**Q7. What are different ways to create String Object.**

A string object can be created in different ways in Java.

1) Using String Literals

--Stored in the String pool

2) Using the new keyword

--Always creates a new object in heap memory.

3) Using Stringvalue()

--Converts other data types to a string.

4) Using StringBuilder or String Buffer

-- are used to create **mutable (changeable) string object**

**--** when you’re doing a lot of string manipulations (like appending, inserting, deleting).

5) Using String. Format () Method

--Formatting strings dynamically.

6) Using String. Join ()

--Used to join multiple strings with a delimiter (comma, space, dash, etc.)

7) Using concat () Method

--Concatenates string

8) Using char [] Array and byte [] Array

--char [] Array is useful for when you have individual characters you want to convert into a String.

--byte [] Array is useful for handling **binary data** or **encoded text.**

**Q8. How can we make String upper case to lower case?**

### **Using *toLowercase()* Method**

The *toLowercase()* method is a built-in method of the String class. It converts all characters in the String to lowercase.

**Example:**

String str = "HELLO WORLD”;

String lowerStr = str. toLowercase();

System.out.println (lowerStr);

**Output**: hello world

**Q9. How can we make String Lower case to Upper case?**

**Using *toUppercase()* Method**

The *toUppercase()* method of the String class is used to convert all the characters in a string to uppercase.

**EXAMPLE:**

String str = “hello world”;

String upperstr=str.toUppercase();

System.out.println(upperstr);

**Output**: HELLO WORLD

**Q10. What is String subSequence method?**

The Java String *subSequence* method is used to retrive a new character sequence. That is a subsequence of the given sequence. This method behaves like the substring() method. The only difference is that it returns a charsequence rather than a string. It takes two integer arguments:

1) startIndex –This is the value of begin index, inclusive.

2) endIndex --This is the value of the end index, exclusive.

**Example:**

**String mystr=”hello,world!”;**

**System.out.println(mystr.subSequence(7,12));**

**Q11. How to Split String in java?**

Split a String using the *split()* method, which is part of the String class. The *split()* method splits the string into an array of substrings based on a given delimiter or regular expression. If a limit is specified, the returned array will not be longer than the limit.

The string *split()* method take two parameters:

1) regex—Required. A regular expression defining the separators where the string is split.

2) limit—Optional. The maximum length of the returned array.

Syntax:

String[] result=str.split(“regex”)

Example:

1)Split by space-(“ ”)

2)Split by comma(“,”)

3)Split using multiple delimiters—(comma or semicolon)(“[,;]”)

**Syntax:**

**Public String[] split(String regex, int limit);**

**Q35. What is JVM and explain me the Java memory allocation.**

The JVM stands for Java virtual Machine. It is a Virtual runtime environment (JRE) that allows java application to run on any device or operating system. That is also know as “Write Once, Run Anywhere”

1. Compiles and runs java code: Java code(.java) is first compiled into bytecode(.class). This bytecode is platform-independent.
2. The takes that bytecode and executes it, converting it into machine-specific instructions. Memory Management (Garbage collector).
3. Security.
4. Multithreading support.
5. Exception handling

Java Memory Allocation in JVM

1. Heap Memory

* + **Where objects live.**
  + All **class instances and arrays** are stored here.
  + It’s shared across all threads.
  + Managed by **Garbage Collector**.

2. Stack Memory

* Stores **method calls**, local variables, and references.
* Each thread has its **own stack**.
* When a method is called, a **stack frame** is created; it’s removed once the method returns.

3. Method Area

* Stores class-level data: class structure, method data, static variables, etc. Shared among all threads.
* From Java 8 onwards, it's known as Metaspace and is allocated in nativememory (not part of heap).

4. Program Counter (PC) Register

* Each thread has its own PC register.
* It keeps track of **which instruction is being executed** currently.

5. Native Method Stack

* For executing native methods (methods written in languages like C/C++).
* Used when Java interacts with non-Java code through JNI (Java Native Interface).

**JVM Memory =**

├── Heap (objects, arrays)

│ ├── Young Gen (short-lived)

│ └── Old Gen (long-lived)

├── Stack (method calls, local vars)

├── Method Area / Metaspace (class data)

├── PC Register (instruction pointer)

└── Native Method Stack (non-Java code)

**Q36. What is Polymorphism and encapsulation?**

**Polymorphism:**

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance. **Inheritance** inherit attributes and methods from another class.Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways.

