**Java Questions**

1. What is JVM?

* Ans : The JVM is an **abstract machine** that provides a runtime environment for executing **Java bytecode**. It acts as an intermediary between compiled Java code and the underlying hardware.
* Essentially, the JVM allows you to write your Java code once and run it on various platforms without modification.
* Its does not exists physically and can also run the programme which are written in the other languages.
* JVM (Java Virtual Machine):
* Execution Environment: JVM enables your computer to run Java programs.
* Bytecode Translation: It translates Java bytecode into native machine code.
* Platform Independence: Executes platform-independent Java code.

* JRE (Java Runtime Environment):
* Runtime Components: Provides Java class libraries, JVM, and necessary components.
* For Running Programs: Use JRE to run Java applications.
* Superset of JVM: JRE includes JVM.
* JDK (Java Development Kit):
* Development Kit: Required for Java application development.
* Includes JRE: JDK contains JRE along with development tools.
* Compilers, Debuggers, Docs: JDK provides compilers, JavaDoc, and other development utilities.
* Remember, JVM is the heart of Java, JRE is for running programs, and JDK is for developing Java applications!

Ways to create object in java?

* Using the new keyword – Classname obj = new Classname();
* Using reflection –
* Using object Deserialization
* Using factory method

Integer to string typecasting ?

* Using String.ValueOf(num);
* Using concatenation with empty set

1. What are the different types of variables?

Local variable , Instance variable (non static variable ) and static variable (class variable ).

**Local Variables**:

* A **local variable** is declared inside the body of a method or a block.
* Its scope is limited to the method or block where it is defined.
* Other methods in the same class are unaware of its existence.
* Local variables cannot be declared with the static keyword.
* public class Example {
* public void someMethod() {
* int localVar = 42; // Local variable
* }
* }
* **Instance Variables (Non-Static Variables):**
* An instance variable is declared within a class but outside any method.
* It is not marked as static.
* Each instance (object) of the class has its own copy of instance variables.
* Instance variables hold data specific to an object.
* public class Person {
* String name; // Instance variable
* int age;
* }

Static Variables (Class Variables):

A static variable is declared with the static keyword.

It is associated with the class rather than any specific instance.

Only one copy of a static variable exists, shared among all instances of the class.

Memory allocation for static variables occurs only once when the class is loaded.

public class MathUtils {

static final double PI = 3.14159; // Static variable

}

1. What are Non-primitive data types?

Java has two categories of data types:

Primitive Data Types: These are basic data types (e.g., int, double, char, boolean) that store simple values.

Non-Primitive Data Types (Reference Types): These include classes, interfaces, arrays, and other user-defined types.

1. What is the difference between an Integer and int in Java

**int:**

* int is a primitive data type.
* It directly stores a 32-bit signed two’s complement integer value.
* It is efficient in terms of memory usage.
* Only allows binary integer values.
* Limited flexibility.
* Requires 4 bytes to store an integer value.
* Cannot directly convert the integer value to another base.
* **Integer:**
* Integer is a wrapper class for the int data type.
* It provides additional functionality and flexibility.
* It behaves like an object and follows object-oriented principles.
* Conversion methods (e.g., toBinaryString(), toOctalString(), toHexString()).
* Allows working with objects and performing operations.
* Useful for scenarios like collections and generics.
* Takes 16 bytes due to its object overhead.
* Supports casting from strings using Integer.parseInt().

1. Can you tell the Different Data types in Java along with its size

byte: Stores whole numbers from -128 to 127. Size: **1 byte**.

short: Stores whole numbers from -32,768 to 32,767. Size**: 2 bytes.**

* boolean: Stores true or false values. Size: **1 bit.**
* char: Stores a single character or ASCII values. Size: **2 bytes**.
* int: Stores whole numbers from -2,147,483,648 to 2,147,483,647. Size: **4 bytes**.= 32bits
* long: Stores whole numbers in a larger range. Size: **8 bytes.**
* float: Stores fractional numbers. Sufficient for 6 to 7 decimal digits. Size: **4 bytes.**
* double: Stores fractional numbers with greater precision. Sufficient for 15 decimal digits. Size: **8 bytes** =64 bits.

**Non primitive**

* String , Array , Class , Interface .

1. What is the difference between float and double datatypes?

* float: Stores fractional numbers. Sufficient for 6 to 7 decimal digits. Size: 4 bytes.
* The float data type represents a 32-bit single-precision IEEE 754 floating-point number.
* Float has lower accuracy and declared as float height = 16.56f ;
* Use float for particle simulations, fluid dynamics, and simulations with large data sets.
* **Double**
* double: Stores fractional numbers with greater precision. Sufficient for 15 decimal digits. Size: 8 bytes.
* Accuracy of double is more than float and declared as double price = 16.568D;
* Use double for precise financial calculations, such as interest rates, currency conversions, or stock market predictions.

1. What is an Array?

* Array is a non-primitive data types used in java .
* It store same types of multiple element at continuous memory location . it uses index .
* Single-Dimensional Arrays: A simple list of elements.
* Multidimensional Arrays: Arrays of arrays (e.g., matrices).
* Example of 2D array ;, to store pixel of images , matrix
* We can use a 2D array to store temperature readings for each city (rows) across different days (columns).
* Each row represents a city, and each column represents a day of the week.
* int[][] temperature = {
* {33, 34, 35, 33, 32, 31, 30}, // City 1
* {23, 22, 21, 24, 22, 25, 26} // City 2
* };
* Int [] arr = new int [6]; , int arr [][]= new int [3][4];
* Int [] arr = {12, 16,17 };
* **Three-Dimensional Array (3D Array**

int[][][] chessboards = {

{ // Chessboard 1

{1, 0, 1, 0},

{0, 1, 0, 1},

{1, 0, 1, 0}

},

{ // Chessboard 2

{0, 1, 0, 1},

{1, 0, 1, 0},

{0, 1, 0, 1}

}

};

1. Is it possible to declare an Array without Array size?

No we can not declare arrya without specifying its size in java . Size of arry is fixed once it created . if you want dynamic changing size of data structure then go with Array list .

List<Integer> dynamicList = new ArrayList<>();

1. Can you write how to declare array?

Int arr [] = new int [];

Int arr [] = {12,16,18};

1. When do you use BREAK statement in java?

In Java, the break statement is used to exit a loop or switch statement prematurely. Breakes allow you to terminate the loop based on the certain conditions .

* Loop control :
* Switch case statement :

Int day =3

Switch(day){

case:1

System.out.println(“Its monaday”);

Break;

case 2:

* Nested loop: When working with nested loops, break can exit the inner loop while keeping the outer loop intact.
* for (int i = 0; i < 5; i++) {
* for (int j = 0; j < 3; j++) {
* if (j == 2) {
* break; // Exit the inner loop when j reaches 2
* }

1. Explain super keyword in Java?

* Super keyword is a reference variable in java which is used to refer the immediate parent class (super class ) of a child class .
* It used to access the parent class fields , methods .
* It is also used to invoke parent class constructor .
* Just write super () in child class and constructor will be called .

1. What is class?

* Collection of objects called class and instance of class is called object.
* Class in not a real-world entity and it does not occupy any spaces or memory, it’s an abstract concept.
* Class contains methods, constructors, interfaces and nested classes.
* Its name starts with capital letter.

1. What is the base class of all classes?

* **Object class** is base for all classes. java.lang.Object
* It is the starting point of the class hierarchy in Java.
* the Object class ensures consistency, enables polymorphism, and provides essential methods for all Java classes
* Every class implicitly inherits from Object, which provides essential methods like equals(), hashCode(), and toString().

1. What is the purpose of THIS keyword?

* The this keyword is a reference variable that refers to the current object instance within a method or constructor.
* Its primary purpose is to distinguish between instance variables (fields) and method parameters or local variables when they have the same name.
* this can be used to **refer current class instance variable**.
* this can be used to **invoke current class method (implicitly)**
* this() can be used **to invoke current class constructor.**
* this can be passed **as an argument in the method call.**
* this can be passed as argument in the constructor call.
* this can be used to return the current class instance from the method.

class Student {

private String name; // Instance variable

Student(String name) {

this.name = name; // Refers to the instance variable

}

}

**Invoking Current Class Methods:**

You can use this to call other methods within the same class.

Useful for chaining method calls or invoking overloaded constructors.

