Java Interview Questions

1.what is java?

Java is a high-level, class-based, object-oriented programming language that is designed to be platform-independent, allowing developers to write code once and run it anywhere.

2.what are the key features of java.

1. **Platform Independence**: Java code is compiled into bytecode, which can be run on any device equipped with a Java Virtual Machine (JVM), making Java platform-independent.
2. **Object-Oriented**: Java is based on the object-oriented programming model, which helps in organizing complex programs into reusable, modular components.
3. **Simple and Familiar**: Java is designed to be easy to use, with a syntax that is clear and concise. It also builds on familiar concepts from C and C++.
4. **Robust**: Java emphasizes early checking for possible errors, with a strong type-checking mechanism and exception handling features to create reliable programs.

5.**High Performance**: Java achieves high performance through Just-In-Time (JIT) compilers that compile bytecode into native machine code at runtime.

6.**MultiThreading** : Java supports multithreading, which allows multiple threads to run concurrently, making it suitable for applications that require efficient performance, such as web servers and GUI applications.

3.what is meant by JVM, JRE, JDK.

JVM

The Java Virtual Machine (JVM) is an abstract computing machine that enables a computer to run a Java program. It provides a runtime environment in which Java bytecode can be executed.

The JVM allows Java programs to be written once and run anywhere (WORA). This means Java bytecode can be executed on any system that has a JVM, regardless of the underlying hardware and operating system.

When a Java program is compiled, it is converted into bytecode, which is a platform-independent code. The JVM interprets or compiles this bytecode into machine code that can be executed by the host machine.

The JVM includes an automatic garbage collector that manages memory allocation and deallocation, reducing the likelihood of memory leaks and other related issues.

JRE

The Java Runtime Environment (JRE) is a software package that provides the libraries, Java Virtual Machine (JVM), and other components necessary to run applications written in the Java programming language. It does not include tools for Java development, such as compilers or debuggers.

Key components of the JRE include:

1. **Java Virtual Machine (JVM)**: The core component of the JRE, which executes Java bytecode.
2. **Class Libraries**: A set of standard libraries that provide commonly used functionality, such as data structures, networking, file I/O, and graphical user interfaces.
3. **Class Loaders**: Components that load Java classes into the JVM, enabling dynamic loading of classes at runtime.
4. **Other Support Files**: Various files that support the execution of Java applications, including configuration files and property settings.

JDK

The Java Development Kit (JDK) is a software development kit provided by Oracle Corporation and other vendors used to develop Java applications and applets. It includes tools and utilities necessary for Java development, in addition to the components provided by the Java Runtime Environment (JRE).

JDK is a comprehensive package that includes everything a developer needs to write, compile, test, and debug Java applications. It is essential for Java development, whereas the JRE is primarily used for running Java applications.

Key components of the JDK include:

1. **Java Runtime Environment (JRE)**: This includes the JVM, core libraries, and other files needed to run Java applications.

2. **Development Tools**: Tools for developing, debugging, and monitoring Java applications, such as:
    - **javac**: The Java compiler, which converts Java source code into bytecode.
    - **java**: The launcher for running Java applications.
    - **javadoc**: A documentation generator that creates HTML documentation from Java source code comments.
    - **jdb**: A debugger for Java programs.
    - **jar**: A tool for creating and managing Java Archive (JAR) files.
3. **Additional Libraries**: Extra libraries and frameworks that are useful for developing Java applications.

- **JVM (Java Virtual Machine):** Executes Java programs. It's the engine that runs Java bytecode on any device, ensuring that Java programs can run on different platforms without modification.
- **JRE (Java Runtime Environment):** Allows you to run Java programs. It includes the JVM and the necessary libraries and components to execute applications written in Java.
- **JDK (Java Development Kit):** Provides the tools for developing Java programs. It includes the JRE, plus the compilers and other tools needed to write, compile, and debug Java applications.

4. Is Java Platform Independent if then how?

Yes, Java is platform-independent. This means that Java programs can run on any device with a Java Virtual Machine (JVM), regardless of the underlying hardware and operating system.

- Java source code is written in **.java** files.
- This source code is compiled by the Java compiler (**javac**) into an intermediate form called bytecode, stored in **.class** files.
- Bytecode is a platform-independent code that can be executed on any system with a JVM.
- The JVM reads and executes the bytecode, translating it into machine code appropriate for the host system.

5.what is JIT.

JIT, or Just-In-Time compilation, is a feature of the Java Virtual Machine (JVM) that enhances the performance of Java applications.

It is a process where the JVM compiles Java bytecode into native machine code at runtime. This machine code can then be executed directly by the computer's hardware, resulting in faster performance.

- When you compile a Java program, the Java compiler (javac) converts the human-readable source code into an intermediate form called bytecode. This bytecode is platform-independent and is designed to be executed by the JVM.
- The JVM can interpret this bytecode and execute it line-by-line. While interpretation is straightforward, it can be slower compared to native machine code execution.
- Instead of interpreting bytecode line-by-line, the JVM includes a JIT compiler that compiles bytecode into native machine code at runtime. This native code can be executed directly by the CPU, resulting in much faster execution.
- The JIT compiler doesn't compile all bytecode upfront. Instead, it monitors which parts of the code are executed frequently (called "hot spots"). When a method or a block of code is identified as a hot spot, the JIT compiler compiles that code to native machine code and stores it in memory for future use.

6.What are the different memory stages available in JVM

the different memory stages available in the JVM:

1. **Method Area**:
    - Stores class-level data, such as class structures (metadata), constants, static fields, and method data.
    - Shared among all threads.
2. **Heap**:
    - The largest memory area in the JVM.
    - Used for dynamic memory allocation, where all class instances (objects) and arrays are allocated.

- Managed by the garbage collector, which reclaims memory used by objects that are no longer needed.

3. **Stack**:
   - Each thread has its own stack.
   - Stores local variables, method call information (such as return addresses), and partial results.
   - Follows the Last-In-First-Out (LIFO) principle.
   - Stack frames are created for each method invocation and destroyed when the method call is completed.
4. **Program Counter (PC) Register**:
   - Each thread has its own PC register.
   - Stores the address of the current instruction being executed.
   - Acts as a pointer to the next instruction to be executed in the thread's method.

7.Difference between Java and Python.

Java	Python
1.Java follows a proper structure while writing the code.	1.Python does not have any pre defined structure to write the code.
2.Requires compilation before execution, resulting in faster performance.	2.Interpreted language, often slower than Java, especially for CPU-intensive tasks.
3.java syntax is some what difficult to understand when compared to python.	3.Python syntax is more easier to learn when compared to java.
4.Steep learning curve for beginners while learning Java.	4.Easy to learn due to its simple syntax.
5.Java variables need to be declared with its datatype before compilation.	5.Python variables are dynamically typed which does not need explicit declaration.
6.Java catches errors at compile time.	6.Python catches errors at run time.

8.Explain public static void main(String[] args) in java.

1. **public**: This keyword is an access modifier that specifies the visibility of the method. In this case, **public** means that the method can be accessed from outside the class.
2. **static**: This keyword indicates that the method belongs to the class itself, rather than to instances of the class. It allows the method to be called without creating an instance of the class.
3. **void**: This keyword specifies that the method does not return any value.
4. **main**: This is the name of the method. In Java, the **main** method is the entry point for a standalone Java application. It is where the program starts execution.
5. **(String[] args)**: This is the parameter list of the **main** method. It specifies that the method takes an array of strings as input. The **args** parameter allows the program to accept command-line arguments when it is run. Each element in the array represents a command-line argument passed to the program.

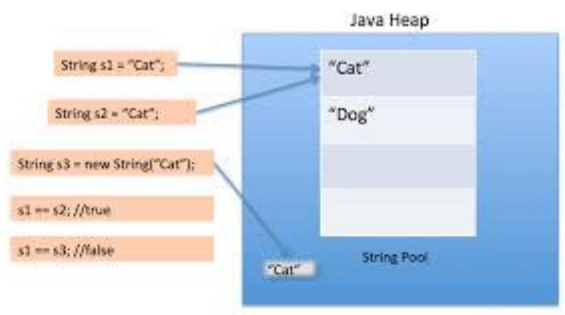
9. What will happen if we declare don't declare the main as static?

If you declare the main method in Java without the static keyword, the Java Virtual Machine (JVM) will not be able to recognize it as the entry point of the program. Consequently, you won't be able to run the program directly from the command line or through an IDE, and you'll encounter a runtime error.

The **main** method needs to be declared as **static** because it belongs to the class itself, not to any specific instance of the class. This allows the JVM to call the **main** method without creating an instance of the class. If the **main** method is not static, it would require an instance of the class to call it, which contradicts the purpose of being an entry point for the program.

When you run a Java program, the JVM looks for the **main** method with a specific signature (**public static void main(String[] args)**) to start the execution of the program. If the **main** method is not static, the JVM won't recognize it as the entry point, and the program won't execute as expected.

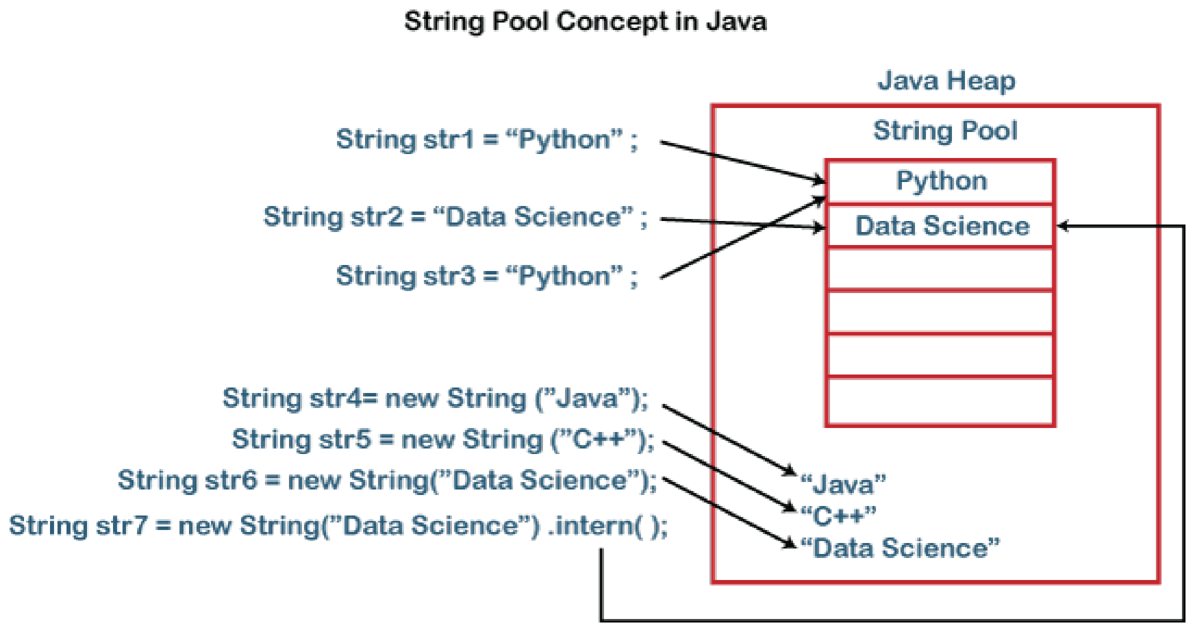
10. What is Java String Constant pool.



The Java String Constant Pool is a special area of memory in the Java Virtual Machine (JVM) that stores unique string literals.

- When you create a string literal in Java using double quotes, such as **"hello"**, the JVM checks if an identical string already exists in the constant pool.
- If an identical string is found, the new string literal does not create a new instance; instead, it references the existing string in the constant pool.
- This process is called string interning and helps conserve memory by avoiding duplicate string objects.

