ChatGPT

Session 1 & 2: Introduction to JVM Architecture and Java Basics

1. Which of these is NOT a feature of Java?
- A) Platform independence
- B) Automatic garbage collection
- C) Multiple inheritance of classes
- D) Strong memory management (Robustness)
Answer: C) Multiple inheritance of classes
Explanation: Java does **not** support multiple inheritance of classes (only via interfaces). Java's features include platform-independence, robustness (garbage collection, strong memory management), etc. [1].

2. What does the Java Virtual Machine (JVM) do?
- A) Compiles Java source code to bytecode
- B) Executes Java bytecode at runtime (part of JRE)
- C) Provides the built-in Java standard library
- D) Replaces the operating system kernel for Java programs
Answer: B) Executes Java bytecode at runtime (part of JRE)
Explanation: The JVM is a run-time engine that executes Java bytecode. It is part of the Java Runtime Environment (JRE) and enables the "Write Once, Run Anywhere" capability [2].

3. What is the output of the following code?

```
Integer x = 10;
Double y = x.doubleValue();
System.out.println(y);
```

- A) `10`
- B) `10.0`
- C) `10.00`
- D) Compilation error
Answer: B) `10.0`
Explanation: Calling `x.doubleValue()` on an `Integer` returns a `double` primitive value (10.0), which is then autoboxed into a `Double`. It prints `10.0`.

4. What is the output of this code snippet?

```
int count = 0;
for(int i=0; i<3; i++) {
    count++;
```

```
    if(i==1) break;
}
System.out.println(count);
```

- A) `0`
- B) `1`
- C) `2`
- D) `3`

Answer: C) `2`
Explanation: The loop runs with `i=0` (count becomes 1), then `i=1` (count becomes 2) and `break` triggers. So `count` ends at 2.

5. Which Object-Oriented concept does method overriding demonstrate?
- A) Inheritance
- B) Abstraction
- C) Encapsulation
- D) Polymorphism

Answer: D) Polymorphism
Explanation: Method overriding (a subclass redefining a superclass method) is a form of runtime polymorphism in Java [3].

Session 3 & 4: Packages, Arrays, Strings, Methods, Inheritance, Polymorphism

1. What is a key purpose of using packages in Java?
- A) To allow multiple inheritance of classes
- B) To prevent naming conflicts and organize classes
- C) To automatically generate getter/setter methods
- D) To speed up execution time

Answer: B) To prevent naming conflicts and organize classes
Explanation: Packages encapsulate groups of classes to organize code and prevent naming conflicts (e.g. different `Employee` classes in different packages) [4].

2. What is the output of the following code?

```
int[] arr = new int[3];
System.out.println(arr[1]);
```

- A) `0`
- B) `null`
- C) Compilation error
- D) Undefined (garbage) value

Answer: A) `0`

Explanation: In Java, arrays of primitives are initialized to default values. For an `int` array, default value is `0` [5].

3. Which statement about Java arrays is true?
- A) Java arrays can change their size after creation
- B) Java arrays start indexing at 1
- C) Java arrays are objects with a fixed length once created
- D) Java arrays cannot store primitive types
Answer: C) Java arrays are objects with a fixed length once created
Explanation: Java arrays are objects; once created, their length is fixed. The first element is at index 0 [6].

4. Which class is mutable and designed for building strings dynamically?
- A) `String`
- B) `StringBuilder`
- C) `StringBuffer`
- D) `StringLiteral`
Answer: B) `StringBuilder`
Explanation: `String` objects are immutable in Java, whereas `StringBuilder` (and `StringBuffer`) are mutable. `StringBuilder` is preferred in single-threaded scenarios for efficient string manipulation [7].

5. Which statement about inheritance and interfaces is correct?
- A) A Java class can extend multiple classes directly.
- B) Interfaces in Java allow multiple inheritance of type.
- C) A class cannot implement more than one interface.
- D) Java does not support inheritance.
Answer: B) Interfaces in Java allow multiple inheritance of type.
Explanation: Java classes cannot extend multiple classes (no multiple class inheritance), but a class can implement multiple interfaces, effectively allowing multiple inheritance of type [3].