1. What is the use of the final keyword?

* The **final keyword** in java is used to restrict the user. It can be used with class , variable and methods .
* **It stops value change** (once value assigned it can’t be change).
* It stops method overriding (If you make any method as final, you cannot override it).
* It stops inheritance (f you make any class as final, you cannot extend it).
* Because constructor is never inherited. That’s why we can not declare it as final .
* Yes, final method is inherited but you cannot override it.(because its static )

1. What do you mean by the Object in java?

* Instance of class is called object.
* Objects contain both data (instance variables or fields) and behavior (methods).
* Its has state and behaviour
* Objects interact with each other by invoking methods or accessing data.
* public class Car {
* private String brand;
* private String model;
* // Constructor
* public Car(String brand, String model) {
* this.brand = brand;
* this.model = model;
* }
* // Getter methods
* public String getBrand() {
* return brand;
* }
* public String getModel() {
* return model;
* }
* }
  + - We can create a Car object:
    - Car myCar = new Car("Toyota", "Camry");

1. How many characteristics an Object possess? What are they?

* The object in java possesses several characteristics like state , behavior (methods) and identity.
* It is a real world entity .
* The state means data or values associated with it and .for example if I want to create an object of car the it will have some name , brand and color etc .
* The behavior means it will perform some actions / will have some methods like it will move , accelerate and brake .
* Unique identity means when creating multiple “Car” objects, each has a distinct identity.
* What are the different ways to create objects in Java?
* **Using new key word**
* **Using newInstance() method**

try {

Class<?> cls = Class.forName("MyClass");

MyClass obj = (MyClass) cls.newInstance();

} catch (ClassNotFoundException | InstantiationException | IllegalAccessException e) {

e.printStackTrace();

}

* **Using clone() method**

The clone() method creates a new object by copying the content of an existing object.

It does not invoke any constructor.

class MyClass implements Cloneable {

@Override

protected Object clone() throws CloneNotSupportedException {

return super.clone();

}

}

MyClass obj1 = new MyClass();

MyClass obj2 = (MyClass) obj1.clone();

* **Using Deserialization**

FileInputStream fileIn = new FileInputStream("object.ser");

ObjectInputStream in = new ObjectInputStream(fileIn);

MyClass obj = (MyClass) in.readObject();

* Explain ways to pass the arguments in Java?
* **Call by value (Pass by value )**
* **Call by reference (Pass by reference)**
* **Command Line Arguments**
* What is static variable in java?

The static variable can be used to to refer common property of all the objects and it is good for memory saving .

* What is static block?

Static block in java is used to run a lock of code for a single time and it runs before the main method at time of class loading .

* What are Packages in Java?
* A java package is a group of similar types of classes, interfaces and sub-packages.
* Package in java can be categorized in two form, built-in package and user-defined package.
* There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.
* What is the abstraction in java?

Process of hiding the implementation details and exposing only essential features of the object . it can ne achieved through abstract classes and interfaces .

* What do you mean by inheritance in java?

When a sub class acquire the property of super class by extending it then it is called inheritance .

* What is a constructor?
* A constructor in Java is a special method **that is used to initialize objects.**
* The constructor is called when an object of a class is created.
* Note that the constructor name must match the class name, and it cannot have a return type (like void).
* Also note that the constructor is called when the object is created.
* All classes have constructors by default: if you do not create a class constructor yourself, Java creates one for you. However, then you are not able to set initial values for object attributes.
* Book example : Suppose we want to create a Book class to represent books in a library. Each book has attributes such as title, author, and ISBN. We’ll use constructors to initialize these attributes when creating a Book object.

class Book {

private String title;

private String author;

private String isbn;

// Constructor with parameters

public Book(String bookTitle, String bookAuthor, String bookIsbn) {

title = bookTitle;

author = bookAuthor;

isbn = bookIsbn;

}

// Getter methods

public String getTitle() {

return title;

}

public String getAuthor() {

return author;

}

public String getIsbn() {

return isbn;

}

}

public class Library {

public static void main(String[] args) {

// Creating Book objects using constructors

Book book1 = new Book("The Great Gatsby", "F. Scott Fitzgerald", "978-0743273565");

Book book2 = new Book("To Kill a Mockingbird", "Harper Lee", "978-0061120084");

// Accessing book information

System.out.println("Book 1: " + book1.getTitle() + " by " + book1.getAuthor());

System.out.println("ISBN: " + book1.getIsbn());

System.out.println("\nBook 2: " + book2.getTitle() + " by " + book2.getAuthor());

System.out.println("ISBN: " + book2.getIsbn());

}

}

* What are the different types of constructors?
* There are two types of constructors in java
* Non argument / default constructor: (If there is no constructor in a class, compiler automatically creates a default constructor). A constructor is called "Default Constructor" when it doesn't have any parameter.

Student4(){

* Parameterized constructor A constructor which has a specific number of parameters is called a parameterized constructor.
* Student4(int i, String n){
* What is default and parameterized constructors?

If there is no constructor in a class, compiler automatically creates a default constructor). A constructor is called "Default Constructor" when it doesn't have any parameter.

* What is polymorphism in java?
* Polymorphism in Java is a concept by which we can perform a single action in different ways. Single method has multiple implementation in different scenario.
* Runtime polymorphism achieved by **method overriding.**
* Compile time polymorphism achieved by **method over loading.**
* Polymorphism in Java is a concept that allows objects of different classes to be treated as objects of a common class. It enables objects to behave differently based on their specific class type.
* What do you mean by the method overloading?
* When a class have multiple method with the same name but with different **parameter or return types** then the method is called overloaded. it can have different parameters in numbers.
* What do you mean by the method overriding in java?
* If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.
* The method must have the same parameter as in the parent class.
* Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
* What is an abstract class in Java?
* Abstract class cannot be instantiated, or we cannot create object of it.
* It can have abstract methods.
* What is Interface in java?
* The interface in Java is a mechanism to achieve abstraction. There can be **only abstract methods in the Java interface, not the method body.**
* It is used to achieve abstraction and multiple inheritances in Java using Interface. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also **represents the IS-A relationship.**
* By interface we can achieved the multiple inheritance.
* To declare an interface, use the interface keyword.
* It is used to provide total abstraction. That means all the methods in an interface are declared with an empty body **and are public and** all fields are **public, static, and final by default.**
* A class that implements an interface must implement all the methods **declared in the interface.**
* To implement the interface, use the implements keyword.
* What is an implementation class of Interface?
* An implementation class is a regular Java class that provides actual implementations for the methods declared in an interface.
* If a method declared in interface the it will be implemented in implementation class .
* interface Drawable {
* void draw(); // Abstract method (no method body)
* }
* class Circle implements Drawable {
* private double radius;
* Circle(double radius) {
* this.radius = radius;
* }
* @Override
* public void draw() {
* System.out.println("Drawing a circle with radius " + radius);
* }
* }
* Describe constructor vs method?

**Constructors**:

* + Initialize newly created objects.
  + Executed implicitly during object creation.
  + No return types.
  + Named after the class.
  + Used to assign initial values to object data members.
  + Cannot be directly invoked by code.
  + Can have multiple constructors with different parameters.
  + Not inherited by subclasses.

**Methods**:

* + Perform specific tasks.
  + Explicitly invoked by the programmer.
  + Have a return type (or void).
  + Named arbitrarily.
  + Reusable code for actions on object state or behavior.
  + Can be inherited by subclasses.

Remember, constructors set the stage for an object’s existence, while methods perform actions on that object.

* What are the differences between abstract class and interface?
* Abstract class can have abstract and non-abstract methods.
* Abstract class doesn't support multiple inheritance.
* **Abstract class can have final, non-final, static and non-static variables.**
* An abstract class can extend another Java class and implement multiple Java interfaces.

Interfaces:

* Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
* Interface supports multiple inheritance.
* **Interface has only static and final variables.**
* An interface can extend another Java interface only.
* Members of a Java **interface is public by default.**
* What are the differences between static and non-static methods?
* A static method is a method that belongs to a class, but it does not belong to an instance of that class and this method can be called without the instance or object of that class.
* In the static method, the **method can only access only static data members** and static methods of another class or the same class but cannot access non-static methods and variables.
* The static method cannot be overridden because they belongs to class not to the object.
* In the static method, less memory is used for execution because memory allocation **happens only once because** the static keyword fixed a particular memory for that method in ram.
* **Non static method**
* Every method in Java defaults to a non-static method without a static keyword preceding it. non-static methods can **access any static method and static variable also, without using the object of the class.**
* In the non-static method, the method can access static data members and static methods as well as non-static members and methods of another class or the same class.
* The non-static method can be overridden because of runtime binding.
* In the non-static method, much memory is used for execution because here memory allocation happens when the method is invoked, and the memory is allocated every time when the method is called.
* What is a wrapper class in Java?
* A wrapper class is a class that wraps or **encloses a primitive data** type (such as int, char, boolean, etc.) into an object.
* Wrapper classes are part of **the java. lang 1package.**
* Convert primitive data types into objects.
* Enable operations on primitive values (e.g., calling methods, using collections).
* Provide additional utility methods (e.g., parsing, conversion)
* Autoboxing: Automatic conversion of primitive types to their corresponding wrapper classes.

int num = 42;

Integer numObj = num; // Autoboxing

* Unboxing: Automatic conversion of a wrapper object to its corresponding primitive type.