11.What is intern() method in java?


String Pool Concept in Java
Java Heap
String Pool
String str1 = "Python" ;
String str2 = "Data Science" ;
String str3 = "Python" ;
Python
Data Science
String str4= new String ("Java");
String str5 = new String ("C++");
String str6 = new String("Data Science");
String str7 = new String("Data Science") .intern( );
"Java"
"C++"
"Data Science"

The **intern()** method in Java is used to place the given string into the string constant pool of the JVM, if it is not already present.

When you call the **intern()** method on a string object, the JVM first checks if an identical string is already present in the string constant pool.

If an identical string is not found in the pool, the string is added to the pool, and the intern() method returns a reference to the string in the pool.

If an identical string is already present in the pool, the **intern()** method returns a reference to the existing string in the pool.

12.What are the advantages of Packages in Java?

Packages in Java provide several advantages that contribute to better organization, modularity, and maintainability of the code. Here are some key advantages:

- Packages help avoid naming conflicts by organizing classes into namespaces. This means that classes with the same name can coexist in different packages.
- By categorizing classes into packages, it becomes easier to understand the structure and functionality of the application.
- Packages support access control, which allows you to define the accessibility of classes, interfaces, and members.
- Packages make it easier to reuse code across different projects. Commonly used utilities and classes can be placed in separate packages and imported whenever needed.
- Packages can be bundled into JAR (Java Archive) files, which simplifies the deployment and distribution of Java applications.

13.What are the different data types in java?

Primitive data types are predefined by the language and are named by a reserved keyword. There are eight primitive data types in Java:

1byte = 8bits

1. **byte**
    - **Size**: 8 bits / 1byte
    - **Range**: -128 to 127
    - **Default Value**: 0
2. **short**

- **Size**: 16 bits / 2bytes
- **Range**: -32,768 to 32,767
- **Default Value**: 0

3. **int**
    - **Size**: 32 bits / 4bytes
    - **Range**: -2^31 to 2^31-1
    - **Default Value**: 0
4. **long**
    - **Size**: 64 bits / 8bytes
    - **Range**: -2^63 to 2^63-1
    - **Default Value**: 0L
5. **float**
    - **Size**: 32 bits / 4bytes
    - **Range**: Approximately ±3.40282347E+38F (6-7 significant decimal digits)
    - **Default Value**: 0.0f
6. **double**
    - **Size**: 64 bits / 8bytes
    - **Range**: Approximately ±1.79769313486231570E+308 (15 significant decimal digits)
    - **Default Value**: 0.0d
7. **boolean**
    - **Size**: Not precisely defined (commonly represented as a single bit)
    - **Range**: **true** or **false**
    - **Default Value**: **false**
8. **char**
    - **Size**: 16 bits / 2bytes
    - **Range**: '\u0000' (0) to '\uffff' (65,535)

- **Default Value**: '\u0000'

Reference Data Types

Reference data types are used to refer to objects. They are created using defined constructors of the classes. They include:

1. **Classes**
   - Reference type that can have fields, methods, constructors, and nested classes/interfaces.
   - **Example**: **String**, **Scanner**, **ArrayList**.
2. **Interfaces**
   - A reference type, similar to a class, that can contain only constants, method signatures, default methods, static methods, and nested types.
   - **Example**: **List**, **Map**.
3. **Arrays**
   - Containers that hold a fixed number of values of a single type.
   - **Example**: **int[]**, **String[]**.
4. **Enumerations (enum)**
   - Special data types that enable a variable to be a set of predefined constants.
   - **Example**:

     enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY }

Differences Between Primitive and Reference Types

- **Memory Allocation**: Primitive types are stored directly in the memory allocated for the variable, whereas reference types store a reference (or address) to the actual data (object) in the heap.
- **Default Values**: Primitive types have default values (e.g., 0 for integers, false for boolean), while reference types default to **null**.
- **Methods**: Only reference types can have methods, while primitive types cannot.

13.What is a wrapper class and why do we need it in java.

A wrapper class in Java is an object representation of a primitive data type. Java provides a wrapper class for each of the eight primitive data types, allowing these primitive types to be used as objects. The wrapper classes are part of the **java.lang** package.

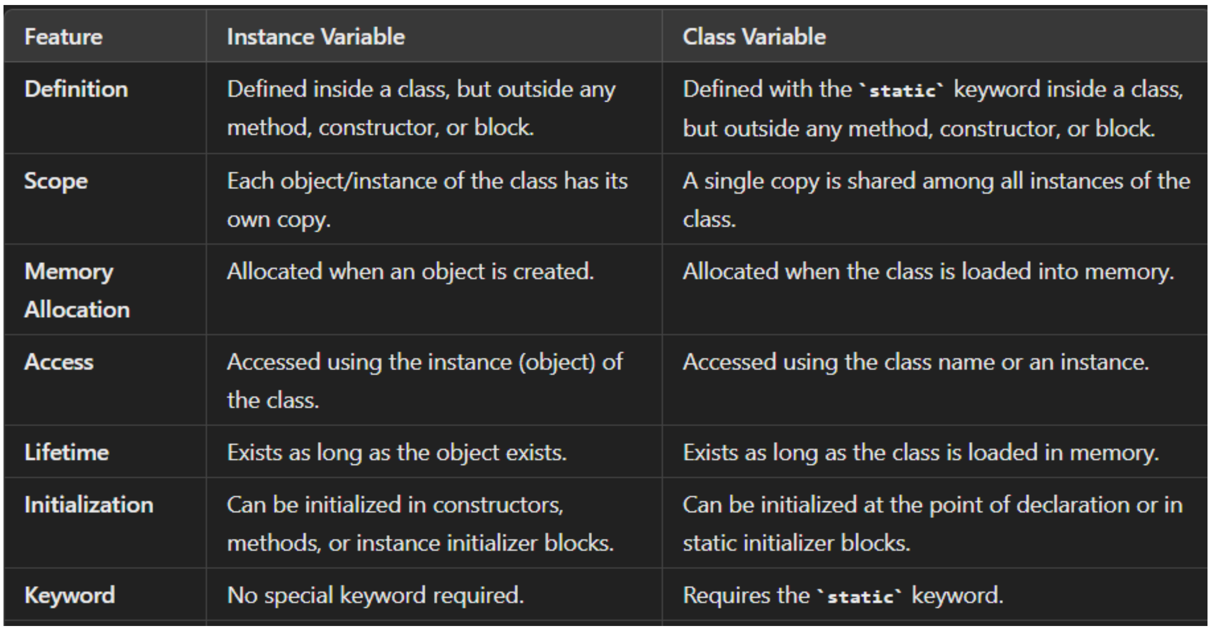
Here are the primitive types and their corresponding wrapper classes:

- **byte → Byte**
- **short → Short**
- **int → Integer**
- **long → Long**
- **float → Float**
- **double → Double**
- **char → Character**
- **boolean → Boolean**

Why we need wrapper classes :

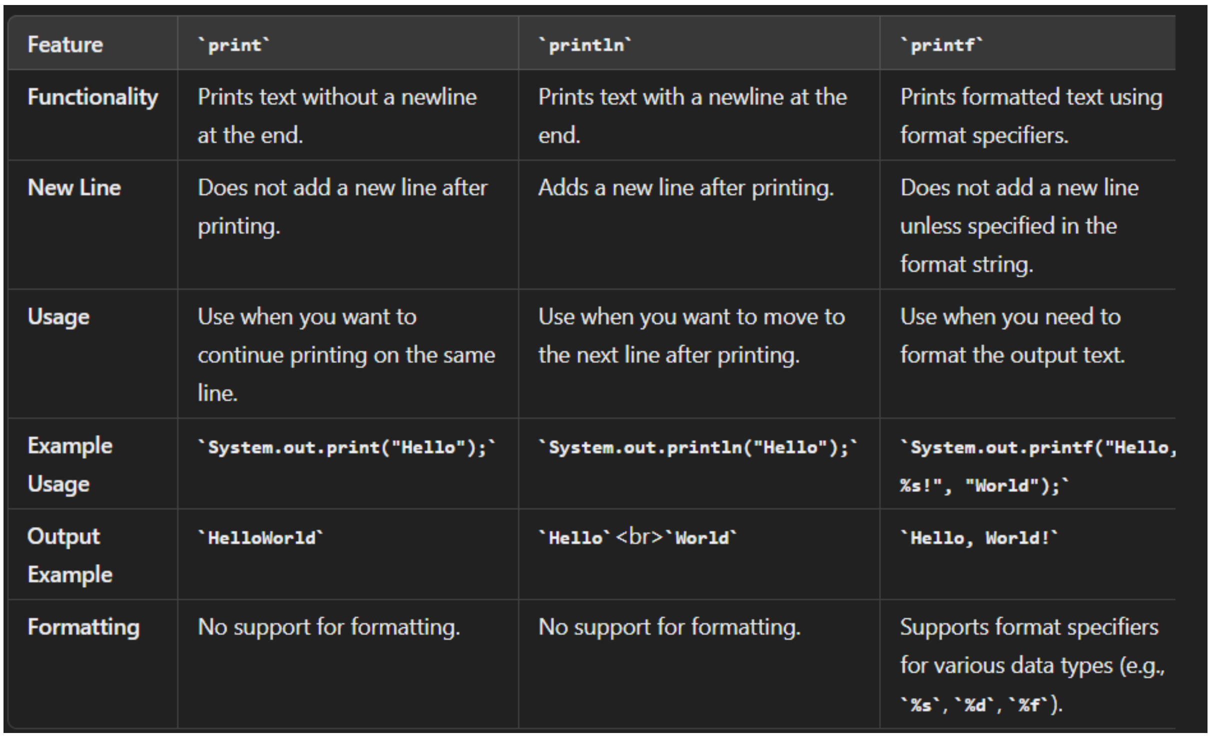
- Java's collection framework (e.g., **ArrayList**, **HashMap**) works with objects and does not support primitive types directly. Wrapper classes allow primitives to be used in collections.
- Each wrapper class provides utility methods for converting values to other types, parsing strings, and other useful operations.
- Java generics work only with objects, not primitive types. Wrapper classes allow primitives to be used in generic classes, methods, and interfaces.

14.Explain the difference between instance variable and a class variable.



Feature	Instance Variable	Class Variable
Definition	Defined inside a class, but outside any method, constructor, or block.	Defined with the `static` keyword inside a class, but outside any method, constructor, or block.
Scope	Each object/instance of the class has its own copy.	A single copy is shared among all instances of the class.
Memory Allocation	Allocated when an object is created.	Allocated when the class is loaded into memory.
Access	Accessed using the instance (object) of the class.	Accessed using the class name or an instance.
Lifetime	Exists as long as the object exists.	Exists as long as the class is loaded in memory.
Initialization	Can be initialized in constructors, methods, or instance initializer blocks.	Can be initialized at the point of declaration or in static initializer blocks.
Keyword	No special keyword required.	Requires the `static` keyword.

15.Difference in the use of print, println, and printf.



Feature	`print`	`println`	`printf`
Functionality	Prints text without a newline at the end.	Prints text with a newline at the end.	Prints formatted text using format specifiers.
New Line	Does not add a new line after printing.	Adds a new line after printing.	Does not add a new line unless specified in the format string.
Usage	Use when you want to continue printing on the same line.	Use when you want to move to the next line after printing.	Use when you need to format the output text.
Example Usage	`System.out.print("Hello");`	`System.out.println("Hello");`	`System.out.printf("Hello, %s!", "World");`

Output Example	`HelloWorld`	`Hello`
		 `World`
			`Hello, World!`

Formatting	No support for formatting.	No support for formatting.	Supports format specifiers for various data types (e.g., `%s`, `%d`, `%f`).

