods that must be implemented by its subclasses.

2. Can an abstract class have a constructor?
   - Answer: Yes, an abstract class can have constructors. However, it cannot be directly instantiated, so the constructors are typically used to initialize the inherited members.

Dynamic Method Dispatch:

1. What is dynamic method dispatch in Java?
   - Answer: Dynamic method dispatch is the mechanism in Java that allows a subclass object to be treated as an instance of its superclass. It is essential for achieving runtime polymorphism.

2. How does dynamic method dispatch work?
   - Answer: When an overridden method is called on a superclass reference that points to a subclass object, the overridden method in the subclass is executed instead of the superclass's implementation.

Method Overriding:

1. What is method overriding in Java?
   - Answer: Method overriding is a feature in Java that allows a subclass to provide a specific implementation of a method that is already defined in its superclass.

2. What are the rules for method overriding?
   - Answer: The rules for method overriding in Java are:
     - The method in the subclass must have the same name, return type, and parameters as the method in the superclass.
     - The access level of the subclass method cannot be more restrictive than the superclass method.
     - The subclass method can throw the same, narrower, or no exceptions compared to the superclass method.

Method Overloading:

1. What is method overloading in Java?
   - Answer: Method overloading is a feature in Java that allows a class to have multiple methods with the same name but different parameters.

2. How is method overloading determined at compile-time?
   - Answer: Method overloading

 is determined at compile-time based on the number, types, and order of the parameters in the method signature.

Copy Constructor:

1. What is a copy constructor in Java?
   - Answer: A copy constructor is a special constructor that creates a new object by copying the contents of an existing object of the same class.

2. When is a copy constructor used?
   - Answer: A copy constructor is used to create a new object with the same state as an existing object. It is often used in situations where deep copying is required.

Constructor Overloading:

1. What is constructor overloading in Java?
   - Answer: Constructor overloading is the practice of having multiple constructors in a class, each with a different parameter list.

2. What is the benefit of constructor overloading?
   - Answer: Constructor overloading provides flexibility in object creation by allowing different ways to initialize an object based on different parameter combinations.

Garbage Collection in Java:

1. What is garbage collection in Java?
   - Answer: Garbage collection is the process in Java where the JVM automatically reclaims memory that is no longer in use by objects. It frees the programmer from explicitly managing memory deallocation.

2. Why is garbage collection needed in Java?
   - Answer: Garbage collection is needed in Java to automatically deallocate memory occupied by objects that are no longer referenced. It prevents memory leaks and simplifies memory management.

Super Keyword:

1. What is the "super" keyword in Java?
   - Answer: The "super" keyword is used to refer to the superclass of a subclass. It can be used to call superclass constructors, access superclass members, and invoke superclass methods.

2. How is the "super" keyword different from "this" keyword?
   - Answer: The "super" keyword is used to refer to the superclass, whereas the "this" keyword is used to refer to the current instance of the class.

These are just sample questions and answers. You can further expand on each topic and explore more questions and concepts related to them for your viva preparation.

import java.util.Scanner; what is happening in this line?
   - Answer:The line import java.util.Scanner; is an import statement in Java. It is used to import the Scanner class from the java.util package into the current Java file.

The Scanner class is a part of the Java Standard Library and is commonly used for reading user input from various sources, such as the keyboard (standard input) or files.

public static void main(String[] args), Explain this line?
  - Answer: In Java, the main method serves as the entry point of a program. When you run a Java program, the Java Virtual Machine (JVM) starts executing the program by invoking the main method.

Let's break down the different parts of the line:

public: It is an access modifier that specifies the visibility of the main method. In this case, public means that the main method can be accessed from any other class.

static: It is a keyword that indicates that the main method belongs to the class itself and not to an instance of the class. This allows the main method to be called without creating an object of the class.

void: It is the return type of the main method. The void keyword means that the main method does not return any value.

main: It is the name of the method. By convention, the entry point method in Java is named main.