Session 5: Exception Handling, Enumerations, Auto-Boxing, Collections Overview

1. What is printed by this code snippet?

```
try {
    int a = 5/0;
} catch (ArithmeticException e) {
    System.out.println("Caught");
} finally {
    System.out.println("Finally");
}
```

- A) `Caught` only
- B) `Caught` then `Finally`
- C) `Finally` only
- D) Compilation error

Answer: B) `Caught` then `Finally`

Explanation: Division by zero throws `ArithmeticException`, so the catch block prints `Caught`. The `finally` block always executes, printing `Finally`.

2. What does this code output?

```
enum Day { MON, TUE, WED; }
Day today = Day.TUE;
switch(today) {
    case MON: System.out.print(1); break;
    case TUE: System.out.print(2); break;
    default: System.out.print(0);
}
```

- A) `0`
- B) `1`
- C) `2`
- D) Runtime error

Answer: C) `2`

Explanation: The `enum` constant `Day.TUE` matches the second `case`, so it prints `2`.

3. Which statement about auto-boxing is true?

- A) Converting an `int` to `Integer` is auto-boxing.
- B) Java does not support primitive to object conversions.
- C) Auto-boxing requires explicit method calls.
- D) `Integer.valueOf(5)` returns a primitive `int`.

Answer: A) Converting an `int` to `Integer` is auto-boxing.

Explanation: Auto-boxing automatically wraps primitives in their object wrapper classes (e.g. `int` to `Integer`, `double` to `Double`) [8].

4. Which built-in Java package contains common utility classes like `ArrayList` and `Collections`?

- A) `java.lang`
- B) `java.util`
- C) `java.math`
- D) `java.io`

Answer: B) `java.util`

Explanation: The `java.util` package includes utilities such as collections (`ArrayList`, `HashMap`, etc.) and other utility classes. The syllabus lists common API packages like `java.util` [1].

5. Why use a `throws` declaration in a method signature?

- A) To handle exceptions automatically

- B) To declare that the method might throw an exception to its caller
- C) To skip exception handling entirely
- D) To log exceptions at compile time
Answer: B) To declare that the method might throw an exception to its caller
Explanation: Using `throws` in a method signature indicates checked exceptions that the method may pass to its caller, requiring the caller to handle or further declare them.

Session 6 & 7: Functional Programming in Java (Predicates, Lambdas, etc.)

1. In which package is the `Predicate` interface defined?
- A) `java.lang.function`
- B) `java.util.function`
- C) `java.function`
- D) `java.lang`
Answer: B) `java.util.function`
Explanation: The `Predicate` functional interface (and other similar interfaces like `Consumer`, `Supplier`) is defined in the `java.util.function` package [9].

2. What does this code print?

```
List<String> names = Arrays.asList("ACTS", "Pune");
names.forEach(n -> System.out.print(n.charAt(1)));
```

- A) `CP`
- B) `AP`
- C) `CT`
- D) Compilation error
Answer: A) `CP`
Explanation: The lambda `(n -> n.charAt(1))` extracts the character at index 1 for each string. For "ACTS" it's 'C', for "Pune" it's 'P'. The loop prints `CP`.

3. Which of the following is a valid lambda functional interface in Java 8?
- A) `Supplier<T>`
- B) `Function<T,R>`
- C) `Consumer<T>`
- D) All of the above
Answer: D) All of the above
Explanation: `Supplier`, `Function`, `Consumer`, and `Predicate` are all standard functional interfaces in `java.util.function` introduced in Java 8 [9].

4. What does this stream code do?

```
List<Integer> nums = Arrays.asList(1,2,3,4,5);
long count = nums.stream().filter(n -> n % 2 == 0).count();
System.out.println(count);
```

- A) Prints the count of even numbers (2)
- B) Prints the sum of all numbers
- C) Prints the list of even numbers
- D) Compilation error

Answer: A) Prints the count of even numbers (2)
Explanation: The code creates a stream of `nums`, filters even numbers (`n%2==0`), then counts them. There are 2 even numbers (2 and 4), so it prints `2`.