For example: Superclass called *Animal* that has a method called *animalSound().* Subclasses of Animals could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound (the pig oinks, and the cat meows, etc.)

**Encapsulation:**

Encapsulation in Java is the mechanism of wrapping data (variables) and methods that work on that data within a single unit (class).A capsule is the perfect example of encapsulation because a capsule is a mixture of several medicines.

**Benefits of encapsulation:**

**Hiding data**: Users will have no idea how classes are being implemented or stored.

**More flexibility**: Enables you to set variables as red or write-only.

**Easy to reuse**: With encapsulation, it's easy to change and adapt to new requirements.

**Q37. What is method overloading and Method over riding?**

In Java, **method overloading** and **method overriding** are two important concepts related to polymorphism (specifically **compile-time** and **run-time** polymorphism).

**Method Overloading:**

Method overloading means **multiple methods in the same class** have the **same name** but **different parameter lists** (different type, number, or order of parameters).

*Key Points:*

* Happens within the same class.
* Return type can be different but it's not enough alone to overload a method.
* Helps in increasing code readability and reusability.

**Method Overriding:**

Method overriding means a **subclass provides a specific implementation** of a method that is **already defined in its superclass.**

*Key Points:*

* Happens between superclass and subclass.
* Method name, return type, and parameter list must be exactly the same.
* The method in the superclass must be inheritable (not private or final).
* It allows dynamic method dispatch at runtime.

**Difference Between Overloading and Overriding:**

| **Feature** | **Method Overloading** | **Method Overriding** |
| --- | --- | --- |
| Class relationship | Same class | Subclass and superclass |
| Method signature | Must differ | Must be exactly same |
| Polymorphism type | Compile-time | Run-time |
| Return type | Can be different | Must be same (or covariant) |
| Access modifiers | No restriction | Can't be more restrictive |

**Q38. Why string is Immutable?**

In Java, the String class is **immutable**, meaning **once a** String **object is created, it cannot be changed**. If you try to modify it, a **new object is created** instead.

Reasons:

**1) Safe to share (Security**)

* Strings are used in sensitive operations like file paths, network connections, class loading, etc.
* If String were mutable, a hacker could modify the contents. (Example: Chage a file path or database URL after it’s been validated).

**2) Memory-efficient (string pool)**

* Java uses a String constant pool to save memory.
* Immutable Strings are safe to share and reuse, which helps with performance and memory optimization.

**3) Thread-safe**

* Since String objects can’t be changed, they are automatically thread safe.
* No need for synchronization, so they’re safe to use in multi-threaded environments.

**4) Reliable as hash keys**

* Strings are often used as **keys in HashMap/HashSet**.

**5) Design Simplicity**

* Reduces unexpected bugs caused by accidental modification.

**Q39. What is the difference between String and String buffer?**

Both String and StringBuffer are used to **handle text** (sequence of characters), but they behave quite differently when it comes to **mutability**, **performance**, and **thread safety**.

| **Feature** | **String** | **StringBuffer** |
| --- | --- | --- |
| **Mutability** | Immutable (cannot change once created) | Mutable (can change contents) |
| **Speed** | Slower (because of immutability) | Faster when doing many changes |
| **Thread Safe** | Yes (immutable) | Yes (synchronaized) |

**Q40. What is the difference between array and array list?**

Both **Array** and **ArrayList** are used to store **multiple values**, but they have some major differences in **functionality**, **flexibility**, and **usage**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Features** | | **Array** | **Arraylist** | |
| Data structure | | Fixed-Length | Variable-length | |
| Size | | Can’t be changed | Can be resize | |
| Data Types | | Can hold both primitives and objects | Can hold only objects | |
| Accessing elements | | Using [] operator | Using a set of Method | |
| Memory allocation | | Less | More | |
| Syntax | | Int [] arr=new int [5] | (ArrayList<Integer>) | |
|  |
|  | | | | |
|  | | | | |
|  | | | | |
| Flexibility | | Less flexible (manual resizing) | More flexible (add/remove dynamically) | |
| Multidimensional  support | | Support  (int [][] matrix=new int [][]) | Not directly support | |

**41. What is the difference between hash map and Hash table?**

**HashMap:**

* A HashMap is a specific implementation of the hash table data structure.
* It stores key-value pairs, just like a hash table.
* Not thread-safe by default (better performance in single-threaded apps)
* 1 null key and multiple null values allowed
* Stores data as Map<Key, Value> pairs
* Does not maintain any order of keys

