**1. Basics of Java**

* **What are the features of Java?** Simple, Object-Oriented, Platform-Independent, Secure, Robust, Multithreaded, High Performance, Distributed, Dynamic.
* **What is the difference between JDK, JRE, and JVM?**
  + **JDK (Java Development Kit):** For developing and running Java applications (includes JRE + development tools like compiler).
  + **JRE (Java Runtime Environment):** For running Java applications (includes JVM + libraries).
  + **JVM (Java Virtual Machine):** An abstract machine that executes Java bytecode.
* **Why is Java platform-independent?** Java code is compiled into bytecode, which can then be executed on any platform that has a JRE, without recompilation ("Write once, run anywhere").
* **What is bytecode in Java?** An intermediate, platform-independent code generated by the Java compiler from source code, which is then executed by the JVM.
* **What is the difference between == and .equals()?**
  + ==: Compares reference equality (checks if two variables point to the same object in memory).
  + .equals(): Compares content equality (checks if two objects have the same value, as defined by the class's equals() implementation).
* **What is the default value of an uninitialized variable?**
  + **Instance/Static variables:** Numeric types (0, 0.0), boolean (false), object references (null).
  + **Local variables:** No default value; must be explicitly initialized before use.

**2. OOPS Concepts**

* **What are the four principles of OOPS?** Abstraction, Encapsulation, Inheritance, Polymorphism.
* **What is the difference between *abstraction* and *encapsulation*?**
  + **Abstraction:** Hiding complex implementation details and showing only essential features (what an object *does*).
  + **Encapsulation:** Binding data (attributes) and methods (behaviors) that operate on the data into a single unit (class), and restricting direct access to some components.
* **What is *method overloading* and *method overriding*?**
  + **Overloading:** Multiple methods in the same class with the same name but different parameters (compile-time polymorphism).
  + **Overriding:** A subclass provides a specific implementation for a method that is already defined in its superclass (run-time polymorphism).
* **Can we override a static method in Java?** No, static methods belong to the class, not an object, and cannot be overridden. You can "hide" a static method in a subclass.
* **What is *polymorphism* in Java?** The ability of an object to take on many forms; allows objects of different classes to be treated as objects of a common type (e.g., a superclass or interface).
* \**What is an interface? How is it different from an abstract class?*
  + **Interface:** A blueprint of a class, defining a contract of methods that a class must implement. Contains only abstract methods (before Java 8), default, and static methods (Java 8+).
  + **Abstract Class:** A class that cannot be instantiated and may contain abstract methods (requiring subclasses to implement them) as well as concrete methods.
  + **Differences:** Interface supports multiple inheritance; abstract class does not. Interface methods are implicitly public and abstract (pre-Java 8); abstract class methods can have any access modifier.
* **What is the *diamond problem*, and how does Java solve it?**
  + **Diamond Problem:** Occurs when a class inherits from two classes that have a common ancestor, leading to ambiguity if both parent classes implement a method from the common ancestor.
  + **Java's Solution:** Java does not allow multiple inheritance of classes (only single inheritance) thus avoiding the diamond problem for class hierarchies. It allows multiple inheritance of interfaces.

**3. Exception Handling**

* **What is the difference between *checked* and *unchecked* exceptions?**
  + **Checked Exceptions:** Checked at compile-time (e.g., IOException, SQLException). Must be handled using try-catch or declared with throws.
  + **Unchecked Exceptions (Runtime Exceptions):** Not checked at compile-time (e.g., NullPointerException, ArrayIndexOutOfBoundsException). Often indicate programming errors.
* **What is the purpose of the finally block?** Ensures that a block of code is always executed, regardless of whether an exception occurred or not (e.g., for resource cleanup).
* **Can we write a try block without catch or finally?** No, a try block must be followed by at least one catch block or a finally block (or both).
* **What is the difference between throw and throws?**
  + throw: Used *inside* a method to explicitly throw an instance of an exception.
  + throws: Used in a method signature to declare that the method *might* throw one or more specified checked exceptions.
* **What is a custom exception and how do you create it?** A user-defined exception class. You create it by extending Exception (for checked) or RuntimeException (for unchecked).