16.How many ways you can take input from the console?

The **Scanner** class, part of the **java.util** package, is the most commonly used way to take input from the console. It can parse primitive types and strings.

import java.util.Scanner;

```
public class ScannerExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter an integer: ");
        int num = scanner.nextInt();
        System.out.print("Enter a string: ");
        String str = scanner.next();
        System.out.print("Enter a float: ");
        float f = scanner.nextFloat();
        scanner.close();
        System.out.println("You entered: " + num + ", " + str + ", " + f);
    }
}
```

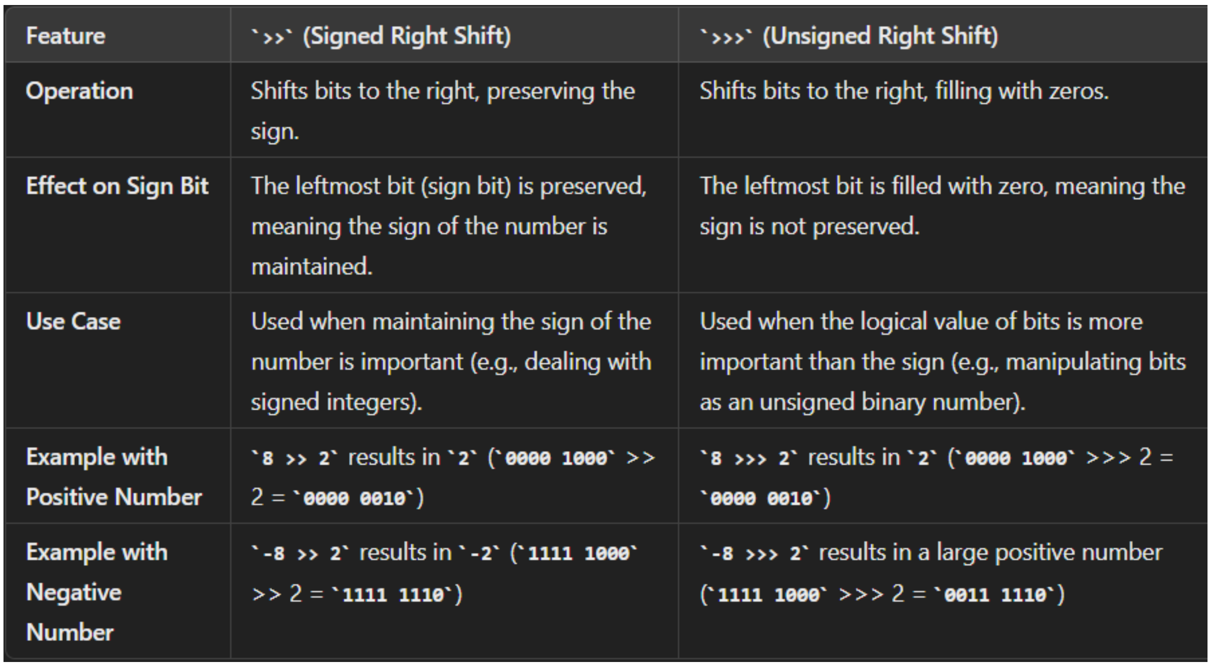
This method involves using **BufferedReader** in conjunction with **InputStreamReader**. It is efficient for reading lines of text.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class BufferedReaderExample {
    public static void main(String[] args) throws IOException {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Enter a line of text: ");
        String line = reader.readLine();
        System.out.print("Enter a number: ");
        int num = Integer.parseInt(reader.readLine());
        System.out.println("You entered: " + line + ", " + num);
    }
}
```

You can also take input directly from the command line when running your program.