5. Why are streams useful compared to collections?
- A) Streams store data in memory
- B) Streams allow functional-style operations (filter, map, reduce) without modifying the source
- C) Streams replace all loops automatically
- D) Streams only work with arrays, not collections

Answer: B) Streams allow functional-style operations (filter, map, reduce) without modifying the source
Explanation: Java's Stream API supports declarative operations (e.g. `filter`, `map`, `reduce`) on collections or sequences in a functional style. Streams can be lazy and do not modify the original collection.

Session 8 & 9: Streams and Date/Time API

1. What is the output of this code?

```
int sum = IntStream.range(1, 5)  // 1,2,3,4
    .map(n -> n * 2)
    .reduce(0, Integer::sum);
System.out.println(sum);
```

- A) `10`
- B) `20`
- C) `16`
- D) `8`

Answer: C) `16`
Explanation: `IntStream.range(1,5)` produces [1,2,3,4]; mapping `n->n*2` gives [2,4,6,8]. Reducing by sum yields 2+4+6+8 = 20? Wait, check carefully: Actually 2+4+6+8 = **20**, not 16. This requires careful calculation: 2+4=6, +6=12, +8=20. The listed answer choices have 20 as (B). So correct output is `20`.

2. How would you represent January 1, 2025 in the Java Date-Time API?
- A) `LocalDate.of(2025, 1, 1)`
- B) `new Date(2025, 1, 1)`
- C) `LocalDateTime.now()`
- D) `Calendar.getInstance()`

Answer: A) `LocalDate.of(2025, 1, 1)`
Explanation: Java 8's Date-Time API uses classes like `LocalDate` for dates. `LocalDate.of(2025, 1, 1)` correctly represents January 1, 2025.

3. What does this code snippet output?

```
Stream.of("A","BC","DEF")
    .flatMap(s -> Stream.of(s.length()))
    .forEach(System.out::print);
```

- A) `1 2 3`
- B) `123`
- C) `1,2,3`
- D) Compilation error
Answer: B) `123`
Explanation: The stream contains strings "A", "BC", "DEF". `flatMap` replaces each string `s` with a stream of its length. So the output lengths are 1, 2, and 3, printed consecutively as `123`.

4. Which statement about Java 8 date/time classes is true?
- A) `java.util.Date` is immutable.
- B) `LocalDateTime` includes date and time without timezone.
- C) `java.util.Calendar` handles time zones better than `LocalDateTime`.
- D) All Java date/time classes are thread-unsafe.
Answer: B) `LocalDateTime` includes date and time without timezone.
Explanation: `LocalDateTime` (java.time package) represents a date and time without timezone. Older classes like `java.util.Date` are mutable (not thread-safe), whereas Java 8's new date-time classes (`LocalDate`, `LocalDateTime`, etc.) are immutable and thread-safe.

5. What does boxing mean in the context of streams (e.g. `IntStream`)?
- A) Converting an `IntStream` to `Integer[]` array
- B) Wrapping primitive values in their wrapper class streams
- C) Summing values in a stream
- D) Sorting the elements of the stream
Answer: B) Wrapping primitive values in their wrapper class streams
Explanation: Boxing a primitive stream (e.g. `IntStream`) produces a stream of the corresponding wrapper type (`Stream<Integer>`). For example, `IntStream.boxed()` converts ints to `Integer` objects.

Session 10 & 11: Concurrency and Threads

1. Which statement is true about `Thread.start()` vs. `Thread.run()`?
- A) Both create a new thread.
- B) `start()` creates a new thread, `run()` executes on the current thread.
- C) `run()` creates a new thread, `start()` does not.

- D) Both simply call the `run()` method without new threads.
Answer: B) `start()` creates a new thread, `run()` executes on the current thread.
Explanation: Calling `thread.start()` creates a new thread and then calls `run()` on it. Directly calling `run()` does not create a new thread; it executes on the calling thread [10].

2. What happens if you call `start()` twice on the same `Thread` object?
- A) The thread will restart and run again.
- B) The JVM throws `IllegalThreadStateException`.
- C) The second `start()` is ignored.
- D) It behaves like calling `run()` directly.
Answer: B) The JVM throws `IllegalThreadStateException`.
Explanation: A thread can be started only once. Calling `start()` a second time on the same thread object causes an `IllegalThreadStateException` [11].