**4. Collections Framework**

* \*\*What is the difference between \*List, \**Set, and Map?*
  + **List:** Ordered collection of elements, allows duplicates (e.g., ArrayList, LinkedList).
  + **Set:** Unordered collection of unique elements, does not allow duplicates (e.g., HashSet, TreeSet).
  + **Map:** Stores key-value pairs, where keys are unique (e.g., HashMap, TreeMap).
* **What is the difference between ArrayList and LinkedList?**
  + **ArrayList:** Implements a dynamic array. Good for random access, poor for insertions/deletions in the middle.
  + **LinkedList:** Implements a doubly linked list. Good for insertions/deletions anywhere, poor for random access.
* **How does HashSet work internally?** Uses a HashMap internally where elements are stored as keys and a dummy Object is stored as values. Elements are stored based on their hashCode() and equals() methods.
* **What is the difference between HashMap and Hashtable?**
  + **HashMap:** Non-synchronized (not thread-safe), allows one null key and multiple null values, generally faster.
  + **Hashtable:** Synchronized (thread-safe), does not allow null keys or values, generally slower.
* **What is the difference between HashMap and TreeMap?**
  + **HashMap:** Stores elements in an unordered fashion, based on hash codes. Faster for insertion, deletion, and retrieval.
  + **TreeMap:** Stores elements in a sorted order based on keys (natural ordering or custom comparator). Slower than HashMap but provides ordered iteration.
* **What is the significance of hashCode() and equals() in HashMap?** They are crucial for correct storage and retrieval of elements. hashCode() is used to determine the bucket location, and equals() is used to resolve collisions within a bucket and ensure uniqueness.
* **What is *fail-fast* and *fail-safe* in Java collections?**
  + **Fail-Fast:** Iterators throw ConcurrentModificationException if the underlying collection is structurally modified during iteration (e.g., ArrayList, HashMap).
  + **Fail-Safe:** Iterators do not throw ConcurrentModificationException because they operate on a copy of the collection (e.g., CopyOnWriteArrayList, ConcurrentHashMap).

**5. Multithreading & Concurrency**

* **What is the difference between Runnable and Thread?**
  + **Runnable:** A functional interface that represents a task to be executed by a thread. Does not create a new thread itself.
  + **Thread:** A class that provides methods to create and control threads. Can directly create a new thread by extending it or by passing a Runnable object.
* **What is synchronization? How is it achieved?**
  + **Synchronization:** A mechanism to control access to shared resources by multiple threads, preventing data inconsistency.
  + **Achieved by:** Using the synchronized keyword for methods or blocks, ReentrantLock, Semaphore, etc.
* **What is a deadlock and how can you prevent it?**
  + **Deadlock:** A situation where two or more threads are blocked indefinitely, waiting for each other to release a resource.
  + **Prevention:** Avoid circular wait, acquire locks in a consistent order, use tryLock(), set timeouts.
* **What are wait(), notify(), and notifyAll()?** Methods of the Object class used for inter-thread communication.
  + wait(): Causes the current thread to wait until another thread invokes notify() or notifyAll() for this object. Releases the lock.
  + notify(): Wakes up a single thread that is waiting on this object's monitor.
  + notifyAll(): Wakes up all threads that are waiting on this object's monitor.
* **What is the difference between synchronized block and method?**
  + **Synchronized Method:** Synchronizes on the current instance's monitor (for non-static methods) or the class's monitor (for static methods). Locks the entire method.
  + **Synchronized Block:** Allows finer-grained synchronization by locking on a specific object's monitor, only for a specific section of code.
* **What is the difference between *concurrent* and *parallel* execution?**
  + **Concurrent:** Multiple tasks appear to be running at the same time, possibly by time-slicing on a single core. Deals with managing multiple tasks that are executing over the same period.
  + **Parallel:** Multiple tasks are actually running simultaneously on multiple processor cores at the exact same instant.