**Hash Table:**

* A hash table is a data structure that stores key-value pairs.
* It uses a hash function to compute an index (or "hash code") into an array of buckets or slots, from which the desired value can be found.
* Slower and Does not allow null keys/values
* All methods are synchronized — safe for multi-threaded use

**Q42. What is a vector in Java?**

Vector is a dynamic array, meaning its size can grow or shrink as needed. It’s part of the java. Util package and was introduced in JDK 1.0. Vector is similar to ArrayList, but with key differences: it is synchronized, making it thread-safe, and it includes legacy methods not part of the collections framework.

**Vector:**

Vector<Type> vector=new Vector<> ();

**Methods of Vector:**

1) Add Elements to Vector

add(element) -- adds an element to vector

add(index, element) -- adds an element to the specific position

addAll(vector) -- adds all elements of a vector to another vector

2) Access Vector Elements

Get(index) -- returns an element specified by the index

iterator() -- returns an [iterator](https://www.programiz.com/java-programming/iterator) object to sequentially access vector elements

3) Remove Vector Elements

remove(index) -- removes an element from specified position

removeAll() -- removes all the elements

clear() -- removes all elements. It is more efficient than removeall()

**Q43. What is set in java?**

* Set is a collection of elements (or objects) that contains no duplicate elements.
* Set is NOT an ordered collection; its elements does NOT have a particular order.
* Java Set interface is a member of the Java Collections Framework.
* Set DOES NOT allow you to add duplicate elements.
* Set allows you to add at most one null element only.
* Set does NOT support indexes or positions of its elements. You cannot access elements by their index and also search elements in the list.

**Q44. What is an abstract class?**

An abstract class is a class that cannot be instantiated (no objects can be created directly from it) but serves as a blueprint for other classes (subclasses) to inherit from. It's declared using the abstract keyword and can contain abstract methods (methods with no implementation) and concrete methods (methods with a full implementation).

**Q45. What is an interface?**

An interface is a reference type, similar to a class, that can contain only constants, method signatures, default methods, static methods and nested types. Interfaces are used to achieve abstraction and multiple inheritance in Java. A class can implement multiple interfaces, which allows it to inherit the behavior of multiple types.

**Q46. Why Java is Platform independent?**

Java is “platform-independent” because the code written in Java can run on any operating system (Windows, macOS, Linux, etc.) without requiring any changes. This is due to its “Write Once, Run Anywhere”, which is made possible by the Java Virtual Machine (JVM).

## **Java Programs Work:**

**Writing the Code**. You write a Java program in a .java file.

**Compilation**. The Java compiler (javac) processes the code and generates **bytecode** stored in a .class file.

**Execution**. The JVM reads this bytecode, converts it into platform-specific machine code, and runs the program.

**Q47. What are access modifiers? Give me an example?**

**Access modifiers** in Java are **keywords** that decide **who can access** a class, method, or variable.

public → accessible everywhere (any class , any package)

private → accessible only in the same class

protected → accessible in same package + subclasses (even in other packages)

default → accessible in same package only

**Simple Example**:

Package mypackage;

public class Example {

// Variables with different access levels

public int publicVar = 1; // Can be accessed from anywhere

private int privateVar = 2; // Can be accessed only inside this class

protected int protectedVar = 3; // Can be accessed in same package & subclasses

int defaultVar = 4; // Default (package-private): same package only

// A method that prints all variable values

public void showValues() {

System.out.println("Public: " + publicVar);

System.out.println("Private: " + privateVar);

System.out.println("Protected: " + protectedVar);

System.out.println("Default: " + defaultVar);

}

}

**Q48. What are java exceptions? Give me an example**

An **exception** in Java is an **unexpected event** that occurs during the execution of a program and **disrupts the normal flow** of instructions. When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an exception (throw an error).

## **Java try and catch**

* The try statement allows you to define a block of code to be tested for errors while it is being executed.
* The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.
* The finally statement lets you execute code, after try...catch, regardless of the result.

The try and catch keywords come in pairs.

**Syntax:**

try {

// Block of code to try

}

catch (Exception e) {

// Block of code to handle errors

}

**Consider the following example:**

public class Main {

public static void main(String[] args) {

try {

int[] myNumbers = {1, 2, 3};

System.out.println(myNumbers[10]);

} catch (Exception e) {

System.out.println("Something went wrong.");

} finally {

System.out.println("The 'try catch' is finished.");

}

}

}

**The Output:**

Something went wrong.