```
public class CommandLineExample {
    public static void main(String[] args) {
        if (args.length < 2) {
            System.out.println("Please provide two inputs");
            return;
        }
        String firstArg = args[0];
        String secondArg = args[1];
        System.out.println("First argument: " + firstArg);
        System.out.println("Second argument: " + secondArg);
    }
}
```

17.Explain the difference between >> and >>> operators.



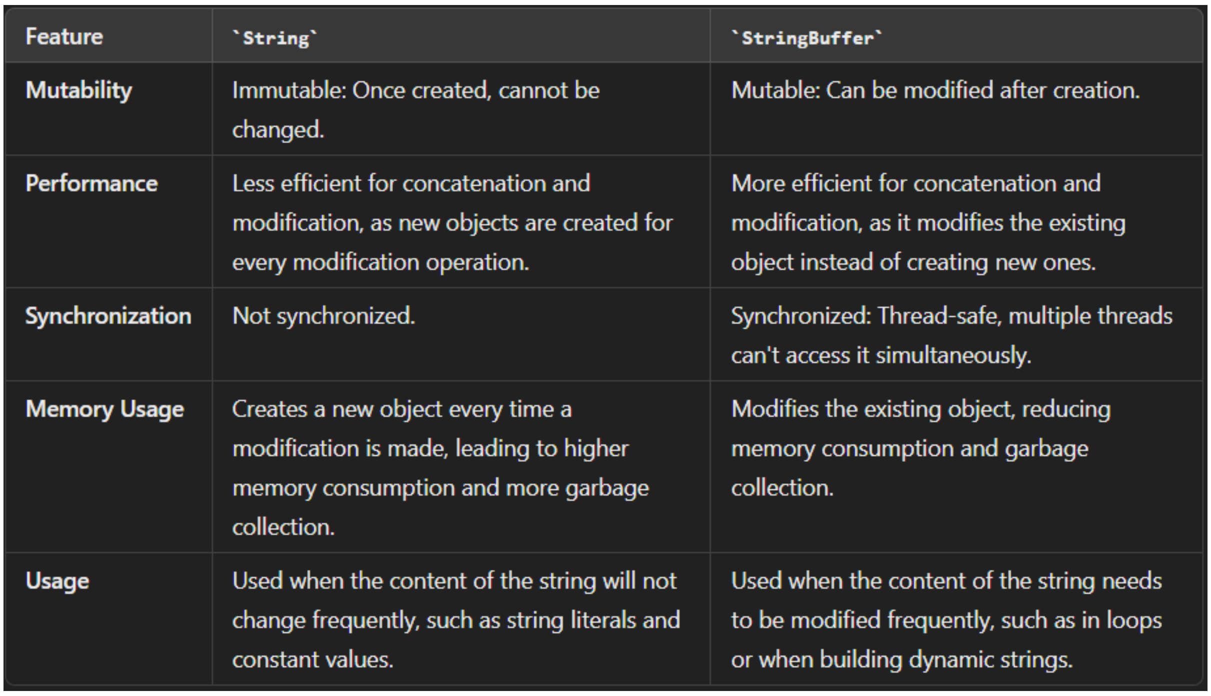
Feature	`>>` (Signed Right Shift)	`>>>` (Unsigned Right Shift)
Operation	Shifts bits to the right, preserving the sign.	Shifts bits to the right, filling with zeros.
Effect on Sign Bit	The leftmost bit (sign bit) is preserved, meaning the sign of the number is maintained.	The leftmost bit is filled with zero, meaning the sign is not preserved.
Use Case	Used when maintaining the sign of the number is important (e.g., dealing with signed integers).	Used when the logical value of bits is more important than the sign (e.g., manipulating bits as an unsigned binary number).
Example with Positive Number	`8 >> 2` results in `2` (`0000 1000` >> 2 = `0000 0010`)	`8 >>> 2` results in `2` (`0000 1000` >>> 2 = `0000 0010`)
Example with Negative Number	`-8 >> 2` results in `-2` (`1111 1000` >> 2 = `1111 1110`)	`-8 >>> 2` results in a large positive number (`1111 1000` >>> 2 = `0011 1110`)

- **Signed Right Shift (>>)**: Maintains the sign of the number. If the original number is negative, the new bits are filled with ones to keep the number negative.
- **Unsigned Right Shift (>>>)**: Does not maintain the sign of the number. The new bits are always filled with zeros, effectively treating the number as an unsigned value.

18.What's the difference between the methods sleep() and wait()?

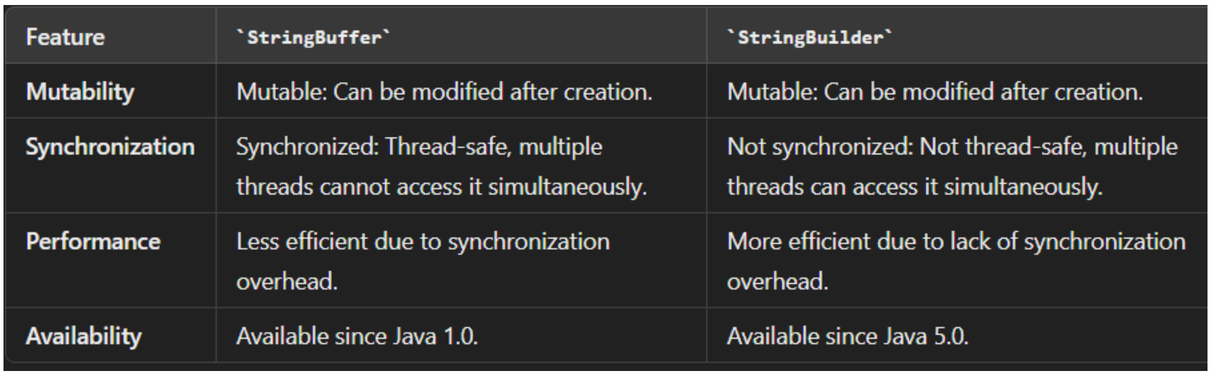
Sleep()	Wait()
Sleep belongs to Thread Class.	Wait belongs to Object Class.
Pauses the execution of the current thread for a specified time.	Makes the current thread to wait until other thread invoked notify() or notifyAll() using the same instance.
It takes time parameter as milliseconds.	It does not take any arguments.
Automatically resumes after the specified time ends.	Resumes after notify() or notifyAll() is called.
Temporarily moves the thread to TIMED_WAITING State	Moves the thread to WAITING or TIMED_WAITING state.
Does not require synchronization.	Requires Synchronization context.

19.What are the differences between String and StringBuffer?



Feature	`String`	`StringBuffer`
Mutability	Immutable: Once created, cannot be changed.	Mutable: Can be modified after creation.
Performance	Less efficient for concatenation and modification, as new objects are created for every modification operation.	More efficient for concatenation and modification, as it modifies the existing object instead of creating new ones.
Synchronization	Not synchronized.	Synchronized: Thread-safe, multiple threads can't access it simultaneously.
Memory Usage	Creates a new object every time a modification is made, leading to higher memory consumption and more garbage collection.	Modifies the existing object, reducing memory consumption and garbage collection.
Usage	Used when the content of the string will not change frequently, such as string literals and constant values.	Used when the content of the string needs to be modified frequently, such as in loops or when building dynamic strings.

20.What are the differences between StringBuffer and StringBuilder?



Feature	`StringBuffer`	`StringBuilder`
Mutability	Mutable: Can be modified after creation.	Mutable: Can be modified after creation.
Synchronization	Synchronized: Thread-safe, multiple threads cannot access it simultaneously.	Not synchronized: Not thread-safe, multiple threads can access it simultaneously.
Performance	Less efficient due to synchronization overhead.	More efficient due to lack of synchronization overhead.
Availability	Available since Java 1.0.	Available since Java 5.0.

21.On which memory arrays are created in Java?

- Heap memory is the part of memory where objects are stored in Java. When you create an array in Java using the **new** keyword, memory is allocated from the heap to hold the elements of the array.
- The heap memory is managed by the JVM's garbage collector, which automatically deallocates memory for objects that are no longer in use.

22.Why does the Java array index start with 0?

The index of an array signifies the distance from the start of the array. So, the first element has 0 distance therefore the starting index is 0.

Syntax : `[Base Address + (index * no_of_bytes)]`

23.What are the advantages and disadvantages of an array?

Advantages :

- Elements in an array can be accessed directly using their index. This allows for fast and efficient retrieval of elements, especially when the index is known.
- Arrays allocate contiguous memory locations to store elements. This leads to efficient memory usage.
- Iterating over elements in an array is efficient, as it can be done using simple loops like **for** or **while** loops.
- Arrays provide compile-time type safety, ensuring that only elements of the declared type can be stored in the array.

Disadvantages :

- Arrays have a fixed size, meaning the number of elements they can hold is predetermined at the time of creation. This makes it challenging to resize arrays dynamically.
- Inserting or deleting elements in the middle of an array or at the beginning requires shifting elements, which can be time-consuming, especially for large arrays.
- Arrays allocate memory for the maximum number of elements they can hold, even if the array is not fully populated. This can lead to wasted memory space, especially if the actual number of elements is small compared to the allocated size.

24. what is a constructor and how many types are there?

A constructor in Java is a special type of method that is called automatically when an object of a class is instantiated. It is used to initialize the newly created object and allocate memory for its instance variables. Constructors have the same name as the class and do not have a return type, not even **void**.

Constructors are commonly used to set initial values for the object's attributes or perform any necessary setup tasks

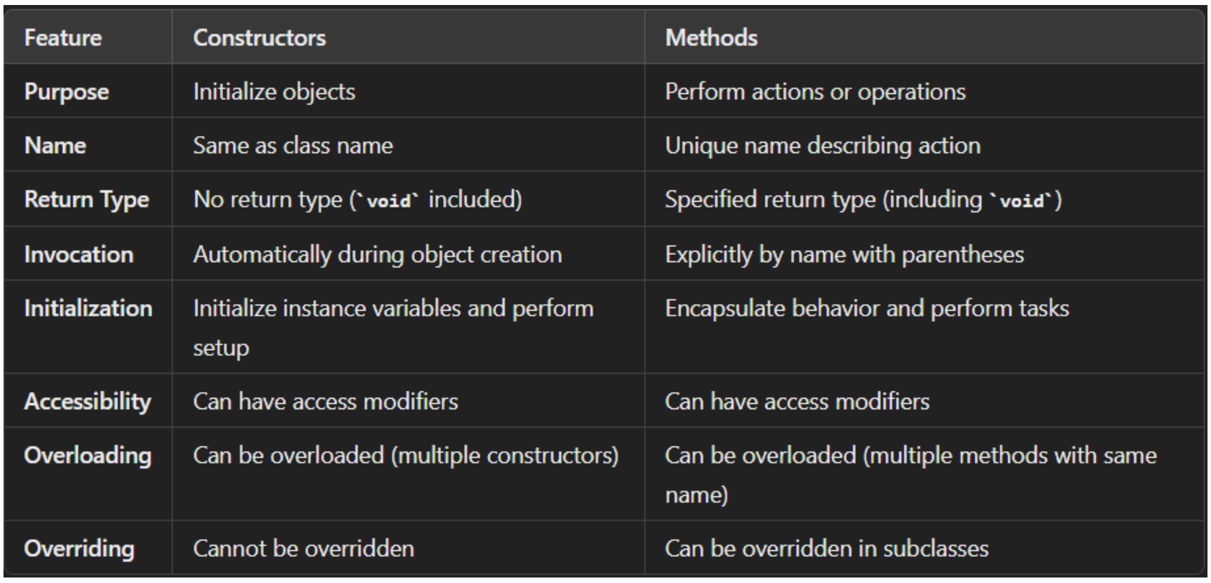
Types of Constructors:

1. **Default Constructor**:
    - A default constructor is automatically provided by the compiler if no other constructors are explicitly defined in the class. It has no parameters and initializes instance variables to their default values (e.g., 0 for numeric types, **null** for reference types).

2.Parameterized Constructor:

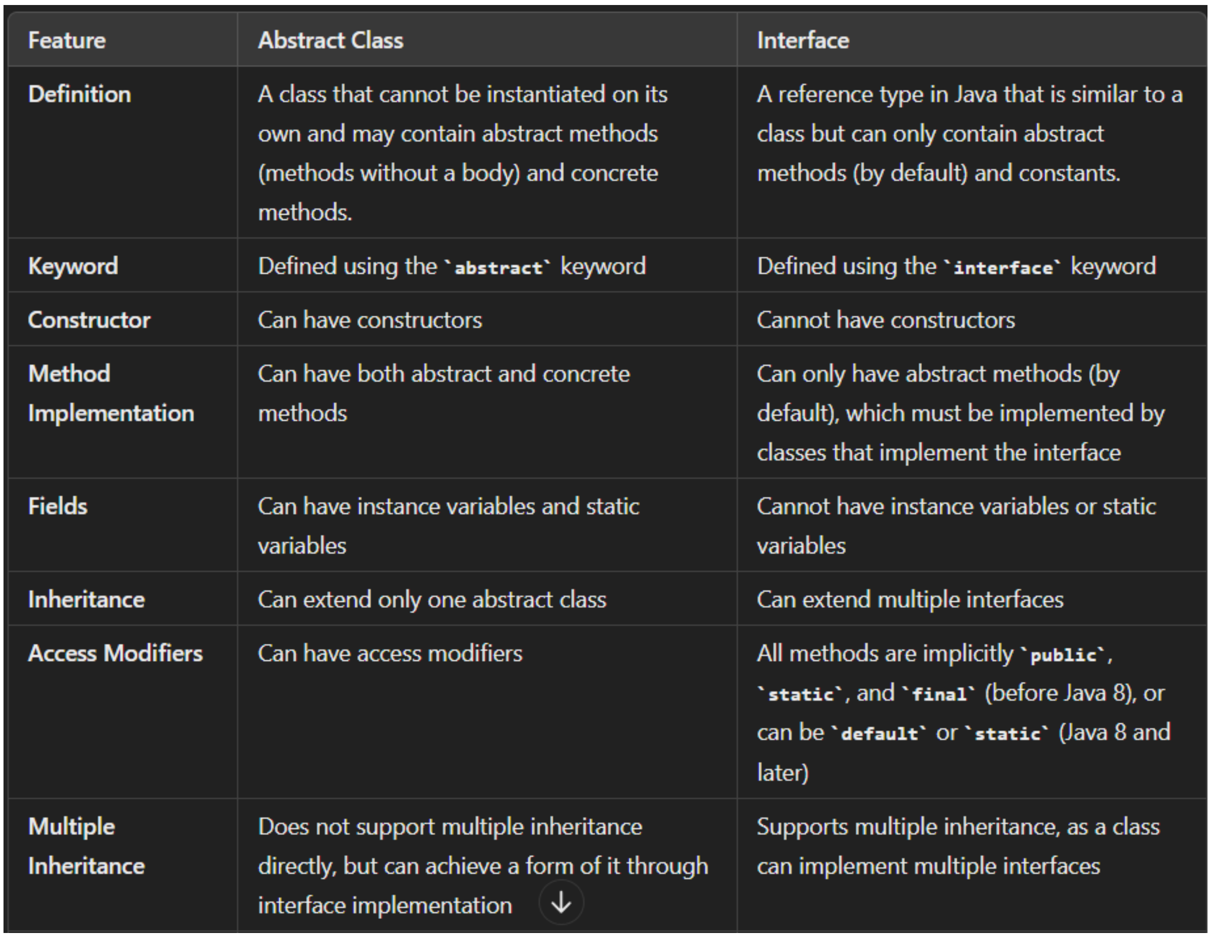
- A parameterized constructor accepts one or more parameters, allowing the caller to specify initial values for the object's attributes during object creation.

25.difference between constructors and methods?



Feature	Constructors	Methods
Purpose	Initialize objects	Perform actions or operations
Name	Same as class name	Unique name describing action
Return Type	No return type (`void` included)	Specified return type (including `void`)
Invocation	Automatically during object creation	Explicitly by name with parentheses
Initialization	Initialize instance variables and perform setup	Encapsulate behavior and perform tasks
Accessibility	Can have access modifiers	Can have access modifiers
Overloading	Can be overloaded (multiple constructors)	Can be overloaded (multiple methods with same name)
Overriding	Cannot be overridden	Can be overridden in subclasses

26.difference between abstract class and interface?



Feature	Abstract Class	Interface
Definition	A class that cannot be instantiated on its own and may contain abstract methods (methods without a body) and concrete methods.	A reference type in Java that is similar to a class but can only contain abstract methods (by default) and constants.
Keyword	Defined using the `abstract` keyword	Defined using the `interface` keyword
Constructor	Can have constructors	Cannot have constructors
Method Implementation	Can have both abstract and concrete methods	Can only have abstract methods (by default), which must be implemented by classes that implement the interface
Fields	Can have instance variables and static variables	Cannot have instance variables or static variables
Inheritance	Can extend only one abstract class	Can extend multiple interfaces
Access Modifiers	Can have access modifiers	All methods are implicitly `public`, `static`, and `final` (before Java 8), or can be `default` or `static` (Java 8 and later)
Multiple Inheritance	Does not support multiple inheritance directly, but can achieve a form of it through interface implementation	Supports multiple inheritance, as a class can implement multiple interfaces

27. What is the primary advantage of data encapsulation?

The primary advantage of data encapsulation is **enhanced security and integrity of the data**. By encapsulating data within classes and providing access only through well-defined methods.

- It restricts direct access to some of an object's components, which means that internal object details can be hidden from the outside world. This ensures that data cannot be altered unintentionally or accessed by unauthorized parts of the program.
- Encapsulation allows the developer to control how the data is accessed and modified. By using getter and setter methods.
- By encapsulating data, software systems achieve a higher level of abstraction, reduce complexity, and improve robustness and flexibility.

28. What is multiple inheritance? Is it supported by Java?

Multiple inheritance is a feature in some object-oriented programming languages that allows a class to inherit characteristics and behaviors (methods and fields) from more than one parent class. This means a subclass can have multiple super classes and can inherit attributes and methods from all of them.

Java does not support multiple inheritance of classes directly. This decision was made to avoid the complexity and ambiguity that can arise with multiple inheritance, such as the "diamond problem," where a class inherits from two classes that both inherit from a common superclass.

However, Java does support multiple inheritance of interfaces. A class in Java can implement multiple interfaces, allowing it to inherit the abstract methods from all of them. This provides some of the benefits of multiple inheritance without the associated complications.

```
interface InterfaceA {
   void methodA();
}
interface InterfaceB {
   void methodB();
}
class MultipleInheritanceExample implements InterfaceA, InterfaceB {
   public void methodA() {
      System.out.println("Method A from Interface A");
   }
   public void methodB() {
      System.out.println("Method B from Interface B");
   }
   public static void main(String[] args) {
      MultipleInheritanceExample example = new
MultipleInheritanceExample();
      example.methodA();
      example.methodB();
   }
}
```

29.Explain about composition in java?

The Composition is a way to design or implement the **"has-a"** relationship. Composition and Inheritance both are design techniques. The Inheritance is used to implement the **"is-a"** relationship. The **"has-a"** relationship is used to ensure the code reusability in our program. In Composition, we use an instance variable that refers to another object.

The composition relationship of two objects is possible when one object contains another object, and that object is fully dependent on it. The contained object should not exist without the existence of its parent object. In a simple way, we can say it is a technique through which we can describe the reference between two or more classes. And for that, we use the instance variable, which should be created before it is used.