**6. Java Keywords & Modifiers**

* **What is the difference between final, finally, and finalize()?**
  + final: Modifier for classes (cannot be subclassed), methods (cannot be overridden), and variables (cannot be reassigned).
  + finally: A block of code in try-catch-finally that is always executed, regardless of exceptions.
  + finalize(): A method called by the Garbage Collector before an object is garbage collected (deprecated and generally not recommended for cleanup).
* **What is the use of the static keyword?** Declares members that belong to the class itself rather than to any specific instance of the class (e.g., static variables, static methods, static blocks).
* **Can a class be both abstract and final?** No. An abstract class is meant to be extended, while a final class cannot be extended. They are contradictory.
* **What is the use of transient, volatile, and synchronized?**
  + transient: Marks a field to indicate that it should not be serialized when an object is written to a persistent storage.
  + volatile: Ensures that a variable's value is always read from main memory and not from a CPU cache, guaranteeing visibility of changes across threads.
  + synchronized: Ensures that only one thread can execute a block or method at a time, preventing race conditions and ensuring data consistency.

**7. String Handling**

* **Why are strings immutable in Java?**
  + **Security:** Prevents malicious changes to string data.
  + **Thread Safety:** Immutable objects are inherently thread-safe.
  + **Performance:** Allows string pooling and caching of hash codes.
  + **Hashing:** Used as keys in collections like HashMap.
* **What is the difference between String, StringBuilder, and StringBuffer?**
  + **String:** Immutable sequence of characters. Every modification creates a new String object.
  + **StringBuilder:** Mutable sequence of characters. Non-synchronized (not thread-safe), generally faster for single-threaded environments.
  + **StringBuffer:** Mutable sequence of characters. Synchronized (thread-safe), generally slower than StringBuilder.
* **What is the string pool in Java?** A special memory area in the heap where String literals are stored. When a String literal is created, the JVM first checks if it exists in the pool; if so, it returns the reference to the existing object.
* **How does intern() work?** Returns a canonical representation for the string object. It checks if the string is already in the string pool. If it is, the reference from the pool is returned. If not, the string is added to the pool, and a reference to the newly added string is returned.

**8. Memory Management & Garbage Collection**

* **How is memory managed in Java?** Primarily through automatic garbage collection, which reclaims memory occupied by objects that are no longer referenced. Developers don't manually allocate/deallocate memory.
* **What are stack and heap memory?**
  + **Stack Memory:** Used for method calls, local variables, and primitive data types. Each thread has its own stack. Memory is allocated/deallocated in a LIFO (Last In, First Out) manner.
  + **Heap Memory:** Used for storing all objects created with new. Shared among all threads. Garbage collection primarily operates on the heap.
* **How does garbage collection work in Java?** Identifies and reclaims memory occupied by objects that are no longer reachable (referenced) by any active part of the program. It involves marking, sweeping, and compacting phases, with various algorithms (e.g., Generational, G1, ZGC).
* **What are strong, weak, soft, and phantom references?** Different types of references that influence how the garbage collector treats objects:
  + **Strong:** Normal references; prevent an object from being garbage collected.
  + **Soft:** Allow an object to be garbage collected only if the JVM runs out of memory (memory-sensitive cache).
  + **Weak:** Do not prevent an object from being garbage collected, even if memory is available (e.g., WeakHashMap).
  + **Phantom:** Do not prevent an object from being garbage collected and are enqueued after an object has been finalized but before its memory is reclaimed (for post-mortem cleanup actions).

**9. Java 8 Features (Commonly Asked Now)**

* **What is a *lambda expression*?** A concise way to represent an anonymous function (a function without a name). It allows writing functional code in a more readable and shorter form.
* **What are *functional interfaces*?** An interface with exactly one abstract method. They are used as the target type for lambda expressions.
* **What is the *Stream API* and its benefits?** A new API for processing collections of objects. It allows for declarative, functional-style operations on streams of data, enabling efficient and parallel processing, often with reduced boilerplate code.
* **What are Optional, map(), filter(), and collect() methods?**
  + **Optional:** A container object that may or may not contain a non-null value. Used to avoid NullPointerException.
  + map(): Transforms each element of a stream into another element.
  + filter(): Selects elements from a stream that match a given predicate.
  + collect(): Performs a mutable reduction operation on the elements of a stream, accumulating them into a collection (e.g., List, Set, Map).
* **Explain the default and static methods in interfaces.**
  + **Default Methods:** Allow adding new methods to interfaces without breaking existing implementations. Provide a default implementation that can be overridden by implementing classes.
  + **Static Methods:** Utility methods related to the interface itself. Cannot be overridden by implementing classes and must be called using the interface name.