Integer ageObj = 25;

int age = ageObj; // Unboxing

* What is a path and classPath in Java?

Path:

The Path refers to an environment variable that acts as a mediator between the operating system and developers. Its primary purpose is to inform the system about binary file paths.

When you set the Path, you’re essentially telling the operating system where to find specific executable files (such as java and javac).

To set the Path:

In Windows OS:

set PATH=%PATH%;C:\Program Files\Java\JDK1.5.10\bin

In Unix-like OS:

export PATH=${PATH}:/opt/Java/JDK1.5.10/bin

Note that the Path cannot be overridden by providing a command, and it is used solely by the operating system to locate binary files 1.

ClassPath:

The ClassPath is another environment variable, but it is specifically used by the application ClassLoader or system to locate and load compiled Java bytecodes stored in .class files.

When you run a Java program, the ClassLoader uses the ClassPath to find the necessary classes.

To set the ClassPath:

In Windows OS:

set CLASSPATH=%CLASSPATH%;C:\Program Files\Java\JDK1.5.10\lib

In Unix-like OS:

export CLASSPATH=${CLASSPATH}:/opt/Java/JDK1.5.10/lib

Unlike the Path, the ClassPath can be overridden by adding classpath entries in the manifest file or using commands like set -classpath 1.

Key Differences:

Path:

Refers to the operating system.

Used by the command prompt (CMD) to find binary files.

Must include the .\\bin folder path.

ClassPath:

Refers to the development environment.

Used by the Java compiler and JVM to find library files.

Must include the .\\lib\\jar file or directory path where .java files are available.

* What is Type casting in Java?
* Type casting involves converting a variable from one data type to another. There are two main types of type casting:
* **Widening Type Casting (Implicit Conversion)**
* In widening type casting, a lower data type is automatically converted into a higher one.
* It is also known as implicit conversion or casting down.
* **The order of compatible data types for widening type casting is: byte → short → char → int → long → float → double.**
* **Narrowing Type Casting (Explicit Conversion):**
* In narrowing type casting, a higher data type is manually converted into a lower one.
* It is also known as explicit conversion or casting up.
* If we don’t perform casting explicitly, the compiler reports a compile-time error.
* Examples of narrowing type casting include converting from double to long, long to int, and so on.
* **The order of compatible data types for narrowing type casting is: double → float → long → int → char → short → byte.**
* Explain the access modifiers that can be applied to the inner classes?

All access modifiers are applied to the inner classes.

**public class Outer {**

**public class Inner {**

**// Public inner class**

**}**

**}**

**package mypackage;**

**public class Outer {**

**protected class Inner {**

**// Protected inner class**

**}**

**}**

**package mypackage;**

**class Outer {**

**class Inner {**

**// Inner class with default access**

**}**

**}**

* What is the difference between Error and Exception?

Exception: its run time error

* They often happen due to invalid data input
* An exception represents an unusual event that occurs during program execution and disrupts the normal flow of instructions.
* Exceptions can occur at compile time (e.g., syntax errors) or runtime (e.g., division by zero, null pointer access).
* Developers write code that can throw exceptions intentionally or unintentionally.
* Exceptions can be categorized into two types:
* Checked exceptions: They are checked at compile time .These must be handled explicitly using try-catch blocks or declared using the throws keyword. Examples include IOException and SQLException.
* Unchecked exceptions: These do not require explicit handling. Examples include NullPointerException and ArrayIndexOutOfBoundsException.

Key points about exceptions:

* They can be caught and handled.
* If unhandled, they propagate up the call stack.
* They allow separation of error handling code from regular code.

Error: its compile time

* Compile time errors prevent the code from running due to incorrect syntax. These errors are detected by the Java compiler during compilation.
* missing semicolons, unmatched brackets, or referencing non-existent classes.
* An error indicates a serious problem that usually occurs due to system resource deficiencies.
* Errors are unrecoverable and typically lead to program termination.
* Examples of errors include OutOfMemoryError, StackOverflowError, and LinkageError.
* Errors are always unchecked (i.e., they don’t need explicit handling).
* Key points about errors:
* They are outside the control of the program.
* They often result from system-level issues.
* Examples include running out of memory or stack space.
* What are the types of exceptions?

Checked exceptions:

* These must be handled explicitly using try-catch blocks or declared using the throws keyword.
* **Examples include IO Exception and SQL Exception.**
* **FileNotFoundException**
* **NoSuchFieldException**
* **InterruptedException**
* **NoSuchMethodException**
* **ClassNotFoundException**

Unchecked exceptions:

* These do not require explicit handling.
* The compiler does not check these exceptions.
* Examples **include NullPointer Exception and ArrayIndexOutOfBounds Exception**.
* They are generated due to mistakes in the program (e.g., null pointer access).
* Unchecked exceptions cannot be caught or handled at compile time.
* **NoSuchElementException**
* **UndeclaredThrowableException**
* **EmptyStackException**
* **ArithmeticException**
* **NullPointerException**
* **ArrayIndexOutOfBoundsException**
* What is throw keyword in exception handling?

The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. It allows developers to create custom exceptions and handle exceptional situations. Here are the key points about the throw keyword:

If we throw an unchecked exception, it is essential to handle the exception or declare it in the throws clause.

If we throw a checked exception, it must be handled using a catch block or declared using the throws declaration.

* What is throws keyword?

Case 1: We have caught the exception i.e. we have handled the exception using try/catch block.

Case 2: We have declared the exception i.e. specified throws keyword with the method.

* Difference between throw and throws in Java

|  |  |
| --- | --- |
| throw | throws |
| The **throw** keyword is used inside a function. It is used when it is required to throw an Exception logically. | The **throws** keyword is used in the function signature. It is used when the function has some statements that can lead to exceptions. |
| The **throw** keyword is used to throw an exception explicitly. It **can throw only one exception at a time.** | The **throws** keyword can be used to declare multiple exceptions, separated by a comma. Whichever exception occurs, if matched with the declared ones, is thrown automatically then. |
| **throw** **new** ArithmeticException(); | **public** **static** **void** writeToFile() **throws** Exception |
| **throw** keyword cannot propagate checked exceptions. It is only used to propagate the unchecked Exceptions that are not checked using the throws keyword. | **throws** keyword is used to propagate the checked Exceptions only. |

* What is finally block?
* The finally block is used to define a set of statements that always execute, regardless of whether an exception occurs or not. It is important for clean up and resource management .
* It ensures that certain cleanup or resource management tasks are performed, even if an exception is thrown or caught.

Behavior:

* The finally block executes under the following conditions:
* After the try block completes execution (whether normally or due to an exception).
* After any matching catch block (if an exception is caught).
* Before control is transferred to the next statement (e.g., after a return statement).
* Even if an exception is thrown and caught, the finally block runs.
* If there is no exception, the finally block still executes.
* The finally block is often used for:
* Closing resources (e.g., closing files, database connections).
* Releasing locks or cleaning up state.
* Logging or reporting.
* Can you handle more than one exception in a single catch block?

Yes, we can do that. In Java, it is possible to handle multiple exceptions within a single catch block. This approach allows you to consolidate exception handling code and improve code readability. Here’s how it works:

try {

// Code that may throw exceptions

} catch (IOException | SQLException e) {

// Handle both IOException and SQLException here

}

* Tell about ClassNotFoundException vs NoClassDefFoundError?

**ClassNotFoundException:**

* It occurs when jvm or class loader instance tries to load a class by name but the class with the specified name can not be found in the class path .
* It is a checked exception
* Solution: Ensure that the required class is available on the classpath during runtime.

@Test(expected = ClassNotFoundException.class)

public void givenNoDrivers\_whenLoadDriverClass\_thenClassNotFoundException() throws ClassNotFoundException {

Class.forName("oracle.jdbc.driver.OracleDriver");

}

NoClassDefFoundError?