```
import java.util.*;
class College
{
   String name;
   String address;
   College(String name, String address)
   {
      this.name = name;
      this.address = address;
   }
}
class University
{
   private final List<College> colleges;
   University(List<College> colleges)
   {
        this.colleges = colleges;
    }
    public List<College> getTotalColleges()
    {
        return(colleges);
    }
}
public class Main
{
        public static void main(String[] args) {
                List<College> college = new ArrayList<>();
                College c1 = new College("MLRIT","Gandimaisamma");
                College c2 = new College("MLRIP","Dundigal");
                College c3 = new College("IARE","ORR Road");
                College c4 = new College("CBIT","Hyderabad");
                college.add(c1);
                college.add(c2);
                college.add(c3);
                college.add(c4);
                University u = new University(college);
                List<College> colleges = u.getTotalColleges();
                for(College cg : colleges)
                {
                    System.out.println("Name     :     "+cg.name+"     Address     :
"+cg.address);
                }


        }
```

}

30. Explain method overloading and overriding?

Method Overloading

Definition: Method overloading is a feature that allows a class to have more than one method with the same name, but different parameter lists (different type, number, or both).

Purpose: The primary purpose of method overloading is to enhance the readability of the program by allowing methods to perform similar but distinct tasks with different inputs.

Key Points:

1. **Parameters**: Methods must have different parameter lists. This means the number of parameters or their types must differ.
2. **Return Type**: Methods can have different return types, but the return type alone is not enough to distinguish overloaded methods.
3. **Access Modifiers**: Methods can have different access modifiers.
4. **Exception Handling**: Methods can throw different exceptions.
5. **Polymorphism**: Method overloading is an example of compile-time polymorphism (static binding).

Method Overriding

Definition: Method overriding occurs when a subclass provides a specific implementation for a method that is already defined in its superclass. The method in the subclass must have the same name, return type, and parameters as the method in the superclass.

Purpose: The primary purpose of method overriding is to provide a specific implementation of a method that is defined in a superclass. This allows a subclass to define behavior that is specific to its type.

Key Points:

1. **Inheritance**: Method overriding requires inheritance. The subclass inherits from the superclass.

2. **Method Signature**: The method in the subclass must have the same name, return type (or a covariant return type), and parameter list as the method in the superclass.
3. **Access Modifiers**: The overridden method cannot have a more restrictive access modifier than the method in the superclass. It can have the same or a less restrictive access modifier.
4. **Exception Handling**: The overridden method can throw the same, fewer, or no exceptions compared to the method in the superclass.
5. **Polymorphism**: Method overriding is an example of runtime polymorphism (dynamic binding).

31.Can we overload the main() method?

Yes, you can overload the **main()** method in Java. However, the JVM (Java Virtual Machine) only looks for the specific signature of the main method used as the entry point for the application, which is:

public static void main(String[] args)

This is the signature that the JVM recognizes and calls to start the program. You can define other **main** methods with different signatures, but they won't be used as the entry point of the program. Instead, they can be called like any other method.

32. what is enumeration in java?

An enumeration, commonly known as an enum, is a special data type in Java that represents a group of constants. Enums are used to define a collection of named constants that can be treated as data types.

Key Features of Enums in Java

1. **Type Safety**: Enums provide type safety by ensuring that the values assigned to variables are within the defined set of constants.
2. **Readability**: Enums make the code more readable by using meaningful names for constants instead of arbitrary values.
3. **Predefined Methods**: Enums come with several predefined methods such as **values()**, **name()**, **ordinal()**, and **valueOf()**.

4. **Enhanced for Loop**: Enums can be used in enhanced for loops for iteration.
5. **Methods and Fields**: Enums can have fields, methods, and constructors, making them more powerful than simple constant groups.

33.Difference between array and arraylist?

Array	ArrayList
Fixed size. Must be specified at the time of creation.	Dynamic size. Can grow or shrink as needed.
Can hold both primitive types and objects.	Can only hold objects
Faster for fixed-size operations.	Slightly slower due to dynamic resizing operations.
Can store primitive types directly (e.g., int, char).	Requires use of wrapper classes for primitive types (e.g., Integer, Character).
Can be initialized with specific values at creation.	Must be instantiated and populated explicitly.

34. Difference between ArrayList and LinkedList.

ArrayList	**LinkedList**
Provides fast random access to elements with O(1) time complexity.	Access time is O(n) because traversal from the beginning or end is required.
Adding elements at the end is O(1) (amortized), but adding or removing elements in the middle is O(n) due to the need to shift elements.	Adding or removing elements at the beginning or end is O(1), but adding or removing elements in the middle is O(n) because of traversal time.
Less memory overhead as it uses a continuous block of memory.	More memory overhead due to storing two references (next and previous) per node.
Faster search for elements with O(n) time complexity, but can use binary search on sorted lists for O(log n).	Search time is O(n) and cannot use binary search.
Faster iteration performance because of contiguous memory storage.	Slower iteration performance due to non-contiguous memory storage.

35. what is exception handling and How many types of exceptions can occur in a Java program?

Exception handling in Java refers to the process of dealing with unexpected events or errors that occur during the execution of a program. These events can disrupt the normal flow of a program and cause it to terminate abruptly if not handled properly. Exception handling allows developers to anticipate such events and take appropriate actions to gracefully recover from them, ensuring that the program continues to run smoothly.

There are two main types of exceptions in Java:

1. Checked Exceptions: These are exceptions that are checked at compile-time. They are typically used to handle anticipated exceptional conditions that may occur during the execution of a program, such as file not found or network connection failure. Checked exceptions must be either caught using a try-catch block or declared in the method signature using the throws keyword.
2. Unchecked Exceptions: Also known as runtime exceptions, unchecked exceptions are not checked at compile-time. They typically represent programming errors or unexpected conditions that arise during the execution of a program, such as null pointer dereference or array index out of bounds. Unchecked exceptions do not need to be caught or declared, but they can still be caught and handled if desired.

Ex :

Checked Exceptions:

1. IOException
2. FileNotFoundException
3. ParseException

Unchecked Exceptions:

1. NullPointerException
2. ArrayIndexOutOfBoundsException
3. IllegalArgumentException

36. Difference between an Error and an Exception.

1. **Exception:**
    - Exceptions represent exceptional conditions that a program should anticipate and recover from. They are typically caused by the program's logic or external factors such as user input or I/O operations.
    - Exceptions in Java are further divided into two categories: checked exceptions and unchecked exceptions.
    - Checked exceptions are anticipated at compile-time and must be either caught using a try-catch block or declared in the method signature using the throws keyword.
    - Unchecked exceptions, also known as runtime exceptions, are not required to be caught or declared. They usually represent programming errors or unexpected conditions and are not anticipated by the compiler.
2. **Error:**
    - Errors represent abnormal conditions that are not expected to be caught or handled by a Java program. They usually denote serious problems that can't be recovered from, such as system failures or resource exhaustion.
    - Examples of errors in Java include **OutOfMemoryError**, **StackOverflowError**, and **VirtualMachineError**.

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

1. **Checked Exceptions:**
    - Checked exceptions are exceptions that are checked by the compiler at compile-time. This means that the compiler ensures that the code either handles the exception using a try-catch block or declares the exception to be thrown using the **throws** keyword in the method signature.
    - Checked exceptions typically represent anticipated exceptional conditions that can occur during the execution of a program, such as file I/O errors or network connection failures.

- Examples of checked exceptions include **IOException**, **FileNotFoundException**, and **ParseException**.
- The main purpose of checked exceptions is to enforce error handling and recovery mechanisms, ensuring that programs handle potential exceptional conditions gracefully.

2. **Unchecked Exceptions:**

   - Unchecked exceptions, also known as runtime exceptions, are not checked by the compiler at compile-time. This means that the compiler does not enforce the handling of these exceptions using try-catch blocks or the **throws** keyword.
   - Unchecked exceptions usually represent programming errors or unexpected conditions that occur during the execution of a program, such as null pointer dereference or array index out of bounds.
   - Examples of unchecked exceptions include **NullPointerException**, **ArrayIndexOutOfBoundsException**, and **IllegalArgumentException**.
   - Unchecked exceptions do not need to be explicitly caught or declared, but they can still be caught and handled if desired by the programmer.

38. Is it necessary that each try block must be followed by a catch block?

In Java, it is not necessary that each try block must be followed by a catch block. However, a try block must be followed by either a catch block or a finally block, or both.

Three cases :

1.Try and multiple catches

2.Try and finally

3.Try, catch and finally.

39. What purpose do the keywords final, finally, and finalize fulfill?

final:

- The **final** keyword is used to declare constants, methods, or classes that cannot be overridden, modified, or extended, respectively.
- When applied to a variable, **final** indicates that its value cannot be changed once initialized.
- When applied to a method, **final** indicates that the method cannot be overridden by subclasses.
- When applied to a class, **final** indicates that the class cannot be subclassed.

finally:

- The **finally** block is used in exception handling to execute code that should always be run, regardless of whether an exception occurred in the try block or not.
- The **finally** block is typically used for cleanup tasks, such as closing resources like files or database connections, that need to be executed regardless of whether an exception occurred.

finalize:

- The **finalize** method is a method provided by the **Object** class that gets called by the garbage collector before an object is garbage collected.
- It's used to perform cleanup or release resources associated with an object before it is destroyed.

40. What is the difference between this() and super() in Java?

this():

- **this()** is used to call another constructor within the same class.
- It is typically used to avoid code duplication when multiple constructors are present in a class, allowing one constructor to invoke another constructor with a different set of parameters.
- The call to **this()** must be the first statement in a constructor and can only be used to invoke another constructor in the same class.

super():

- **super()** is used to call a constructor of the superclass (i.e., the parent class).
- It is typically used when a subclass wants to explicitly call a constructor of its superclass.
- The call to **super()** must be the first statement in a constructor and can only be used to invoke a constructor of the superclass.
- If the superclass has multiple constructors, the appropriate **super()** call can be made based on the desired constructor of the superclass.

41.difference between throw and throws keyword in java?

1. **throw:**
    - The **throw** keyword is used to explicitly throw an exception from a method or a block of code.
    - It is used when a method encounters an exceptional situation that it cannot handle itself, and it wants to pass that exception to its caller for handling.
    - The **throw** statement must be followed by an instance of an exception class or subclass.
    - Here's an example:

```
public void doSomething()
{ if (condition)
{ throw new SomeException("Exception message");
}
}
```

2.throws:

- The **throws** keyword is used in method declarations to indicate that the method may throw certain types of exceptions during its execution.

- It specifies the types of exceptions that the method may throw, allowing the calling code to handle those exceptions or propagate them further.
- Multiple exceptions can be declared using a comma-separated list.
- The **throws** keyword does not throw exceptions itself; it simply declares the exceptions that a method might throw.
- Here's an example:

```
public void someMethod() throws IOException, SQLException
{ // Method implementation that may throw IOException or SQLException }
```

41.Why java is not completely OOP language?

Java is often considered a predominantly object-oriented programming (OOP) language, but it's true that it's not purely object-oriented. Here are a few reasons why:

1. **Primitive Data Types:** Java includes primitive data types like **int**, **float**, **boolean**, etc., which are not objects. In a purely object-oriented language, everything would be an object, including these basic data types.
2. **Static Methods and Variables:** Java allows the declaration of static methods and variables, which belong to the class rather than to individual objects. In a purely object-oriented language, all behavior would be encapsulated within objects.
3. **Procedural Constructs:** Java supports procedural programming constructs like loops (**for**, **while**) and conditional statements (**if**, **else**). While these can be used within object-oriented code, they are not strictly part of the object-oriented paradigm.
4. **Garbage Collection**: Java has a garbage collection mechanism that manages memory automatically. While this helps in managing memory efficiently, it abstracts away some of the memory management responsibilities that would otherwise be handled by the programmer in a purely object-oriented language.

42.what is meant by multi threading?

Multithreading in Java is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation, etc.

Advantages :

- Multithreading allows for parallel execution of tasks, which can significantly enhance the performance of an application, particularly on multi-core processors.
- Threads within the same process share the same memory space and resources, leading to efficient communication and data sharing.
- Many real-world problems can be naturally decomposed into parallel tasks, making multithreading a suitable approach for simulation, real-time processing, and other complex computations.

Disadvantages :

- Writing, debugging, and maintaining multithreaded applications can be significantly more complex than single-threaded applications.
- When multiple threads access shared resources simultaneously, it can lead to race conditions, where the outcome depends on the sequence or timing of the threads' execution.
- Improper handling of locks and synchronization can lead to deadlocks, where two or more threads are waiting indefinitely for each other to release resources.

43.What are the two ways in which the threads can be created in java?

1.Extending the thread class

In this approach, a new class is created that extends the Thread class. The run method is overridden to define the code that should be executed by the thread. An instance of this new class is created and the start method is called to begin execution.

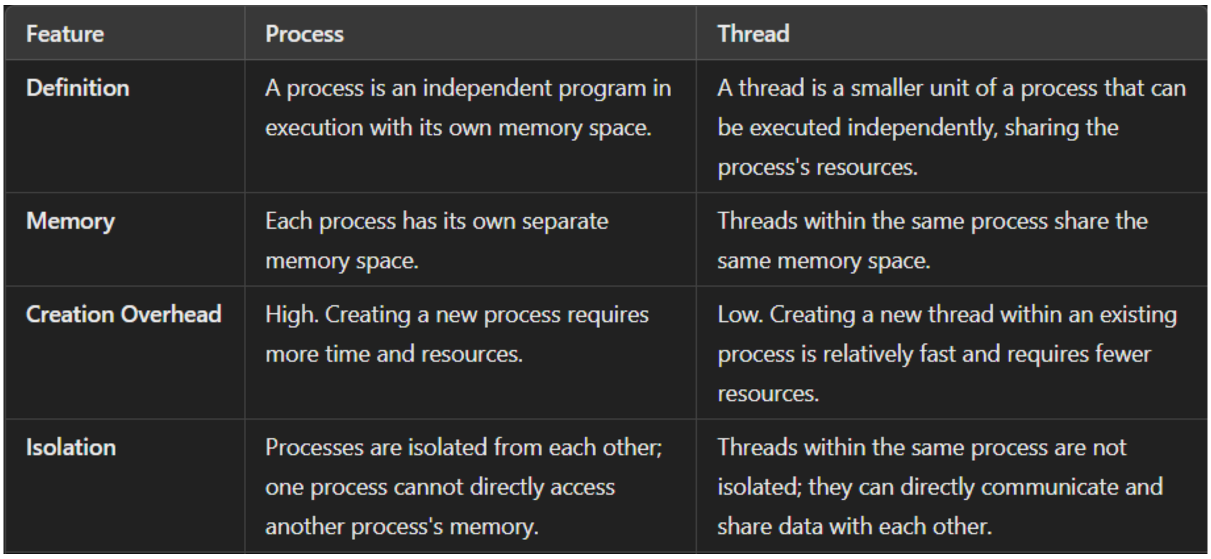
```
class MyThread extends Thread {
   public void run() {
        System.out.println("Thread is running.");
    }
    public static void main(String[] args) {
        MyThread thread = new MyThread();
        thread.start();
    }
}
```

2.Implementing the Runnable Interface

In this approach, a class implements the Runnable interface (in Java). The run method is defined in this class. An instance of the Runnable class is then passed to a Thread object, and the start method is called on the Thread object to begin execution.

```
class MyRunnable implements Runnable {
    public void run() {
        System.out.println("Thread is running.");
    }
    public static void main(String[] args) {
        MyRunnable runnable = new MyRunnable();
        Thread thread = new Thread(runnable);
        thread.start();
    }
}
```

44. Differentiate between process and thread?



Feature	Process	Thread
Definition	A process is an independent program in execution with its own memory space.	A thread is a smaller unit of a process that can be executed independently, sharing the process's resources.
Memory	Each process has its own separate memory space.	Threads within the same process share the same memory space.
Creation Overhead	High. Creating a new process requires more time and resources.	Low. Creating a new thread within an existing process is relatively fast and requires fewer resources.
Isolation	Processes are isolated from each other; one process cannot directly access another process's memory.	Threads within the same process are not isolated; they can directly communicate and share data with each other.

45. Describe the life cycle of the thread?

- **New (Created)**:
  - A thread is in this state when it is created but has not yet started. It remains in this state until the start() method is called.
- **Runnable (Ready to Run)**:
  - After calling the start() method, the thread moves to the runnable state. In this state, the thread is ready to run and is waiting for the CPU to allocate execution time.
- **Running**:
  - When the thread scheduler selects the thread, it moves from the runnable state to the running state, and the run() method is executed.
- **Blocked/Waiting**:
  - A thread enters this state when it is waiting for a specific condition to be met or a resource to become available. There are a few sub-states under the blocked/waiting state:
    - **Blocked on I/O**: Waiting for an I/O operation to complete.
    - **Blocked on Synchronization**: Waiting to acquire a lock or monitor.
    - **Waiting**: Waiting indefinitely for another thread to perform a specific action (e.g., wait() method in Java).

    - **Timed Waiting**: Waiting for a specified amount of time (e.g., sleep() method in Java, or wait(long timeout)).

 Terminated (Dead):

- A thread reaches this state when it has completed its execution (the run() method has finished) or has been explicitly terminated. Once a thread is in this state, it cannot be restarted.

Explanation of State Transitions

- **New to Runnable**: When the start() method is called on a thread, it transitions from the new state to the runnable state.
- **Runnable to Running**: The thread scheduler picks a thread from the runnable pool and transitions it to the running state.
- **Running to Blocked/Waiting**: A running thread can enter the blocked/waiting state if it waits for a resource, calls the wait() method, or sleeps.
- **Blocked/Waiting to Runnable**: A thread in the blocked/waiting state transitions back to the runnable state when the condition it was waiting for is met or the specified waiting time elapses.
- **Running to Terminated**: A thread transitions to the terminated state when its run() method completes or it is explicitly terminated.

46. What are the different types of Thread Priorities in Java?

In Java, thread priorities determine the relative importance of threads for the CPU scheduling. The Java runtime system's thread scheduler uses these priorities to decide when each thread should run. Thread priorities are integers that range between Thread.MIN_PRIORITY (1) and Thread.MAX_PRIORITY (10). The default priority for a thread is Thread.NORM_PRIORITY (5).

Types of Thread Priorities

1. **MIN_PRIORITY (1)**:

    - This is the lowest priority a thread can have. Threads with this priority are executed less frequently compared to higher-priority threads.
2. **NORM_PRIORITY (5)**:
    - This is the default priority assigned to a thread if no priority is explicitly set. It represents an average priority level.
3. **MAX_PRIORITY (10)**:
    - This is the highest priority a thread can have. Threads with this priority are executed more frequently compared to lower-priority threads.

47.what is meant by daemon thread in java?

- It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.
- Its life depends on user threads.
- It is a low priority thread.

The sole purpose of the daemon thread is that it provides services to user thread for background supporting task. If there is no user thread, why should JVM keep running this thread. That is why JVM terminates the daemon thread if there is no user thread.

48. what is garbage collection and how can it be useful in java?

Garbage collection (GC) is an automatic memory management feature in Java that helps in identifying and disposing of objects that are no longer in use by the application. The primary purpose of garbage collection is to reclaim memory occupied by these unused objects, thus preventing memory leaks and optimizing the use of available memory.

Garbage Collection Algorithms:

- **Mark-and-Sweep**: The garbage collector marks all reachable objects and then sweeps through memory to collect unmarked objects.
- **Copying**: The garbage collector copies live objects from one space to another, compacting memory and leaving behind garbage.

- **Mark-and-Compact**: Similar to mark-and-sweep but compacts live objects to one end of the memory, reducing fragmentation.
- **Generational**: Divides objects by age and applies different collection strategies based on their age.

49. what is meant by JDBC and explain the steps involved to connect database.

JDBC (Java Database Connectivity) is an API (Application Programming Interface) in Java that allows Java applications to interact with a wide range of databases. It provides methods for querying and updating data in a database, and it can be used with different types of relational databases, such as MySQL, Oracle, PostgreSQL, SQL Server, and others. JDBC abstracts the database-specific details, allowing developers to write database-independent code.

Steps :

- Before you can connect to a database, you need to load the appropriate JDBC driver. This driver is a library (usually a JAR file) that implements the JDBC API for a specific database.
- Use the DriverManager.getConnection() method to establish a connection to the database. You need to provide the database URL, username, and password.
- Create a Statement object to send SQL queries to the database.
- Use the Statement object to execute SQL queries. You can execute a SELECT query to retrieve data, or INSERT, UPDATE, or DELETE queries to modify data.
- If you executed a SELECT query, process the ResultSet object to retrieve the results.

```
import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

public class JDBCExample {
    // JDBC URL, username, and password of MySQL server
    private static final String URL = "jdbc:mysql://localhost:3306/mydatabase";
    private static final String USER = "root";
    private static final String PASSWORD = "password";
    // JDBC variables for opening and managing connection
    private static Connection connection;
    private static Statement statement;
    private static ResultSet resultSet;
    public static void main(String[] args) {
        try {
            // 1. Load the JDBC driver (optional for JDBC 4.0 and later)
            Class.forName("com.mysql.cj.jdbc.Driver");
            // 2. Establish a connection to the database
            connection = DriverManager.getConnection(URL, USER,
PASSWORD);
            System.out.println("Connection established successfully.");

            // 3. Create a statement
            statement = connection.createStatement();
            // 4. Execute a query
            String query = "SELECT * FROM users";
            resultSet = statement.executeQuery(query);
            // 5. Process the result set
            while (resultSet.next()) {
                int id = resultSet.getInt("id");
                String name = resultSet.getString("name");
                String email = resultSet.getString("email");
                System.out.println("ID: " + id + ", Name: " + name + ", Email: " + email);
            }
        } catch (ClassNotFoundException e) {
            System.err.println("JDBC Driver not found.");
            e.printStackTrace();
        } catch (SQLException e) {
            System.err.println("SQL Exception.");
            e.printStackTrace();
        }
```

50.what is the difference between equals() and == operator in java?

== Operator

- **Purpose**: The == operator is used to compare the memory addresses of the objects, i.e., it checks if both references point to the same object in memory.
- **Usage**: It is mainly used for primitive data types and to check if two object references point to the same instance.

```
Integer a = new Integer(10);
Integer b = new Integer(10);
Integer c = a;

System.out.println(a == b); // false, because a and b are different objects
System.out.println(a == c); // true, because a and c refer to the same object
```

equals() Method

- **Purpose**: The equals() method is used to compare the contents (values) of two objects for logical equality.

- **Usage**: It is meant for checking logical equality between objects. By default, the equals() method in the Object class compares memory addresses (like ==). However, many classes (such as String, Integer, etc.) override this method to provide value-based equality.

```
Integer a = new Integer(10);

Integer b = new Integer(10);

System.out.println(a.equals(b)); // true, because a and b have the same value
```

51.what is the difference between == and === operator in java?

In Java, the == operator is used for comparison, but there is no === operator. The === operator is not part of the Java language syntax. The === operator is commonly found in other languages like JavaScript, where it is used to compare both value and type strictly.

== Operator in Java

- **Purpose**: The == operator is used to compare primitive values or to check if two object references point to the same memory location (i.e., if they refer to the same instance).
- **Usage**: It can be used with both primitive types and object references.