**10. Miscellaneous**

* **What is autoboxing and unboxing?**
  + **Autoboxing:** Automatic conversion of a primitive type to its corresponding wrapper class (e.g., int to Integer).
  + **Unboxing:** Automatic conversion of a wrapper class object to its corresponding primitive type (e.g., Integer to int).
* **What is the difference between shallow copy and deep copy?**
  + **Shallow Copy:** Creates a new object, but only copies the references to the original object's fields. Both objects share the same underlying referenced objects.
  + **Deep Copy:** Creates a new object and recursively creates new copies of all referenced objects. The new object is completely independent of the original.
* **How does Java achieve pass-by-value or pass-by-reference?** Java is strictly **pass-by-value**.
  + For **primitives**, the actual value is passed.
  + For **objects**, the *value of the reference* (the memory address) is passed. So, you can modify the object that the reference points to, but you cannot change which object the original reference points to from within the method.
* **What is reflection in Java?** A set of APIs that allows a Java program to examine or modify its own behavior at runtime. It can inspect classes, interfaces, fields, and methods, and dynamically invoke methods or create instances.
* **What are annotations in Java?** Metadata that can be added to source code. They provide information to the compiler, runtime, or other tools, but do not directly affect the program's execution. (e.g., @Override, @Deprecated, custom annotations).

**11. Modern Java (Java 8+) & Functional Programming**

* **What is the Optional class and why is it used?**
  + **Answer:** Optional is a container object that may or may not contain a non-null value. It's used to represent the absence of a value, helping to prevent NullPointerException by forcing developers to explicitly handle the "no value" scenario.
* **Explain the difference between map() and flatMap() in Streams.**
  + **Answer:**
    - map(): Transforms each element of a stream into *one* new element, maintaining the stream's structure.
    - flatMap(): Transforms each element into *a stream of elements* and then flattens these individual streams into a single, merged stream. Useful for scenarios where one input element can produce multiple output elements.
* **What is method reference in Java 8? Give an example.**
  + **Answer:** A compact, readable way to refer to methods without invoking them. It's often used with lambda expressions to simplify code.
  + **Example:** list.forEach(System.out::println); (Equivalent to list.forEach(s -> System.out.println(s));)
* **What is the Collectors class used for in Stream API?**
  + **Answer:** Collectors provides static methods for common terminal operations on streams, such as accumulating elements into collections (toList(), toSet(), toMap()), grouping elements (groupingBy()), or partitioning them.
* **How do peek() and forEach() differ in Streams?**
  + **Answer:**
    - peek(): An *intermediate* operation that allows you to perform an action on each element *as it passes through the stream*, without consuming the stream. Useful for debugging.
    - forEach(): A *terminal* operation that performs an action for each element in the stream, consuming the stream and typically used for side effects.