The ‘try catch’ is finished.

**Q49. What is the difference between throws and throwable?**

The terms throws and Throwable relate to exception handling in Java, but they serve different purposes.

**Throws:** This keyword is used in a method signature to declare that the method might throw one or more exceptions. It acts as a warning to the caller of the method, indicating that they need to handle these potential exceptions using a try-catch block.

**Throwable:** This is a class in Java, specifically the superclass of all errors and exceptions. It forms the base of the exception hierarchy. Exception and Error classes are subclasses of Throwable. Programmers rarely use Throwable directly in code, but it's crucial for the structure of Java's exception handling mechanism.

**Q50. What is the difference between Error and exception?**

* Errors occur at runtime when the system encounters resource limitations.
* Expections occur both runtime and compile time.

**Errors**

* Occurs at runtime
* Not handled using try-catch in most cases
* Caused by hardware failure, memory issues, system limitations, or bugs in the JVM

*Examples:*

OutOfMemoryError

StackOverflowError

InternalError

**Exception:**

* Exceptions are recoverable
* They can be handled using try-catch blocks
* Represents conditions that a program might want to catch
* Help prevent program crashes when unexpected events happen

Divided into:

1) Checked Exceptions (must handle)

*e.g. IOException, SQLException*

2) Unchecked Exceptions (runtime)

*e.g. NullPointerException, ArrayIndexOutOfBoundsException*

|  |  |  |
| --- | --- | --- |
| **Feature** | **Error** | **Exception** |
| Type | System level issue | Application level issue |
| Can be detected? | Not usually | Yes, with try-catch |
| Recoverable? | Rarely | Often |
| Examples | OutOfMemoryError, StackOverflowError | IOException, NullPointerException |

**Q51. What is the difference between Error, throwable and exception?**

Throwable = Anything you can throw

Exception = You can handle it

Error = System broke, not your fault

**Error**

* Represents serious issues that are not intended to be caught.
* Examples: OutOfMemoryError, StackOverflowError
* Usually caused by the environment or JVM, not the application code.

**Example**

Public class ErrorExample{

Public static void main(String[] args){

Throw new StackOverFlowError(“This a serious error”)

}

}

**Throwable**

* It is the superclass for anything that can be thrown using throw.
* Only objects that are instances of Throwable (or its subclasses) can be thrown or caught.
* It has two direct subclasses:

Exception

Error

**Example:**

**try** {

throw new Throwable(“This is a throwable”);

}catch (Throwable t)

**Exception**

* Represents recoverable problems (e.g., file not found, invalid input).
* Used for regular application errors.

Checked exceptions (must be handled or declared)

Unchecked exceptions (subclass of RuntimeException)

Example:

try{

int result=10/0;

}catch(ArithemeticException e){

System.out.println(“caught exception”+e)

}

**Q52. What are collection APIs, give me an example**

The Java Collection API provides a set of classes and interfaces for storing and manipulating groups of data (like lists, sets, maps, etc).

**List:** Ordered collection, allows duplicates

**Example Implementation:** ArrayList, LinkedList

**Set:** No duplicates allowed

**Example Implementation:** HashSet, TreeSet

**Map:** Key-value pairs, Unorder Collections. Duplicate Values are allowed. Duplicate keys are not allowed.

**Example Implementation:** HashMap ,TreeMap

**Queue: FIFO** Structure. Peek and Poll Method

**Example Implementation:** LinkedList, PriorityQueue

**Example: ArrayList**

public class CollectionExample {

public static void main(String[] args) {

// Create a list

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

// Add elements

names.add("uma");

names.add("Bala");

names.add("Charlie");

// Print all elements

System.out.println("Names in the list:");

for (String name : names) {

System.out.println(name);

}

// Check if list contains "uma"

if (names.contains("uma")) {

System.out.println("uma is in the list!");

}

}

}

**Q53. What is the difference between final and finally?**

final and finally are completely different in Java in terms of usage and purpose.

**final-Keyword**

Used to declare constants, prevent inheritance, or prevent method overriding.

Uses of final:

1)Final Variable- value cannot be changed

2)Final Method- cannot be overridden

3)Final Class- cannot be extended

**Finally-Block**

Used with try-catch to execute code always, whether an exception is thrown or not.