```
int a = 10;

int b = 10;

System.out.println(a == b); // true, compares primitive values

Integer x = new Integer(10);

Integer y = new Integer(10);

System.out.println(x == y); // false, compares object references

Integer z = x;

System.out.println(x == z); // true, same reference
```

Since Java does not have a === operator, it's helpful to understand how it works in other languages for context, but this comparison is not applicable in Java.

In JavaScript (for context):

- **== Operator**: Checks for equality after type coercion.

console.log(5 == '5'); // true, because '5' is coerced to 5 before comparison

=== Operator: Checks for equality without type coercion (strict equality)

console.log(5 === '5'); // false, because the types are different

 Use the == operator to compare primitive values or to check if two references point to the same object.

 There is no === operator in Java. The == operator in Java is equivalent to the === operator in languages like JavaScript when used with object references, as it checks both reference (memory location) and type (both operands must be references of the same type).

52. What is the output of the following Java program?

```
class Test
{
    public static void main (String args[])
    {
        System.out.println(10 + 20 + "Javatpoint");
        System.out.println("Javatpoint" + 10 + 20);
    }
}
```

```
30Javatpoint
Javatpoint1020
```

53.what is meant by implicit and explicit typecasting explain?

Implicit Type Casting (Automatic)

Implicit type casting happens automatically when:

1. The two types are compatible.
2. The target type is larger or more comprehensive than the source type (e.g., converting an int to a long).

This type of casting is also known as "widening conversion" because it goes from a smaller to a larger data type.

Converting from int to long:

int num = 100;

long longNum = num; // Implicit casting

Explicit Type Casting (Manual)

Explicit type casting requires manual intervention by the programmer to convert one data type into another. This is necessary when:

1. The target type is smaller than the source type (e.g., converting a double to an int).
2. The two types are not compatible and there is a potential for data loss.

This type of casting is also known as "narrowing conversion" because it goes from a larger to a smaller data type.

targetType variableName = (targetType) value;

Converting from long to int:

long longNum = 100L;

int num = (int) longNum; // Explicit casting

 Implicit Casting:

- No data loss (since it converts to a larger or more comprehensive type).
- Happens automatically.
- Example: int to long.

 Explicit Casting:

- May involve data loss (since it converts to a smaller or less comprehensive type).
- Requires manual intervention.
- Example: double to int.

54. What is the difference between an object-oriented programming language and object-based programming language?

- Object-oriented languages follow all the concepts of OOPs whereas, the object-based language doesn't follow all the concepts of OOPs like inheritance and polymorphism.
- Object-oriented languages do not have the inbuilt objects whereas Object-based languages have the inbuilt objects, for example, JavaScript has window object.
- Examples of object-oriented programming are Java, C#, Smalltalk, etc. whereas the examples of object-based languages are JavaScript, VBScript, etc.

55.what is the difference between object oriented programming and procedure oriented programming?

OOPS	POP
In procedural programming, the program is divided into small parts called ***functions***.	In object-oriented programming, the program is divided into small parts called ***objects***.
Procedural programming follows a ***top-down approach***.	Object-oriented programming follows a ***bottom-up approach***.
There is no access specifier in procedural programming.	Object-oriented programming has access specifiers like private, public, protected, etc.
Procedural programming does not have any proper way of hiding data so it is ***less secure***.	Object-oriented programming provides data hiding so it is ***more secure***.
Examples: C, FORTRAN, Pascal, Basic, etc.	**Examples:** C++, Java, Python, C#, etc.

56.Explain about static block and instance block in java?

When a block is decorated or associated with the word static, it is called a static block. Static Block is known as the static clause. A static block can be used for the static initialization of a class. The code that is written inside the static block run once, when the class is getting loaded into the memory.

1. **public class** StaticBlock

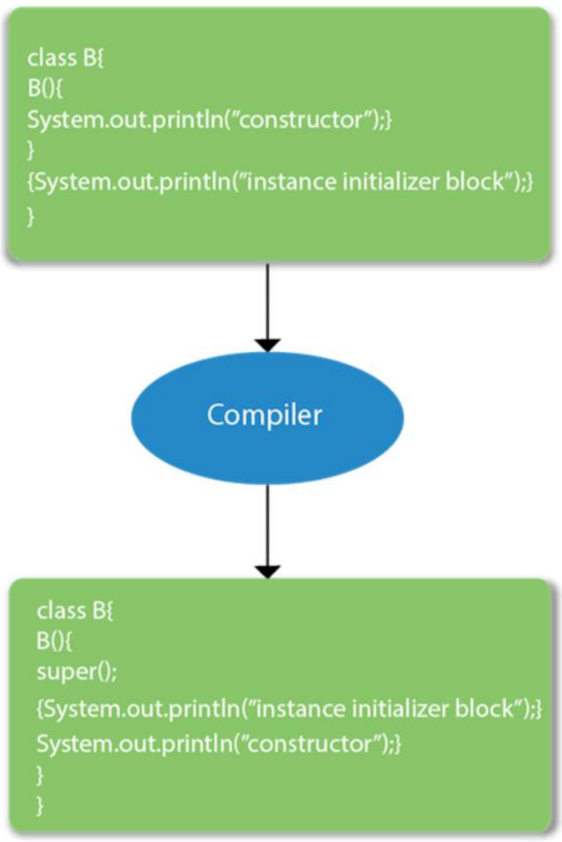
```
{

// Constructor of the class StaticBlock
StaticBlock()
{
System.out.println("Inside the constructor of the class.");
}

// print method of the StaticBlock class
public static void print()
{
System.out.println("Inside the print method.");
}

static
{
System.out.println("Inside the static block.");
}

// main method
public static void main(String[] args)
{

// instantiating the class StaticBlock
StaticBlock sbObj = new StaticBlock();
sbObj.print(); // invoking the print() method

// invoking the constructor inside the main() method
new StaticBlock();

}
}
```

Instance Initializer block

Instance Initializer block is used to initialize the instance data member. It run each time when object of the class is created.

The initialization of the instance variable can be done directly but there can be performed extra operations while initializing the instance variable in the instance initializer block.

```
class Bike7{
   int speed;
   Bike7(){System.out.println("speed is "+speed);}
   {speed=100;}
   public static void main(String args[]){
   Bike7 b1=new Bike7();
   Bike7 b2=new Bike7();
   }
}
```



```
class B{
B(){
System.out.println("constructor");}
}
{System.out.println("instance initializer block");}
}
```

```
class B{
B(){
super();
{System.out.println("instance initializer block");}
System.out.println("constructor");}
}
}
```

Internal processing happens when we use instance initializer block in java.

57.why java does not support pointers?

In languages like C and C++, pointers allow direct manipulation of memory addresses. This can lead to errors like buffer overflows, which can corrupt memory and cause unpredictable behavior. By eliminating pointers, Java prevents such risks, ensuring that memory cannot be arbitrarily accessed or modified.

Without pointers, Java controls how memory is accessed and manipulated. Java uses an automatic garbage collector to manage memory. Without pointers, the garbage collector can more safely and effectively reclaim memory without the risk of encountering dangling pointers or memory leaks.

58. Can we call the run() method instead of start()?

Yes, calling run() method directly is valid, but it will not work as a thread instead it will work as a normal object. There will not be context-switching between the threads. When we call the start() method, it internally calls the run() method, which creates a new stack for a thread while directly calling the run() will not create a new stack.

59. what is synchronization and why is it used in java?

Synchronization in Java

Synchronization in Java is a mechanism that ensures that only one thread can access a resource at a time. This is crucial in multithreaded applications where multiple threads might try to access and modify shared resources concurrently, leading to inconsistent states or race conditions.

Why Synchronization is Used in Java

1. **Thread Safety**: Synchronization ensures that shared resources are accessed by only one thread at a time, preventing data corruption and ensuring consistency.
2. **Preventing Race Conditions**: Without synchronization, multiple threads could modify shared data simultaneously, leading to unpredictable results. Synchronization ensures that only one thread modifies the data at a time.
3. **Atomicity**: It guarantees that a sequence of operations on a shared resource is executed atomically, meaning without interruption by other threads.
4. **Coordination Between Threads**: Synchronization allows threads to coordinate their actions. For example, it can be used to implement critical sections, where only one thread can execute a block of code at a time.

We can make a synchronized block, synchronized Method.

Example code :

```
class Counter {
  private int count = 0;

  public synchronized void increment() {
    count++;
  }
```

```
    public int getCount() {
        return count;
    }
}


public class Main {
    public static void main(String[] args) {
        Counter counter = new Counter();

        // Create multiple threads to increment the counter concurrently
        Thread t1 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) {
                counter.increment();
            }
        });