**12. Spring Boot & Spring Framework (Highly Common)**

* **What is Spring Boot and what are its main advantages?**
  + **Answer:** Spring Boot is a framework built on top of the Spring framework that simplifies the creation of stand-alone, production-ready Spring applications.
  + **Advantages:** Auto-configuration, embedded servers (Tomcat, Jetty), opinionated defaults, no XML configuration, Spring Initializr for quick project setup, Actuator for monitoring.
* **What is Dependency Injection (DI) and how does Spring Boot implement it?**
  + **Answer:** DI is an Inversion of Control (IoC) technique where objects' dependencies are provided to them by an external container (the Spring IoC container) rather than the objects creating them themselves. Spring Boot primarily uses annotation-based DI (@Autowired, @Qualifier, @Inject) and constructor injection.
* **Explain @SpringBootApplication annotation.**
  + **Answer:** A convenience annotation that combines @Configuration, @EnableAutoConfiguration, and @ComponentScan. It's typically placed on the main application class to enable Spring Boot's core features.
* **What are Spring Boot Starters?**
  + **Answer:** A set of convenient dependency descriptors that you can include in your application. They bring in all the necessary transitive dependencies for a particular functionality (e.g., spring-boot-starter-web for web applications, spring-boot-starter-data-jpa for JPA).
* **How do you handle configuration in Spring Boot (e.g., application.properties/application.yml)?**
  + **Answer:** Spring Boot automatically loads configuration from application.properties or application.yml files, allowing externalized configuration for different environments (e.g., application-dev.properties, application-prod.properties).
* **What is Spring @RestController and @Controller? What's the difference?**
  + **Answer:**
    - @Controller: Marks a class as a Spring MVC controller. Typically used with @ResponseBody for REST APIs or for traditional MVC applications returning views.
    - @RestController: A convenience annotation that combines @Controller and @ResponseBody. It indicates that the class is a controller whose methods return domain objects directly, and Spring converts them to JSON/XML responses. Used exclusively for RESTful web services.
* **What are Spring Profiles? How are they used?**
  + **Answer:** A mechanism to provide different bean definitions for different environments. You can activate profiles (e.g., dev, test, prod) using spring.profiles.active property or command-line arguments to load specific configurations.
* **Explain Spring Data JPA and its benefits.**
  + **Answer:** Part of the Spring Data family, it simplifies data access layer development by providing an abstraction over JPA.
  + **Benefits:** Reduces boilerplate code for CRUD operations, provides query methods from method names, supports custom queries, integrates easily with Spring.
* **What is Spring Boot Actuator?**
  + **Answer:** Provides production-ready features to help you monitor and manage your application, such as health checks, metrics, info endpoints, and environment properties.

**13. Microservices Architecture**

* **What are Microservices and what are their advantages/disadvantages?**
  + **Answer:** An architectural style that structures an application as a collection of small, autonomous services, each built around a specific business capability, independently deployable, and communicating via lightweight mechanisms (e.g., HTTP/REST).
  + **Advantages:** Scalability, fault isolation, technology independence, faster development cycles, easier maintenance.
  + **Disadvantages:** Increased complexity (distributed systems), data consistency challenges, operational overhead, inter-service communication issues.
* **What is the role of an API Gateway in a microservices architecture?**
  + **Answer:** A single entry point for all client requests. It handles request routing, composition, protocol translation, authentication, authorization, caching, and rate limiting, offloading these concerns from individual microservices.
* **Explain Service Discovery in microservices.**
  + **Answer:** The process by which microservices dynamically locate and communicate with each other. Since service instances can be created, scaled, and destroyed, their network locations are not fixed. Service discovery typically involves a **Service Registry** (e.g., Eureka, Consul) where services register themselves, and clients query the registry to find available instances.
* **What is a Circuit Breaker pattern and why is it important?**
  + **Answer:** A fault-tolerance pattern that prevents a failing service from causing cascading failures in a distributed system. If a service repeatedly fails, the circuit breaker "trips," preventing further calls to that service for a period, allowing it to recover and preventing resource exhaustion in the calling service. Libraries like Resilience4j or Netflix Hystrix (legacy) implement this.
* **How do microservices communicate with each other?**
  + **Answer:**
    - **Synchronous:** HTTP/REST (e.g., using RestTemplate or WebClient in Spring Boot), gRPC.
    - **Asynchronous:** Message Queues (e.g., Kafka, RabbitMQ) for event-driven communication.
* **What are the challenges of data management in microservices?**
  + **Answer:** Maintaining data consistency across multiple independent databases (distributed transactions), defining data ownership for each service, and handling eventual consistency.