**Q54. Will java support multiple inheritance?**

Java does NOT support multiple inheritance with classes, but it does support multiple inheritance using interfaces.

**Q55. What are the different types of interfaces?**

There are a few different types or styles of interfaces based on their structure and purpose.

1)Normal Interface

2)Functional Interface

3)Marker Interface

4)Tagged Interface

5)Hybrid Interface

6)Static and Default Method Interface

**Q56. What are wrapper class? Give me an example**

Wrapper Class is a class that wraps (encapsulates) a primitive data type into an object.

Java is an object-oriented language, but primitive types (int, char, float, etc.) are not objects. So, to use them where objects are required (like in collections), Java provides wrapper classes.

**Example:**

**Primitive Type:** byte, short, int ,long, float

**Wrapper class:** Byte, Short, Integer, Long, Float

public class WrapperExample {

public static void main(String[] args) {

int num = 10;

// Manually wrapping primitive into object

Integer obj = Integer.valueOf(num); // Boxing

// Automatically converting primitive to object

Integer autoBoxed = num; // Auto-boxing

// Automatically converting object to primitive

int unboxed = autoBoxed; // Auto-unboxing

// Using in collections

ArrayList<Integer> list = new ArrayList<>();

list.add(25); // Auto-boxed

list.add(30);

System.out.println("List: " + list);

System.out.println("Unboxed value: " + unboxed);

}

}

**Q57. What is boxing and unboxing in Java? Explain with an example**

Boxing and Unboxing refer to the automatic conversion between primitive types (like int, double, char) and their corresponding wrapper classes (Integer, Double, Character, etc).

**Boxing** (Auto-boxing)

Boxing is when Java converts a primitive into its corresponding wrapper class automatically.

Example:

Int num=10;

Integer boxedNum=num//auto boxing int->Integer

**Unboxing**

Unboxing is when Java converts a wrapper class back to its primitive type automatically.

Example:

Integer boxed =20;

Int unboxed=boxed;//auto boxing Integer->int

**Q58. Explain for each loop**

**for-each loop:**

The for-each loop (also known as the enhanced for loop) is used to iterate over elements in arrays or collections like ArrayList, HashSet, etc., without needing to use an index.

**Syntax:**

For(datatype element: collectionOrArray)

{

//element

}

**Example**

Public class ForEachArrayExample {

Public static void main(String[] args){

Int[] numbers = {10, 20, 30 ,40};

For(int num : numbers){

System.out.println(num);

}

**Q59. What are iterators, explain with an example**

**Iterator:**

An Iterator is an object in Java that allows you to traverse through a collection (like ArrayList, HashSet, etc.) one element at a time. It is part of the java.util package.

**When to use Iterator:**

When you need to remove elements safely while looping

When you want more control than for-each

Works with all Collection types

**Key Methods in Iterator:**

hasNext() –Returns true if there are more elements

next() –Returns the next element

remove() –Removes the last returned element

**Example: Remove elements while iterating**

public class RemoveWithIterator {

public static void main(String[] args) {

List<Integer> numbers = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5));

Iterator<Integer> it = numbers.iterator();

while (it.hasNext()) {

int num = it.next();

if (num % 2 == 0) {

it.remove(); // Safely remove even numbers

}

}

System.out.println("List after removing even numbers: " + numbers);

}

}

**Q 61. What is multithreading, serialization and Generics in Java**

**Multithreading** is the ability of a program to execute multiple threads simultaneously, allowing concurrent execution of two or more parts of a program for maximum CPU utilization.

Each thread runs independently but shares the same memory.

**Serialization** is the process of converting a Java object into a byte stream, so it can be:

Saved to a file

Sent over a network

The reverse process is called Deserialization.

**Generics** allow you to write code that works with any data type, providing type safety and avoiding casting.

**Benefits:**

Compile-time type checking

No need for casting

Reusable code

**What is Polymorphism in Java?**

Polymorphism means "many forms."

In Java (and object-oriented programming), it allows one thing to behave in different ways depending on the context.

👉 It allows us to:

Use one method name to perform different tasks.

Write flexible and reusable code.

🧠 Types of Polymorphism in Java

Type Description Also Known As

Compile-time Polymorphism Method overloading (same method name, different params) Static Polymorphism

Runtime Polymorphism Method overriding (subclass changes parent method behavior) Dynamic Polymorphism