        Thread t2 = new Thread(() -> {
            for (int i = 0; i < 1000; i++) {
                counter.increment();
            }
        });
        t1.start();
        t2.start();
        System.out.println("Final count (without synchronization): " +
counter.getCount());
    }
}
```

60.What are the real time applications of different data structures?

- **Arrays**:
  - **Image Processing**: Arrays are used to represent the pixels in an image. Each pixel's color intensity values (RGB) can be stored in a multi-dimensional array.
  - **Signal Processing**: Arrays are utilized to store time-series data, such as audio signals or sensor readings from IoT devices.
- **Linked Lists**:
  - **Memory Management**: Linked lists are commonly used in memory allocation algorithms, such as malloc/free in C programming language, to manage dynamic memory allocation.
  - **Undo Functionality**: In text editors or design software, linked lists can be used to implement undo functionality by storing the sequence of previous states.
- **Stacks**:
  - **Expression Evaluation**: Stacks are used in compilers to evaluate arithmetic expressions, parse infix expressions into postfix notation, and perform calculations.
  - **Web Browser History**: Stacks can be employed to manage the history of web pages visited by users, enabling forward and backward navigation.
- **Queues**:
  - **Task Scheduling**: In real-time operating systems, queues are used for task scheduling, ensuring that tasks are executed in the order they are received.
  - **Print Spooling**: Queues are used in operating systems to manage print jobs in a spooler, ensuring they are printed in the order they were submitted.
- **Trees**:
  - **Database Indexing**: Binary search trees (BSTs) are used in databases for indexing, facilitating fast retrieval of records based on keys.
  - **Decision Trees**: In machine learning, decision trees are used for classification and regression tasks, making decisions based on feature values at internal nodes.
- **Graphs**:

- **Social Networks**: Graphs are used to model social networks, with users represented as nodes and connections as edges, enabling friend recommendations and network analysis.
- **Network Routing**: Graphs are used in networking protocols to find the shortest path between nodes, optimizing data packet routing.

 Hash Tables:

- **Symbol Tables**: Hash tables are used to implement symbol tables in compilers, storing variable names and their associated memory locations efficiently.
- **Caching Mechanisms**: Hash tables are employed in web servers and databases for caching frequently accessed data, improving system performance by reducing data retrieval time.

61.Real time examples of OOPs?

 Encapsulation:

- **Real-time Example**: Consider a thermostat system in a smart home. The internal workings, such as temperature sensors, algorithms for controlling heating/cooling, and user interface elements, are encapsulated within the thermostat object. External entities interact with the thermostat through a well-defined interface, such as setting temperature preferences or querying current room temperature. This encapsulation ensures that the complexity of the thermostat's internal logic is hidden from the user, promoting ease of use and maintenance.

 Inheritance:

- **Real-time Example**: In software for a retail business, there may be different types of employees, such as regular employees, supervisors, and managers. They all share common attributes like name, ID, and contact information, and they perform tasks like handling transactions and managing inventory. Instead of duplicating code for each type of employee, a base class Employee can be created with common attributes and methods. Subclasses like Manager and Cashier inherit from Employee and provide specialized functionality. This inheritance hierarchy promotes code reuse and ensures consistency across different employee types.

 Polymorphism:

- **Real-time Example**: In a video conferencing application, various devices (e.g., webcams, microphones, speakers) may be used for audio and video input/output. Each device implements a common interface for starting/stopping audio/video streams and adjusting volume or brightness. During a video call, the application can interact with these devices polymorphically, treating them uniformly regardless of their specific types. For example, the application can call a generic startStream() method on a device object without needing to know whether it's a webcam or a microphone.

 Abstraction:

- **Real-time Example**: Consider an online banking system. The system provides functionalities such as transferring funds, checking account balances, and updating personal information. Internally, the system interacts with various components, including databases, external APIs, and authentication services. However, from the perspective of a user interacting with the banking application, these implementation details are abstracted away. Users interact with the system through a user-friendly interface, where they can perform banking operations without needing to understand the underlying complexities of data storage or network communication.

62.difference between linear and non linear data structures?

 Linear Data Structures:

- Linear data structures organize data elements in a sequential manner, where each element is connected to its previous and next element, forming a linear sequence.
- Examples of linear data structures include arrays, linked lists, stacks, and queues.
- Accessing elements in linear data structures typically involves traversing through each element sequentially from the starting point to the desired location.
- Linear data structures are suitable for situations where data needs to be processed in a sequential order, such as processing a list of tasks or items in a queue.

 Non-linear Data Structures:

- Non-linear data structures do not organize data in a sequential order; instead, they allow data elements to be connected in a hierarchical or arbitrary manner.
- Examples of non-linear data structures include trees, graphs, and hash tables.
- Accessing elements in non-linear data structures may involve traversing through multiple paths or branches to reach the desired element.
- Non-linear data structures are suitable for representing relationships or connections between data elements that may not follow a strict linear order, such as representing organizational hierarchies or network connections.

63.what is the difference between max and min priority queue?

Max Priority Queue:

- In a max priority queue, elements with the highest priority are removed first.
- The element with the highest priority (maximum priority value) is at the front of the queue.
- For example, in a queue representing tasks with priorities, tasks with the highest priority (e.g., highest importance or earliest deadline) are processed first.

Min Priority Queue:

- In a min priority queue, elements with the lowest priority are removed first.
- The element with the lowest priority (minimum priority value) is at the front of the queue.
- For example, in a queue representing jobs with priorities, jobs with the lowest priority (e.g., least importance or latest deadline) are processed first.

64.what is a deadlock explain?

A deadlock is a situation in computer science, especially in concurrent programming and operating systems, where two or more processes or threads are unable to proceed because each is waiting for the other to release a resource. This creates a cycle of dependencies that cannot be broken, causing the involved processes or threads to remain stuck indefinitely.

Example of Deadlock

To better understand deadlock, let's consider a simple example with two threads and two resources:

1. **Thread A** locks Resource 1 and waits to lock Resource 2.
2. **Thread B** locks Resource 2 and waits to lock Resource 1.

In this scenario, Thread A cannot proceed until it gets Resource 2, and Thread B cannot proceed until it gets Resource 1. Both threads are waiting for each other to release the resources, creating a deadlock.

Necessary Conditions for Deadlock

Four conditions must be present simultaneously for a deadlock to occur:

1. **Mutual Exclusion**: At least one resource must be held in a non-shareable mode. Only one process can use the resource at any given time.
2. **Hold and Wait**: A process holding at least one resource is waiting to acquire additional resources held by other processes.
3. **No Preemption**: Resources cannot be forcibly taken away from a process holding them. They must be released voluntarily.
4. **Circular Wait**: A set of processes are waiting for each other in a circular chain. Each process holds at least one resource and is waiting for a resource held by the next process in the chain.

65.Explain paging in OS?

In Operating Systems, Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages.

The main idea behind the paging is to divide each process in the form of pages. The main memory will also be divided in the form of frames.

One page of the process is to be stored in one of the frames of the memory. The pages can be stored at the different locations of the memory but the priority is always to find the contiguous frames or holes.

Pages of the process are brought into the main memory only when they are required otherwise they reside in the secondary storage.

Different operating system defines different frame sizes. The sizes of each frame must be equal. Considering the fact that the pages are mapped to the frames in Paging, page size needs to be as same as frame size.

Example :

Let us consider the main memory size 16 Kb and Frame size is 1 KB therefore the main memory will be divided into the collection of 16 frames of 1 KB each.

There are 4 processes in the system that is P1, P2, P3 and P4 of 4 KB each. Each process is divided into pages of 1 KB each so that one page can be stored in one frame.

Initially, all the frames are empty therefore pages of the processes will get stored in the contiguous way.

Let us consider that, P2 and P4 are moved to waiting state after some time. Now, 8 frames become empty and therefore other pages can be loaded in that empty place. The process P5 of size 8 KB (8 pages) is waiting inside the ready queue.

Given the fact that, we have 8 non contiguous frames available in the memory and paging provides the flexibility of storing the process at the different places. Therefore, we can load the pages of process P5 in the place of P2 and P4.

66. Differences between Data Encapsulation and Data Abstraction

1.purpose :

Encapsulation: Protects and restricts access to the data by bundling it with methods. Focuses on how data is protected.

Abstraction: Hides the complexity of the system by exposing only the necessary parts. Focuses on what the object does rather than how it does it.

2.Visibility :

Encapsulation: Makes use of access modifiers (private, protected, public) to control access to the data.

Abstraction: Uses abstract classes and interfaces to define the necessary operations without revealing the implementation.

67.What is the difference between Hashmap and Hashtable?

Hashmap	Hashtable
Introduced in Java 1.2	Introduced in Java 1.0
Allows one null key and multiple null values.	Does not allow null key and values
It is not synchronized i.e multiple threads can access.	It is synchronized i.e only one thread can access at a time.
Initial capacity can hold 16 key values.	Initial capacity can hold 11 key values.
It is not thread safe.	It is thread safe.

68.what is meant by throwable in java?

1. In Java, Throwable is the superclass for all errors and exceptions that can be thrown by the Java Virtual Machine (JVM) or by user code.
2. It is a part of the java.lang package.
3. The Throwable class provides a common framework for error handling in Java and contains methods to get information about the error or exception that occurred.

Usage of throwable :

Error Handling: It is used to handle both recoverable conditions (exceptions) and non-recoverable conditions (errors).

Custom Exceptions: You can extend Throwable to create your own custom exceptions.

69.What is a marker interface in java?

In Java, a marker interface is an interface that does not contain any method declarations.

Examples of Marker Interfaces in Java

1. **Serializable**: Indicates that a class can be serialized (converted to a byte stream and then deserialized back into an object).
2. **Cloneable**: Indicates that a class allows for a field-for-field copy to be made (supports the clone() method).
3. **Remote**: Indicates that a class can be used for remote method invocation (RMI).

Example :

```
public interface MyMarkerInterface {
   // No methods
}
public class MyClass implements MyMarkerInterface {
   // Class implementation
}
```

In this example:

- MyMarkerInterface is a custom marker interface with no methods.
- MyClass implements MyMarkerInterface, indicating that it possesses the property signified by MyMarkerInterface.

70.what is meant by concrete class and concrete method?

A concrete class is a class that has complete implementations for all of its methods. Unlike abstract classes, which can have one or more abstract methods (methods without a body), a concrete class provides the code for all of its methods. Concrete classes can be instantiated to create objects.

A concrete method is a method that has a body, meaning it has a complete implementation. In contrast, an abstract method, which is declared in an abstract class or interface, does not have a body and must be implemented by subclasses or implementing classes.

71.how does hashmap works internally in java?

Internal Structure of HashMap

1. **Buckets**: A HashMap consists of an array of buckets. Each bucket is essentially a LinkedList (in Java 7) or a Node (in Java 8 and later) that holds Entry objects.
2. **Entries (Nodes)**: Each entry (or node) contains a key, a value, a hash code of the key, and a reference to the next entry (for handling collisions).

Key Components

- **Hash Function**: Used to compute an index (bucket location) based on the hash code of the key.

- **Load Factor**: Determines when the HashMap should resize (rehash). The default load factor is 0.75, meaning the HashMap will resize when it is 75% full.
- **Initial Capacity**: The default initial capacity of a HashMap is 16 buckets.

Putting Data

1. **Hashing the Key**: When you use the put() method to add a key-value pair, the HashMap uses the hash function on the key to get a hash code.
2. **Finding the Bucket**: The hash code is used to determine which bucket (drawer) to use. This is done using the formula:

index=hash%number of buckets\text{index} = \text{hash} \% \text{number of buckets}index=hash%number of buckets

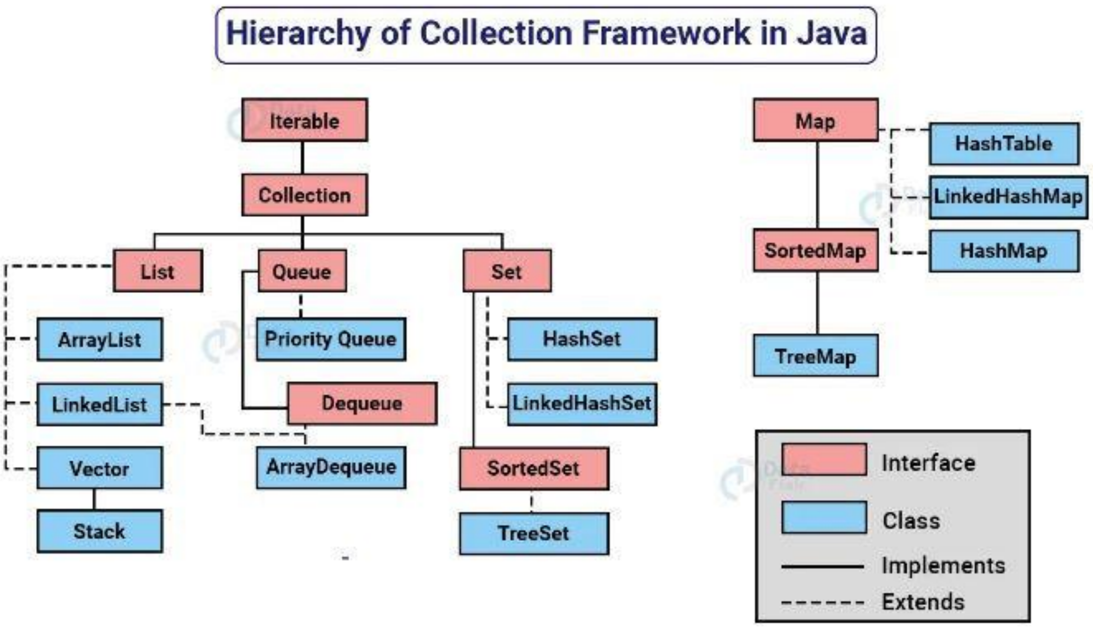
This ensures the hash code fits within the available buckets.

3. **Storing the Entry**: If the bucket is empty, the HashMap simply puts the key-value pair there. If not, it handles the collision by adding the new entry to the list of entries already in that bucket.

Getting Data

1. **Hashing the Key**: When you use the get() method to retrieve a value, the HashMap hashes the key again to find the hash code.
2. **Finding the Bucket**: It uses the hash code to find the right bucket.
3. **Searching the Bucket**: It searches through the entries in that bucket to find the one with the matching key and returns the corresponding value.

72.Explain the Java collections framework?



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