**14. Design Patterns & Principles**

* **Explain the Singleton Design Pattern and how to implement it in Java.**
  + **Answer:** Ensures that a class has only one instance and provides a global point of access to that instance.
  + **Implementation:** Private constructor, static instance variable, and a public static method to return the instance (e.g., lazy initialization, eager initialization, enum-based singleton for thread safety and serialization safety).
* **What is the Factory Design Pattern?**
  + **Answer:** Provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created. It's about "creating objects without exposing the instantiation logic to the client."
* **What is the Observer Design Pattern?**
  + **Answer:** Defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
* **What is SOLID principles in OOP?**
  + **Answer:** A set of five design principles intended to make software designs more understandable, flexible, and maintainable.
    - **S**ingle Responsibility Principle (SRP)
    - **O**pen/Closed Principle (OCP)
    - **L**iskov Substitution Principle (LSP)
    - **I**nterface Segregation Principle (ISP)
    - **D**ependency Inversion Principle (DIP)

**15. Testing in Java**

* **What is JUnit? How is it used for unit testing?**
  + **Answer:** A popular open-source framework for writing and running unit tests in Java. It provides annotations (@Test, @BeforeEach, @AfterEach) and assertion methods to define test cases and verify expected outcomes.
* **What is Mockito and why is it used?**
  + **Answer:** A mocking framework for Java that allows you to create mock objects (test doubles) to isolate the code under test. It's used to simulate the behavior of dependencies (e.g., external services, databases) during unit testing, making tests faster and more reliable.
* **What is integration testing and how does it differ from unit testing?**
  + **Answer:**
    - **Unit Testing:** Tests individual components (units) of an application in isolation to verify they work as expected.
    - **Integration Testing:** Tests the interaction and communication between different components or modules of an application to ensure they work together correctly.

**16. Database & Persistence**

* **What is JDBC? Explain its steps.**
  + **Answer:** Java Database Connectivity. It's a standard Java API for connecting to and interacting with relational databases.
  + **Steps:**
    1. Load the JDBC Driver.
    2. Establish a Connection.
    3. Create a Statement.
    4. Execute a Query.
    5. Process the ResultSet.
    6. Close the Connection.
* **What is Hibernate? Why is it used?**
  + **Answer:** An open-source Object-Relational Mapping (ORM) framework for Java. It maps Java objects to database tables and vice-versa, abstracting away much of the boilerplate JDBC code.
  + **Benefits:** Reduces development time, provides a higher level of abstraction, supports various databases, handles transaction management.
* **Explain the difference between get() and load() methods in Hibernate.**
  + **Answer:**
    1. get(): Immediately fetches the data from the database. Returns null if the object is not found.
    2. load(): Returns a proxy object (lazy loading). It only hits the database when the object's properties are actually accessed. Throws ObjectNotFoundException if the object doesn't exist when accessed.
* **What are the features of java ?**

platform independence (Write Once, Run Anywhere), object-oriented design, simplicity, robustness, and high performance.

* **What are the four pillars of OOP?**

Inheritance, Encapsulation, Polymorphism, Abstraction.

* **Difference between method overloading and overriding?**

Overloading: same method name, different parameters (compile-time).

Overriding: same method signature in subclass (runtime).

* **What is encapsulation and how does Java achieve it?**

Wrapping data and methods; using private fields and public getters/setters.

* What is the difference between == and .equals() in Java?

== compares values for primitives and references for objects.

.equals() compares content for objects (when overridden) and references by default.

* **What are wrapper classes? Why are they used?**

Wrapper classes in Java allow primitive data types (like int, char, boolean) to be treated as objects. They are used to enable functionality like using primitives with collections (which require objects), and methods requiring objects as parameters. Autoboxing and unboxing simplify the conversion between primitives and their wrapper class equivalents.

* **What is the role of final, finally, and finalize()?**

The **final** keyword is used to declare constants, prevent method overriding, and prevent inheritance.

final int MAX = 100;

MAX = 200; // Error

The **finally** block is used in **exception handling** to execute important code such as closing resources. It always runs, whether an exception is thrown or not.

The **finalize()** method was used to perform cleanup actions before an object is garbage collected. – deprecated.

* **What is the difference between String, StringBuilder, and StringBuffer?**

In Java, String, StringBuilder, and StringBuffer are classes used to handle text (sequences of characters), but they differ significantly in **mutability**, **performance**, and **thread safety**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **String** | **StringBuilder** | **StringBuffer** |
| Mutability | Immutable | Mutable | Mutable |
| Thread-safe | Yes (immutable) | ❌ No | ✅ Yes (synchronized) |
| Performance | Slowest | Fastest | Slower than StringBuilder |