* It occurs when jvm tries to load the definition of a class but while loading the class definition it encounters an exception , typically due to runtime issues .
* Cause: Occurs when the JVM cannot find the definition of a class at runtime.
* Solution: Verify class initialization and ensure that all dependencies are available.
* Instantiating a class using the new keyword.
* Loading a class via method calls.

@Test(expected = NoClassDefFoundError.class)

public void givenInitErrorInClass\_whenloadClass\_thenNoClassDefFoundError() {

NoClassDefFoundErrorExample sample = new NoClassDefFoundErrorExample();

sample.getClassWithInitErrors();

}

* What is Input Mismatch Exception?
* It is a common exception arises when the user provide wrong data types of input like integer instead **of string or user provide input out or range.**
* Suppose the program expects an integer input, but the user provides a floating-point value or a non-numeric string. In such cases, the InputMismatchException is thrown.
* It is an unchecked exception (a subclass of java.lang.RuntimeException).
* It arises using Scanner class .

import java.util.InputMismatchException;

import java.util.Scanner;

public class InputMismatchExample {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

try {

System.out.println("Enter an integer value:");

int userInput = scanner.nextInt(); // Throws InputMismatchException if non-integer input

System.out.println("You entered: " + userInput);

} catch (InputMismatchException ex) {

System.out.println("Invalid input. Please enter an integer.");

}

}

}

* Explain on Collection Interface.
* The Collection interface serves as one of the root interfaces in the Java Collections Framework. And it extends iterable interface .
* Unlike many other interfaces, it is not directly implemented by any class. Instead, it is implemented indirectly through its subinterfaces.
* It resides in the java.util package.
* Interfaces like List , Set and Queue extends collection interface .
* **List:** Represents an ordered collection where duplicate elements are allowed. Common implementations include ArrayList, Vector, and LinkedList.
* **Set:** Represents an unordered collection where duplicate values are not allowed. Examples include HashSet, TreeSet, and LinkedHashSet.
* SortedSet: Similar to Set, but maintains elements in sorted order. Implemented by TreeSet.
* Queue: Maintains a First-In-First-Out (FIFO) order. Useful for scenarios like task scheduling. Examples include LinkedList and PriorityQueue.
* The Collection interface defines several methods that are inherited by its subinterfaces. Some common methods include:
* add(E e): Adds an element to the collection.
* remove(Object o): Removes the specified element.
* contains(Object o): Checks if the collection contains a given element.
* size(): Returns the number of elements in the collection.
* isEmpty(): Checks if the collection is empty.
* iterator():
* What are the important Methods in Collections Class?
* The Collections class in Java provides a set of static utility methods for working with collections. These methods operate on or return collections and are useful for various tasks. Let’s explore some of the important methods from the Collections class:
* addAll(Collection<? super T> c, T... elements):

Adds all specified elements to the given collection.

List<String> myList = new ArrayList<>();

Collections.addAll(myList, "Apple", "Banana", "Cherry");\

* binarySearch(List<? extends Comparable<? super T>> list, T key):
* Searches for the specified object (key) in a sorted list and returns its position.
* Requires that the list be sorted in ascending order.
* List<Integer> sortedList = Arrays.asList(10, 20, 30, 40, 50);
* int index = Collections.binarySearch(sortedList, 30); // Returns 2
* copy(List<? super T> dest, List<? extends T> src)
* disjoint(Collection<?> c1, Collection<?> c2):
* Checks if two collections have no common elements.
* **emptyList (), emptySet(), emptyMap():Returns immutable empty lists, sets, and maps.**
* synchronizedCollection(Collection<T> c), synchronizedList(List<T> list), synchronizedSet(Set<T> s)
* List<String> emptyList = Collections.emptyList();
* What are the Important Methods in Collection Interface?
* The Collection interface defines several methods that are inherited by its subinterfaces. Some common methods include:
* add(E e): Adds an element to the collection.
* remove(Object o): Removes the specified element.
* contains(Object o): Checks if the collection contains a given element.
* size(): Returns the number of elements in the collection.
* isEmpty(): Checks if the collection is empty.
* iterator():
* What is the difference between Collections & Collection

Collection: it is an interface .

Interface: Collection is an interface present in the java.util package.

Purpose: It represents a group of individual objects as a single unit.

Hierarchy: The collection interface serves as the root interface of the Java Collections Framework.

Sub-Interfaces: The main sub-interfaces of the Collection interface include List, Set, and Queue.

Methods: It defines essential methods like add(), remove(), clear(), size(), and contains().

Collection<String> myCollection = new ArrayList<>();

myCollection.add("Apple");

myCollection.add("Banana");

myCollection.add("Cherry");

Collections: it is a utility class in java which provide static utility method.

* Utility Class: Collections is a utility class present in the java.util package.
* Purpose: It provides static utility methods for operating on collections.
* Static Methods: **All methods in Collections are static.**
* Examples:
* sort(): Sorts collection elements according to the default sorting order.
* min(), max(): Find the minimum and maximum values in the collection.
* addAll(): Adds elements to a collection.
* Usage:

List<String> myList = new ArrayList<>();

myList.add("geeks");

myList.add("for");

myList.add("geeks");

Collections.addAll(myList, "web", "site");

Collections.sort(myList);

* What is List?
* List is an interface which extends collection interface.
* It is used in collection farmework to stored **ordered data .**
* **Duplicate value allowed in list .**
* Elements in a List are stored in the order they were added.
* You can access the element using index .
* Classes like ArraList , LinkedList and vector implements List Interface .
* Common method add() , get (), remove() , size() and
* Several classes implement the List interface:
* ArrayList: A dynamic array that provides fast random access.
* LinkedList: A doubly-linked list that allows efficient insertions and deletions.
* Vector: Similar to ArrayList, but synchronized (use ArrayList instead, as Vector is deprecated).
* Stack: A specialized list for implementing a stack data structure.
* What is Map?
* In Java, a Map is an interface that allows you to store data in **a key-value format.**
* Each key is unique, and it maps to a corresponding value.
* It uses key to retrieve value .
* No duplicate KEY allowed in map .
* Common Implementations: Popular **implementations include HashMap, LinkedHashMap, and TreeMap.**
* Common Methods (from the Map interface):
* put(K key, V value): Inserts a key-value pair into the map.
* get(Object key): Retrieves the value associated with a given key.
* remove(Object key): Removes the entry for a specified key.
* containsKey(Object key): Checks if a key exists in the map.
* keySet(): Returns a set of all keys.
* entrySet(): Returns a set of key-value pairs.
* What is Set?
* A Set in Java is an interface that represents **an unordered collection of objects**.
* Set interface extends collection interface and it stores unordered objects .
* It does not allow duplicate values
* Use a set to automatically remove duplicate value .
* Several classes implement the Set interface:
* HashSet: Uses a hash table for efficient storage and retrieval.
* LinkedHashSet: Maintains insertion order along with uniqueness.
* TreeSet: Stores elements in a sorted order (natural or custom).
* Set<String> fruitSet = new HashSet<>();
* fruitSet.add("Apple");
* boolean containsCherry = fruitSet.contains("Cherry");
* fruitSet.remove("Cherry");
* Why ArrayList is better than Arrays?
* ArrayList provides dynamic sizing. You don’t need to specify the size upfront. While Regular arrays have a fixed size, which can lead to inefficiencies if you need to resize them frequently.
* When an ArrayList reaches its capacity, it automatically resizes itself (doubles its size by default).Regular arrays cannot resize, so you must manually create a new array with a larger size and copy elements over.
* ArrayList provides methods like add(), remove(), and contains(). Regular arrays lack these convenient methods, requiring manual element shifting.
* ArrayList can hold any type of objects like in a same Arraylist we can store integer , string of (including custom classes). But array can hold only one data types only integer of only string .
* **ArrayList operations (like adding/removing elements) are slower due to resizing and method calls.Regular arrays are faster for direct element access (no method calls).**
* ArrayList consumes more memory due to **its internal array and additional object overhea**d.Regular arrays are memory-efficient but lack flexibility.
* Depends on the situations. In general arrays have a fixed size in Java, which means once the size of an array is declared, it cannot be changed. Its limited to that size.
* In cases, where you are not certain of the size of your collection (in your case Array), its better to use ArrayList. The default size of ArrayList (as of Java 8) is 10. Once the size of ArrayList reaches its maximum, the size of ArrayList is increased dynamically to its double. Therefore, in this case, the size would become 20. Below is the syntax of initializing an ArrayList.
* Array vs ArrayList
* Array and ArrayList both are used for storing elements.
* Array and ArrayList both can store null values.
* They can have duplicate values.
* They do not preserve the order of elements.