String[] args: It is the parameter of the main method. It represents an array of strings and allows you to pass command-line arguments to the program. The args parameter can be used to receive and process command-line inputs during program execution.

Access specifiers in java
- Answer: There are four access specifiers in Java:

public: The public access specifier allows unrestricted access to the class, method, variable, or constructor from anywhere in the program. It has the widest scope and is commonly used for methods or members that need to be accessible to all other classes.

private: The private access specifier restricts access to the class, method, variable, or constructor within the same class. It is commonly used to encapsulate implementation details and provide data hiding.

protected: The protected access specifier allows access to the class, method, or variable within the same package or subclasses in different packages. It provides a level of access between public and private and is commonly used in inheritance scenarios.

Default (no specifier): If no access specifier is specified, it is considered the default access specifier. The default access specifier allows access to the class, method, or variable within the same package only. It is not accessible from outside the package.

Object-Oriented Programming (OOP) is a programming paradigm that emphasizes the concept of objects, which can contain data and code to

manipulate that data. Java is a widely used programming language that fully supports OOP principles. Let's discuss the main OOP concepts and their sub-concepts in Java:

1. **Class:** A class is a blueprint or template that defines the properties (attributes) and behaviors (methods) that objects of the class can possess. It serves as a blueprint for creating objects.

   - **Constructor:** A constructor is a special method that is called when an object is created. It is used to initialize the object's state and allocate memory.

   - **Instance Variables:** Instance variables are the attributes or data members of a class. They represent the state of an object and are declared within a class but outside any method.

   - **Methods:** Methods are functions defined in a class that define the behavior of objects. They encapsulate reusable code and allow objects to perform actions and manipulate data.

2. **Object:** An object is an instance of a class. It represents a real-world entity and has its own state and behavior. Objects are created using the `new` keyword and can access the class's methods and variables.

3. **Inheritance:** Inheritance is a mechanism that allows a class to inherit properties and behaviors from another class. It promotes code reuse and supports the concept of "is-a" relationship.

   - **Superclass and Subclass:** The class from which properties and behaviors are inherited is called the superclass or base class, while the class that inherits those properties and behaviors is called the subclass or derived class.

   - **Method Overriding:** Method overriding occurs when a subclass provides its own implementation of a method that is already defined in its

superclass. It allows the subclass to modify the behavior of the inherited method.

- **Method Overloading:** Method overloading is the ability to define multiple methods with the same name but different parameters in a class. It allows methods to perform similar operations with different inputs.

4. **Polymorphism:** Polymorphism means the ability of an object to take on many forms. In Java, polymorphism is achieved through method overriding and method overloading.

- **Runtime Polymorphism:** Runtime polymorphism is achieved when a method is overridden in the subclass and the appropriate implementation is determined at runtime based on the actual object type.

- **Compile-time Polymorphism:** Compile-time polymorphism is achieved through method overloading, where different methods with the same name but different parameters are resolved at compile-time based on the method arguments.

5. **Encapsulation:** Encapsulation is the process of hiding the internal details of an object and providing access to only the essential features. It helps in achieving data abstraction, data integrity, and modularity.

- **Access Modifiers:** Access modifiers like `public`, `private`, `protected`, and default (no modifier) control the visibility and accessibility of classes, methods, and variables.

6. **Abstraction:** Abstraction is the process of representing complex real-world entities as simplified models in code. It focuses on the essential features and hides unnecessary details.

- **Abstract Class:** An abstract class is a class that cannot be instantiated and is typically used as a base class for other classes. It may contain abstract methods, which are methods without an implementation.

- **Interface:** An interface is a collection of abstract methods that define a contract. Classes can implement interfaces to provide a common behavior across different classes.

7. **Association, Aggregation, and Composition:** These are types of relationships between classes to represent dependencies and connections.

8. **Static:** The `static` keyword is used to define class-level members that belong to the class itself rather than specific instances of the