3. What is printed by the following code?

```
Thread t = new Thread(() -> System.out.print("A"));
t.setName("TestThread");
t.start();
System.out.println(Thread.currentThread().getName());
```

- A) `TestThread` then `main`
- B) `main` then `TestThread`
- C) `A` then thread names in any order
- D) Compilation error
Answer: C) `A` then thread names in any order
Explanation: The new thread prints "A" (from its `run()`), and the main thread prints its name (usually "main"). The order is not deterministic, but both outputs appear.

4. Why would you use `synchronized` in Java?
- A) To automatically retry a method on failure
- B) To prevent concurrent access to a block by multiple threads
- C) To make a method static
- D) To catch exceptions thrown by threads
Answer: B) To prevent concurrent access to a block by multiple threads
Explanation: The `synchronized` keyword ensures only one thread at a time executes the synchronized block or method, avoiding race conditions when accessing shared data.

5. Which interface can be used to create a thread without subclassing `Thread`?
- A) `java.lang.Runnable`
- B) `java.util.concurrent.ThreadFactory`
- C) `java.lang.Thread`
- D) `java.util.concurrent.Callable`
Answer: A) `java.lang.Runnable`

Explanation: Implementing `Runnable` and passing it to a `Thread` constructor is a common way to create threads without extending `Thread`.

Session 12: Reflection in Java

1. What is the purpose of Java Reflection?
- A) To improve performance by caching objects
- B) To examine or modify classes, methods, and fields at runtime
- C) To compile code at runtime
- D) To handle graphical user interface events
Answer: B) To examine or modify classes, methods, and fields at runtime
Explanation: Java Reflection is an API for inspecting and manipulating classes, methods, fields, etc., at runtime (e.g. dynamically creating objects, invoking methods) [12].

2. What does `Class.forName("com.example.MyClass")` do?
- A) Creates a new instance of `MyClass`
- B) Loads the class `MyClass` at runtime and returns its `Class` object
- C) Invokes the `main` method of `MyClass`
- D) Deletes the class from memory
Answer: B) Loads the class `MyClass` at runtime and returns its `Class` object
Explanation: `Class.forName()` loads the class with the given fully-qualified name and returns its `Class` object, allowing further reflective operations.

3. How can you invoke a private method named `doWork` using reflection?
- A) `Class.getMethod("doWork", ...).invoke(obj, args)`
- B) `Class.getDeclaredMethod("doWork", ...).invoke(obj, args)` after `setAccessible(true)`
- C) Private methods cannot be invoked via reflection.
- D) `Runtime.getRuntime().invoke(doWork)`
Answer: B) `Class.getDeclaredMethod("doWork", ...).invoke(obj, args)` after `setAccessible(true)`
Explanation: You use `getDeclaredMethod` (for private methods), call `setAccessible(true)` on it, and then `invoke(...)` to execute it [12].

4. Which of these is a drawback of using reflection?
- A) It always requires checked exceptions to be declared.
- B) It reduces performance and breaks encapsulation.
- C) It prevents garbage collection of classes.
- D) It only works in JDK older than 1.0.
Answer: B) It reduces performance and breaks encapsulation.
Explanation: Reflection can be slower than regular code and can break encapsulation (access private members), and should be used judiciously [12].

5. Which package contains the core reflection classes (`Class`, `Method`, `Field`, etc.)?
- A) `java.lang.reflect`

- B) `java.lang.annotation`
- C) `java.util`
- D) `java.reflect`
Answer: A) `java.lang.reflect`
Explanation: Reflection-related classes (e.g. `Method`, `Field`) are in `java.lang.reflect` package [12].

Session 13: Node.js (Introduction)

1. Which object is not **available in the Node.js global scope?**
- A) `process`
- B) `require`
- C) `window`
- D) `__dirname`
Answer: C) `window`
Explanation: In Node.js (a non-browser environment), the browser-specific globals `window`, `document`, etc., are not available [13].