The following table describes the key differences between array and ArrayList:

|  |  |
| --- | --- |
| **Array** | **ArrayList** |
| **Definition** | An **array** is a dynamically-created object. It serves as a container that holds the constant number of values of the same type. It has a contiguous memory location. | The **ArrayList** is a class of Java **Collections** framework. It contains popular classes like **Vector, HashTable**, and **HashMap**. |
| **Static/ Dynamic** | Array is **static** in size. | ArrayList is **dynamic** in size. |
| **Resizable** | An array is a **fixed-length** data structure. | ArrayList is a **variable-length** data structure. It can be resized itself when needed. |
| **Initialization** | It is mandatory to provide the size of an array while initializing it directly or indirectly. | We can create an instance of ArrayList without specifying its size. Java creates ArrayList of default size. |
| **Performance** | It performs **fast** in comparison to ArrayList because of fixed size. | ArrayList is internally backed by the array in Java. The resize operation in ArrayList slows down the performance. |
| **Primitive/ Generic type** | An array can store both **objects** and **primitives** type. | We cannot store **primitive** type in ArrayList. It automatically converts primitive type to object. |
| **Iterating Values** | We use **for** loop or **for each** loop to iterate over an array. | We use an **iterator** to iterate over ArrayList. |
| **Type-Safety** | We cannot use generics along with array because it is not a convertible type of array. | ArrayList allows us to store only **generic/ type, that's why it is type-safe.** |
| **Length** | Array provides a **length** variable which denotes the length of an array. | ArrayList provides the **size()** method to determine the size of ArrayList. |
| **Adding Elements** | We can add elements in an array by using the **assignment**operator. | Java provides the **add()** method to add elements in the ArrayList. |
| **Single/ Multi-Dimensional** | Array can be **multi-dimensional**. | ArrayList is always **single-dimensional**. |

* What is the difference between ArrayList and LinkedList?

In Java, **ArrayList** is a class of Collections framework. It implements **List<E>, Collection<E>, Iterable<E>, Cloneable, Serializable**, and **RandomAccess** interfaces. It extends **AbstractList<E>** class.

We can not store primitive data types into ArrayList.

ArrayList arrayObj=**new** ArrayList()//object of ArrayList

arrayObj(**new** Integer(12)); //converts integer primitive to Integer object and added to ArrayList object

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly linked list** to store the elements. |
| Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. |
| An ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |
| The memory location for the elements of an ArrayList is contiguous. | The location for the elements of a linked list is not contagious. |
| Generally, when an ArrayList is initialized, a default capacity **of 10 is assigned** to the ArrayList. | There is no case of default capacity in a LinkedList. In LinkedList, an empty list is created when a LinkedList is initialized. |
| To be precise, an ArrayList is a resizable array. | LinkedList implements the doubly linked list of the list interface. |

* Which one is most preferred for addition and deletion? ArrayList or LinkedList?
* LinkedList is Better for manipulating data (additions/removals).

Adding or removing elements is faster (O(1)) because it involves changing pointers.

* In ArrayList adding or removing elements in the middle of the list is slower (O(n)) because it involves shifting elements. That’s why linked is good for add and remove .
* If you mostly need to insert and delete elements at the start or middle of the container, then a LinkedList might be a better option.
* Which one is most preferred For searches? ArrayList or LinkedList?
* For searches or fast random access Arraylist is preferred because it uses index to retrieve element (O(1))
* In LinkedList retrieving elements by index is slower (O(n)) because it requires traversing the list.
* What is the difference between Iterator and Enumeration?

Iterator and Enumeration both are the cursors to traverse and access an element from the collection. They both belong to the collection framework. Enumeration was added in JDK1.0 and Iterator in the JDK.1.2 version in the collection framework.

**Enumeration :**

* Enumeration can’t make structural changes in the collection because it has read-only access to the element in the collection. It has the following methods :
* It applies only to legacy classes (Vector and Hashtable).
* Supports only forward traversal.
* Provides read-only access to elements; no remove() method.
* Supports only forward traversal.
* **hasMoreElements()**
* **nextElement()**

// Using Enumeration (legacy)

Vector<Integer> myVector = new Vector<>();

Enumeration<Integer> enumeration = myVector.elements();

while (enumeration.hasMoreElements()) {

Integer value = enumeration.nextElement();

// Process value

}

**Interator :**

* On the other hand, an **iterator can read and remove** the element in the collection. It has the following methods –
* It is applicable for all collection classes (including newer ones like ArrayList, HashSet, etc.).
* Supports both forward and backward traversal.
* You can add and remove elements during traversal using the remove() method.
* Traversing newer collections like ArrayList, LinkedList, HashSet, TreeMap, and TreeSet.
* hasNext()
* next()
* remove()

// Using Iterator

List<String> myList = new ArrayList<>();

Iterator<String> iterator = myList.iterator();

while (iterator.hasNext()) {

String element = iterator.next();

// Process element

}

* What is polymorphism ?
* Define method overloading in java ?
* Use of this keyword and super keyword ?
* Types of constructor ?? define its types and uses ?
* What is meaning of destructor??
* The destructor is the opposite of the constructor. The constructor is used to initialize objects while the destructor is **used to delete or destroy the object** that releases the resource occupied by the object.
* In Java, when we create an object of the class it occupies some space in the memory (heap). If we do not delete these objects, it remains in the memory and occupies unnecessary space that is not upright from the aspect of programming.
* It is a special method that automatically gets called when an object is no longer used. When an object completes its life-cycle the garbage collector deletes that object and deallocates or releases the memory occupied by the object.
* It does not accept any parameter and cannot be overloaded.
* The Java Object class provides the finalize() method that works the same as the destructor
* It is noted that when the garbage collector destroys the object, the JRE calls the finalize() method to close the connections such as database and network connection.
* Explicitly calling finalize() is uncommon and not recommended. The garbage collector automatically invokes it when needed.
* Modern Java code rarely uses finalize(), as it has limitations and is less predictable.
* Destructor and garage collector are used to destroy the unused object and destructor notifies exactly when the object will be destroyed while garbage collector does the same work automatically

protected void finalize throws Throwable()

{

//resources to be close

}

* Can we initialize blank final variable?

**Yes, but only in constructor. For example:**

class Bike10{

final int speedlimit; //blank final variable

Bike10(){ // constructor

speedlimit=70;

System.out.println(speedlimit);

}

public static void main(String args[]){

new Bike10();

}

}

Out : 70

* What is blank or uninitialized final variable?
* A final variable that is not initialized at the time of declaration is known as blank final variable.
* If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful. For example PAN CARD number of an employee.
* It can be initialized only in constructor.

class Student{

int id;

String name;

final String PAN\_CARD\_NUMBER;

...

}

* Can we execute a program without main() method?
* No, one of the ways was the static block, but it was possible till JDK 1.6. Since JDK 1.7, it is not possible to execute a Java class without the [main method](https://www.javatpoint.com/java-main-method).
* How to run java package program

|  |
| --- |
| **To Compile:** javac -d . Simple.java |
| **To Run:** java mypack.Simple  Upcasting : If the reference variable of Parent class refers to the object of Child class, it is known as upcasting   * What is blank or uninitialized final variable? * A final variable that is not initialized at the time of declaration is known as blank final variable. * If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful. For example PAN CARD number of an employee. * It can be initialized only in constructor. * **class** Student{ * **int** id; * String name; * **final** String PAN\_CARD\_NUMBER; * ... * } * Can we initialize blank final variable?   Yes, but only in constructor.  class Bike10{  final int speedlimit;//blank final variable    Bike10(){  speedlimit=70;  System.out.println(speedlimit);  }    public static void main(String args[]){  new Bike10();  }  }  Output -70   * What is static blank final variable * A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block. * class A{ * static final int data; //static blank final variable * static{ data=50;} * public static void main(String args[]){ * System.out.println(A.data); * } * } * Can we declare a constructor final?   No, because constructor is never inherited.   * Give real world example of encapsulation   **Bank Account Balance**:   * 1. Suppose you have a bank account.   2. If the balance variable were declared as public, anyone could directly access it.   3. To maintain security, the balance variable is declared as **private**.   4. Access to the balance is provided through **getter methods**, which ask for authentication (account holder name, user ID, and password).   5. This encapsulation ensures that only authorized users can access account details. * **Capsule (Medicine)**:   1. Just like a **capsule** encloses various combinations of medicine, encapsulation in Java involves bundling data and methods.   2. The capsule acts as a class, and the medicine combinations are like private members.   3. The whole process ensures that the medicine’s complexity is hidden from external access. * Storage of variable in stack * The declaration int a = 10; creates a primitive integer variable named a. * This variable is stored in the stack. * The value 10 is directly stored in the reserved space on the stack. * If you were dealing with a non-primitive type, such as an Integer (which is an object wrapper for int): * The reference to the Integer object would be stored in the stack. * The actual Integer object (with the value 10) would be allocated in the heap. * Certainly! Let’s break it down for the declaration String name = "Azhar" in Java: * The variable name is of type String, which is a non-primitive type (an object). * The reference to the "Azhar" string literal will be stored in the stack. * The actual String object (containing the characters “Azhar”) will be allocated in the heap.   Remember:  Primitive types (like int, double, etc.) store their values directly in the stack.  Objects (non-primitive types) store references to their memory locations in the heap. Why String class is Final in Java?  * The reason behind the String class being final is because no one can override the methods of the String class. So that it can provide the same features to the new String objects as well as to the old ones.  Why String objects are immutable in Java?  * As Java uses the concept of String literal. Suppose there are 5 reference variables, all refer to one object "Sachin". If one reference variable changes the value of the object, it will be affected by all the reference variables. That is why String objects are immutable in Java. * String s="Sachin"; * s.concat(" Tendulkar");//concat() method appends the string at the end * System.out.println(s);//will print Sachin because strings are immutable objects * As you can see in the above figure that two objects are created but **s** reference variable still refers to "Sachin" not to "Sachin Tendulkar". * But if we explicitly assign it to the reference variable, it will refer to "Sachin Tendulkar" object. * String s="Sachin"; * s=s.concat(" Tendulkar"); * System.out.println(s);   output : Sachin Tendulkar   In case, s points to the " Sachin Tendulkar". Please notice that still Sachin object is not modified.   * How Java platform independent ? * **Java** is considered **platform independent** due to its ability to **compile code into bytecode**, which is then executed by the **Java Virtual Machine (JVM)**. Let’s explore why Java achieves this and what it means: * When you write Java code, it is first **compiled** into an **intermediate form** called **bytecode**. * **Write Once, Run Anywhere (WORA)**: * Java’s platform independence is often summarized as **“Write Once, Run Anywhere.”** * This means that once you’ve written and compiled your Java code, you can run it on **any platform** that has a compatible JVM. * You don’t need to modify the code for each specific operating system. * Why use Static Methods? * To access and change static variables and other non-object-based static methods. * Utility and assist classes frequently employ static methods. * **Global Accessibility**: They provide global accessibility without needing to create instances. * To save memory : No need to create instances, saving memory and processing time.  ****Restrictions in Static Methods:****  * Non-static data members or non-static methods cannot be used by static methods, and static methods cannot call non-static methods directly. * In a static environment, this and super aren’t allowed to be used. * It can not be overridden but I can be overloaded . * However, **static methods cannot be overridden** in the usual sense because they are belongs to classs and bound at compile time, not runtime. * When a subclass defines a static method with the same signature as a static method in the base class, it **hides** the method in the base class rather than overriding it. * Why is the main method in Java static? * It’s because calling a static method isn’t needed of the object. If it were a non-static function, JVM would first build an object before calling the main() method, resulting in an extra memory allocation difficulty. * **Features of static method:** * A static method in Java is a method that is part of a class rather than an instance of that class. * Every instance of a class has access to the method. * Static methods have access to class variables (static variables) without using the class’s object (instance). * Only static data may be accessed by a static method. It is unable to access data that is not static (instance variables).   public class MyClass {  static int a = 40; // Static variable  int b = 50; // Instance variable  void simpleDisplay() {  System.out.println(a); // Accessing static variable  System.out.println(b); // Accessing instance variable  }  static void staticDisplay() {  System.out.println(a); // Accessing static variable  // System.out.println(b); // Error! Cannot access instance variable  }   * In both static and non-static methods, static methods can be accessed directly. * Why need static variable * **Code Organization**: Static methods help organize related functions within a class. * **Global Accessibility**: They provide global accessibility without needing to create instances. * **Utility Functions**: Static methods are commonly used for utility functions and helper tasks. * Examples of utility classes in Java include **Math, Arrays, and Collections**. * Converting String to int:   1. You can convert a String containing numeric characters to an int using either of the following methods:   2. Using Integer.valueOf():   3. String str = "12345";   4. int value = Integer.valueOf(str); // Converts "12345" to 12345   5. System.out.println("The integer value is: " + value);   Using Integer.parseInt():   * 1. String str = "12345";   2. int value = Integer.parseInt(str); // Converts "12345" to 12345   3. System.out.println("The integer value is: " + value);   Converting int to String:   * 1. To convert an int to a String, you can use the String.valueOf() method:   2. int num = 9876;   String str = String.valueOf(num); // Converts 9876 to "9876"   * Size of data types? * byte: 1 byte * short: 2 bytes * char: 2 bytes * int: 4 bytes * float: 4 bytes * long: 8 bytes * double: 8 bytes * Overloading and overriding different method in java   1. Static method – overloading allowed but can not override (static belongs to class )   2. Abstract method – both allowed   3. Final method – override and overload both not allowed   4. Constructor – both allowed main   5. method – only overloading allowed but not override because its static * final class in java   + Final classes are often used for **utility classes**.   + Utility classes contain static methods or constants that provide common functionality across your application.   + Examples include classes for mathematical calculations, date formatting, or file manipulation.   + A **final class** is one that cannot be **extended (inherited)** by any other class.   + All wrapper classes (such as Integer, Float, etc.) are declared as final.   + This ensures that they cannot be extended or modified.   Forward and back function in selenium ?   * + driver.forward() - Navigates forward to the next page in the browser’s history (if available).   + driver.back() -  Takes you back to the previous page in the browser’s history.   Thread.sleep () vs wait() method   * sleep() is for **pausing and belongs to Thread class** , while wait() is for **synchronization and belongs to Object class**. * sleep() does **not** release the lock, whereas wait() releases it.  |  |  |  | | --- | --- | --- | | **No.** | **Wait()** | **Sleep()** | | 1. | The Wait() method is related to the Object class. | The Sleep () method is related to the Thread class. | | 2. | It releases lock | The Sleep () method does not release the lock on the object during Synchronization. | | 3. | It is not a static method. | It is a static method. | | 4. | At the time of the Synchronization, the Wait() method releases obj. | At the time of the Synchronization, the Sleep() method doesn't release the obj, i.e., lock. | | 5. | We can call the Wait () method only from the Synchronized context. | We can call the Sleep () method from outside the Synchronized context. | | 6. | The Sleep() method has two overloaded methods, which are as follows:   * sleep(long milliseconds, int nanoseconds) * sleep(long milliseconds) | The Sleep() method has three overloaded methods, which are as follows:   * Wait() * wait(long timeout, int nanoseconds) * wait(long timeout) | | 7. | The constructor of the Wait() method is defined in the following way: public final void Wait(long timeout) | The constructor of the Sleep () method in the following way: public static void Sleep (long millis) throws Interrupted\_Execption |   Difference between local instance and static variable   |  |  |  |  | | --- | --- | --- | --- | | **Sr. No.** | **Local variables** | **Instance variables** | **Static variables** | | 1. | Variables declared within a method are local variables. | An instance variable is declared inside a class but outside of any method or block. | Static variables are declared inside a class but outside of a method starting with a keyword static. | | 2. | The scope of the local variable is limited to the method it is declared inside. | An instance variable is accessible throughout the class. | The static variable is accessible throughout the class. | | 3. | A local variable starts its lifetime when the method is invoked. | The object associated with the instance variable decides its lifetime. | The static variable has the same lifetime as the program. | | 4. | Local variable is accessible to all the objects of the class. | Instance variable has different copies for different objects. | Static variables only have one single copy of the entire class. | | 5. | Used to store values that are required for a particular method. | Used to store values that are needed to be accessed by different methods of the class. | Used for storing constants. |   Null vs Empty in java   * **An empty String is a valid String object having no characters,** and as a result, all the String operations are available on this object. * **null is a reserved keyword in Java that signifies the absence of any value**. Moreover, assigning a null value to an object reference implies that it doesn’t refer to any object or value in the memory. * By default, Java initializes reference variables with null values and primitives with default values based on their type. As a result, **we cannot assign null to primitives**. * We can not use string operation/ methods on null string .   **String** nullString = null; **String** emptyString = ""; assertTrue(emptyString.equals(""));  assertThrows(NullPointerException.class, () -> nullString.length());  What are the issues with thread.sleep () in java  One of the most common issues you might encounter when using Thread. sleep() is InterruptedException . This exception is thrown when another thread interrupts the current thread while it's sleeping or waiting.  Assert vs verify in selenium ?   * When an "assert" command fails then test execution will be aborted. * Assert is best used when the check value has to pass for the test to be able to continue to run. Like a log in. * When a "verify" command fails then test will continue executing and logging the failure. * Verify is best used to check non critical things. Like hte presence of a headline element.   Difference between abstract class and interface   | **Abstract Class** | **Interface Class** | | --- | --- | | Both abstract and non-abstract methods may be found in an abstract class. | The interface contains only abstract methods. | | Abstract Class supports Final methods. | The interface class does not support Final methods. | | Multiple inheritance is not supported by the Abstract class. | Multiple inheritances is supported by Interface Class. | | Abstract Keyword is used to declare Abstract class. | Interface Keyword is used to declare the interface class. | | **extend**keyword is used to extend an Abstract Class. | **implements**Keyword is used to implement the interface. | | Abstract Class has members like protected, private, etc. | All class members are public by default. |  What are the advantages of Encapsulation in Java?  * Data Hiding:  it is a way of restricting the access of our data members by hiding the implementation details. Encapsulation also provides a way for data hiding. The user will have no idea about the inner implementation of the class. * Biding the data into the single unit . * The main benefit is of providing a way to control and manage the state and the behavior of an object and also protecting it from modification and unauthorized access at the same time.   **import** **java.io.\***;  **class** **Person** {  **private** String Name;  **private** int age;  **public** String getName() { **return** Name; }  **public** void setName(String Name) { **this**.Name = Name; }  **public** int getAge() { **return** age; }  **public** void setAge(int age) { **this**.age = age; }  }  *// Driver class*  **class** **GFG** {  *// main function*  **public** **static** void main(String[] args)  {  Person p = **new** Person();  p.setName("Rohan");  p.setAge(29);  System.out.println("Name is " + p.getName());  System.out.println("Age is " + p.getAge());  }  }  Different types of inheritance in java?  Single inheritance - When a child or subclass extends only one superclass  **Multilevel inheritance -** When a child or subclass extends any other subclass a hierarchy of inheritance is created which is known as multilevel inheritance.  Class B extends A and class C extends B  **Hierarchical inheritance –** one parent class multiple child class  B and C extends class A  **Multiple inheritance –** by using interfaces class A implements interface x and y.  Aggregation - Aggregation is a term related to the relationship between two classes best described as a “has-a” relationship.  IS -A relationship - ‘IS-A’ is a type of relationship in OOPs Java where one class inherits another class.  Association-The association is a relation between two separate classes established through their Objects. It represents Has-A’s relationship. What is the composition of Java? Composition implies a relationship where the child **cannot exist independently** of the parent. For example Human heart, the heart doesn’t exist separately from a Human. Can we override the private methods?No, The private methods are accessible only within the class in which it is declared. Since this method is not visible to other classes and cannot be accessed, it cannot be overridden.Can we change the scope of the overridden method in the subclass? In Java, it is not possible to modify the overridden method’s scope. The subclass method’s scope must be equal to or wider than the Superclass method’s overridden method’s scope. Explain the method to convert ArrayList to Array and Array to ArrayList.  Array to ArrayList using asList() method of the Arrays class  Collection.addAll()  Add() method  Arrays.asList(item) Conversion of ArrayList to ArrayList\_object.toArray(new String[List\_object.size()]) How to make Java ArrayList Read-Only? An ArrayList can be made ready only using the method provided by Collections using the Collections.unmodifiableList() method. What is a Vector in Java? Vectors in Java are similar and can store multiple elements inside them. Vectors follow certain rules mentioned below:   * Vector can be imported using Java.util.Vector. * Vector is implemented using a dynamic array as the size of the vector increases and decreases depending upon the elements inserted in it. * Elements of the Vector using index numbers. * Vectors are synchronized in nature means they only used a single thread ( only one process is performed at a particular time ).  What is the Stack class in Java and what are the various methods provided by it? A Stack class in Java is a LIFO data structure that implements the Last In First Out data structure.   * peek(): returns the top item from the stack without removing it * empty(): returns true if the stack is empty and false otherwise * push(): pushes an item onto the top of the stack * pop(): removes and returns the top item from the stack   Sets- Sets are collections that don’t store duplicate elements. They don’t keep any order of the elements.  Map- The map interface is present in the Java collection and can be used with Java.util package. A map interface is used for mapping values in the form of a key-value form. The map contains all unique keys.  There are multiple types of maps in the map interface as mentioned below:   * SortedMap * TreeMap * HashMap * LinkedHashMap   What is the difference between Collection and Collections?  | **Collection** | **Collections** | | --- | --- | | The Collection is an Interface. | Collections is a class. | | It provides the standard functionality of data structure. | It is to sort and synchronize the collection elements. | | It provides the methods that can be used for the data structure. | It provides static methods that can be used for various operations. |  142. Differentiate between Array and ArrayList in Java.  | **Array** | **ArrayList** | | --- | --- | | Single-dimensional or multidimensional | Single-dimensional | | For and for each used for iteration | Here iterator is used to traverse riverArrayList | | length keyword returns the size of the array. | size() method is used to compute the size of ArrayList. | | The array has Fixed-size. | ArrayList size is dynamic and can be increased or decreased in size when required. | | It is faster as above we see it of fixed size | It is relatively slower because of its dynamic nature | | Primitive data types can be stored directly in unlikely objects. | Primitive data types are not directly added to unlikely arrays, they are added indirectly with help of autoboxing and unboxing | | They can not be added here hence the type is in the unsafe. | They can be added here hence makingArrayList type-safe. | | The assignment operator only serves the purpose | Here a special method is used known as add() method |  **143.** What is the difference between Array and Collection in Java?  | **Array** | **Collections** | | --- | --- | | Array in Java has a fixed size. | Collections in Java have dynamic sizes. | | In an Array, Elements are stored in contiguous memory locations. | In Collections, Elements are not necessarily stored in contiguous memory locations. | | Objects and primitive data types can be stored in an array. | We can only store objects in collections. | | Manual manipulation is required for resizing the array. | Resizing in collections is handled automatically. | | The array has basic methods for manipulation. | Collections have advanced methods for manipulation and iteration. | | The array is available since the beginning of Java. | Collections were introduced in Java 1.2. |  144. Difference between ArrayList and LinkedList.  | **ArrayList** | **LinkedList** | | --- | --- | | ArrayList is Implemented as an expandable Array. | LinkedList is Implemented as a doubly-linked list. | | In ArrayList, Elements are stored in contiguous memory locations | LinkedList Elements are stored in non-contiguous memory locations as each element has a reference to the next and previous elements. | | ArrayLists are faster for random access. | LinkedLists are faster for insertion and deletion operations | | ArrayLists are more memory efficient. | LinkedList is less memory efficient | | ArrayLists Use more memory due to maintaining the array size. | LinkedList Uses less memory as it only has references to elements | | The search operation is faster in ArrayList. | The search operation is slower in LinkedList |  145. Differentiate between ArrayList and Vector in Java.  | **ArrayList** | **Vector** | | --- | --- | | ArrayLists are implemented as an expandable array. | Vector is Implemented as a growable array. | | ArrayList is not synchronized. | The vector is synchronized. | | ArrayLists are Faster for non-concurrent operations. | Vector is Slower for non-concurrent operations due to added overhead of synchronization. | | ArrayLists were Introduced in Java 1.2. | Vector was Introduced in JDK 1.0. | | Recommended for use in a single-threaded environment. | Vectors are Recommended for use in a multi-threaded environment. | | The default initial capacity of ArrayLists is 10. | In Vectors, the default initial capacity is 10 but the default increment is twice the size. | | ArrayList performance is high. | Vector performance is low. |  146. What is the difference between Iterator and ListIterator?  | **Iterator** | **ListIterator** | | --- | --- | | Can traverse elements present in Collection only in the forward direction. | Can traverse elements present in Collection both in forward and backward directions. | | Used to traverse Map, List, and Set. | Can only traverse List and not the other two. | | Indexes can’t be obtained using Iterator | It has methods like nextIndex() and previousIndex() to obtain indexes of elements at any time while traversing the List. | | Can’t modify or replace elements present in Collection | Can modify or replace elements with the help of set(E e) | | Can’t add elements, and also throws ConcurrentModificationException. | Can easily add elements to a collection at any time. | | Certain methods of Iterator are next(), remove(), and hasNext(). | Certain methods of ListIterator are next(), previous(), hasNext(), hasPrevious(), add(E e). |  147. Differentiate between HashMap and HashTable.  | **HashMap** | **HashTable** | | --- | --- | | HashMap is not synchronized | HashTable is synchronized | | One key can be a NULL value | NULL values not allowed | | The iterator is used to traverse HashMap. | Both Iterator and Enumertar can be used | | HashMap is faster. | HashTable is slower as compared to HashMap. |  148. What is the difference between Iterator and Enumeration?  | **Iterator** | **Enumeration** | | --- | --- | | The Iterator can traverse both legacies as well as non-legacy elements. | Enumeration can traverse only legacy elements. | | The Iterator is fail-fast. | Enumeration is not fail-fast. | | The Iterators are slower. | Enumeration is faster. | | The Iterator can perform a remove operation while traversing the collection. | The Enumeration can perform only traverse operations on the collection. |  149. What is the difference between Comparable and Comparator?  | **Comparable** | **Comparator** | | --- | --- | | The interface is present in java.lang package. | The Interface is present in java.util package. | | Provides compareTo() method to sort elements. | Provides compare() method to sort elements. | | It provides single sorting sequences. | It provides multiple sorting sequences. | | The logic of sorting must be in the same class whose object you are going to sort. | The logic of sorting should be in a separate class to write different sorting based on different attributes of objects. | | Method sorts the data according to fixed sorting order. | Method sorts the data according to the customized sorting order. | | It affects the original class. | It doesn’t affect the original class. | | Implemented frequently in the API by Calendar, Wrapper classes, Date, and String. | It is implemented to sort instances of third-party classes. |  150. What is the difference between Set and Map?  | **Set** | **Map** | | --- | --- | | The Set interface is implemented using java.util package. | The map is implemented using java.util package. | | It can extend the collection interface. | It does not extend the collection interface. | | It does not allow duplicate values. | It allows duplicate values. | | The set can sort only one null value. | The map can sort multiple null values. |  | **this( )** | **super( )** | | --- | --- | | It represents the current instance of the class. | It represents the current instance of the parent class. | | Calls the default constructor of the same class. | Calls the default constructor of the base class. | | Access the methods of the same class. | Access the methods of the parent class. | | Points current class instance. | Points the superclass instance. |  | **Process** | **Thread** | | --- | --- | | A process is a program in execution. | A thread is a single sequence of instructions within a process. | | The process takes more time to terminate. | The thread takes less time to terminate. | | The process takes more time for context switching. | The thread takes less time for context switching. | | The process is less efficient in terms of communication. | Thread is more efficient in terms of communication. | | The process is isolated. | Threads share memory. |  | **Iterable** | **Iterator** | | --- | --- | | Iterable provides a way to iterate over a sequence of elements. | Iterator helps in iterating over a collection of elements sequentially. | | **iterator()** method returns an Iterator. | **hasNext()** and **next()**methods are required. | | **remove()**method is optional. | **remove()**method is required in the iterator. | | Examples are **List, Queue, and Set.** | Examples are **ListIterator, Enumeration, and ArrayIterator.** |  | **List** | **Set** | | --- | --- | | Ordered | Unordered | | List allows duplicates. | Set does not allow duplicate values. | | List is accessed by index. | Set is accessed by hashcode. | | Multiple null elements can be stored. | Null element can store only once. | | Examples are ArrayList, LinkedList, etc. | Examples are HashSet and TreeSet. LinkedHashSet etc. |  | **List** | **Map** | | --- | --- | | List interface allows duplicate elements. | Map does not allow duplicate elements. | | The list maintains insertion order. | The list maintains insertion order. | | Multiple null elements can be stored. | The map allows a single null key at most and any number of null values. | | The list provides get() method to get the element at a specified index. | The map does not provide a get method to get the elements at a specified index. |  | **Queue** | **Stack** | | --- | --- | | Queue data structure is used to store elements, and is used to perform operations like enqueue, dequeue from back or end of the queue. | Stack data structure is used to store elements, and is used to perform operations like push, pop from top of the stack. | | Queue data structure Implements FIFO order. | Stack data structure Implements LIFO order. |  | **Priority Queue** | **TreeSet** | | --- | --- | | It uses Queue as an underlying data structure. | It uses a Set as an underlying data structure. | | This data structure allows duplicate elements | This data structure does not allow duplicate elements | | Priority Queue is Implemented by PriorityQueue class. | TreeSet is implemented by TreeSet class. |  | **Singly Linked List** | **Doubly Linked List** | | --- | --- | | Singly Linked List contain only two segments i.e, Data and Link. | Doubly Linked List contains three segments i.e, Data, and two pointers. | | Traversal in a singly linked list is possible in only a forward direction. | Traversal in a doubly linked list is only possible in both directions forward as well as backward. | | It uses less memory as every single node has only one pointer. | It requires more memory than a singly linked list as each node has two pointers. | | Easy to use and insert nodes at the beginning of the list. | Slightly more complex to use and easy to insert at the end of the list. | | The time complexity of insertion and deletion is O(n). | The time complexity of insertion and deletion is O(1). |  | **FailFast** | **FailSafe** | | --- | --- | | Failsfast fails immediately when it detects concurrent modification during the time of iteration. | Failsafe continues to iterate over the original collection and also creates a copy to modify. | | Failfast is generally used in single-threaded environments. | Failsafe is used in multithreaded environments. |  | **HashMap** | **TreeMap** | | --- | --- | | Hasmap uses a hashtable in order to store key-value pairs. | Treemap uses Red-black trees to store key-value pair. | | Hashmap does not maintain any specific order for key-value pairs. | Treemap maintains a natural ordering based on the keys. | | Order of iteration is not guaranteed in the hashmap. | Iteration is of sorted order based on keys. |  | **Queue** | **Deque** | | --- | --- | | The queue is a linear Data structure that is used to store a collection of elements. | Deque also known as a Double-ended queue is also a linear data structure that stores a collection of elements with operations to remove and add from both ends. | | Elements in the queue can only be inserted at the end of the data structure. | Elements can be inserted from both ends of the data structure. | | Queue can be implemented using Array or Linked List. | Dequeue can be implemented using Circular Array or Doubly Linked List. |  | **HashSet** | **TreeSet** | | --- | --- | | HashSet is unordered. | TreeSet is based on natural ordering. | | HashSet allows null elements. | TreeSet does not allow null elements. | | HashSet is Implemented by the HashSet class. | TreeSet is Implemented by TreeSet class. |  What do you understand by Object Cloning and how do you achieve it in Java? It is the process of creating an exact copy of any object. In order to support this, a java class has to implement the Cloneable interface of java.lang package and override the clone() method provided by the Object class the syntax of which is:  Protected Object clone() throws CloneNotSupportedException{ return (Object)super.clone();} finalize method It is a method that is called just before deleting/destructing the objects which are eligible for Garbage collection to perform clean-up activity.  **public** void finalize()  {  System.out.println("finalize method overridden");  } final keyword final is a keyword is used with the variable, method, or class so that they can’t be overridden  finally block finally is a block of code used with “try-catch” in exception handling. Code written in finally block runs despite the fact exception is thrown or not.  What are the two ways in which Thread can be created?  * Thread is java is a light weight process * By extending the Thread class * By implementing a Runnable interface. * **New:** The thread has been created but has not yet started. * **Runnable:** The thread is running, executing its task, or is ready to run if there are no other higher-priority threads. * **Blocked:**The thread is temporarily suspended, waiting for a resource or an event. * **Waiting:** The thread is waiting for another thread to perform a task or for a specified amount of time to elapse. * **Terminated:** The thread has completed its task or been terminated by another thread.  What is JDBC? [JDBC](https://www.geeksforgeeks.org/introduction-to-jdbc/) standard API is used to link Java applications and relational databases. It provides a collection of classes and interfaces that let programmers to use the Java programming language to communicate with the database. There are generally four components of JDBC by which it interacts with the database:   * JDBC API * JDBC Driver manager * JDBC Test Suite * JDBC-ODBC Bridge Drivers  What are the steps to connect to the database in Java? There are certain steps to connect the database and Java Program as mentioned below:   * Import the Packages * Load the drivers using the forName() method * Register the drivers using DriverManager * Establish a connection using the Connection class object * Create a statement * Execute the query * Close the connections |
|  |