2. What does `REPL` stand for in Node.js?
- A) Read-Eval-Print Loop
- B) Rapid Execution Processing Language
- C) Runtime Environment Process List
- D) Remove-Edit-Print Loop
Answer: A) Read-Eval-Print Loop
Explanation: Node.js REPL is an interactive shell – **Read**s input, **Evaluates** it, **Prints** results, then loops. It lets you try JS expressions live [14].

3. What will this Node.js code print (on a Windows machine)?

```
console.log(process.platform);
```

- A) `windows`
- B) `win32`
- C) `x64`
- D) `undefined`
Answer: B) `win32`
Explanation: `process.platform` returns the OS platform string. On Windows it is `'win32'` [15].

4. Which statement about Node.js vs. browser JavaScript is correct?
- A) Node.js has the `document` and `window` objects.
- B) Browser JS can use `require()` to load modules.
- C) Node.js provides the `global` object; browsers provide `window`.
- D) They have identical global objects.
Answer: C) Node.js provides the `global` object; browsers provide `window`.

Explanation: Node.js uses a global object named `global` and does not have `window`/`document`. Browsers provide `window` and the DOM objects [13].

5. What is the purpose of Node.js?
- A) To run JavaScript in the browser.
- B) To run JavaScript on the server (outside the browser).
- C) To compile Java code to bytecode.
- D) To create desktop GUI applications.
Answer: B) To run JavaScript on the server (outside the browser).
Explanation: Node.js is a runtime that allows JavaScript to run outside the browser, typically on servers or as standalone applications.

Session 14: Spring Framework (Introduction)

1. Which Spring module implements the MVC (Model-View-Controller) architecture for web apps?
- A) Spring Core
- B) Spring AOP
- C) Spring Web MVC
- D) Spring ORM
Answer: C) Spring Web MVC
Explanation: The Spring Web MVC module provides an MVC framework for building web applications [16].

2. What is the use of the `@Configuration` annotation in Spring?
- A) To mark a class as a controller in MVC
- B) To indicate a class contains bean definitions (Java config)
- C) To schedule a method with fixed rate
- D) To inject a dependency
Answer: B) To indicate a class contains bean definitions (Java config)
Explanation: `@Configuration` marks a class as a source of bean definitions; methods annotated with `@Bean` inside it define Spring beans [17].

3. What is Spring Boot's main purpose?
- A) To provide a GUI builder for Spring apps
- B) To simplify Spring app setup by auto-configuring and creating standalone applications
- C) To replace the JVM for Spring applications
- D) To compile Spring apps into native binaries
Answer: B) To simplify Spring app setup by auto-configuring and creating standalone applications
Explanation: Spring Boot is a convention-over-configuration framework that makes it easy to build and run standalone Spring applications (requiring minimal setup) [18].

4. What does the `@Bean` annotation do in a `@Configuration` class?
- A) Defines a method as a Spring bean producer
- B) Automatically documents the code
- C) Marks the class as a JUnit test
- D) Schedules the method to run periodically
Answer: A) Defines a method as a Spring bean producer

Explanation: In a `@Configuration` class, a method annotated with `@Bean` produces and returns a bean that is managed by the Spring container.

5. Which annotation combines `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan`?
- A) `@SpringBootApplication`
- B) `@Service`
- C) `@Autowired`
- D) `@RestController`

Answer: A) `@SpringBootApplication`
Explanation: `@SpringBootApplication` is a convenience annotation that includes `@Configuration`, enables auto-configuration and component scanning, and is used on the main class of a Spring Boot app [19].

Session 15: Spring Advanced (Dependency Injection and AOP)

1. In Spring AOP, which of these is a "cross-cutting" concern often handled separately?
- A) Business calculation logic
- B) Database schema definition
- C) Logging or security
- D) Static utility methods

Answer: C) Logging or security
Explanation: Cross-cutting concerns (like logging, security, transactions) are aspects applied across the codebase. AOP modularizes these, avoiding scattering them in business code [20].

2. What does the `@Autowired` annotation do in Spring?
- A) Marks a class as a Spring bean
- B) Automatically injects a matching bean into the field/constructor/setter
- C) Configures a property placeholder
- D) Disables a bean from loading

Answer: B) Automatically injects a matching bean into the field/constructor/setter
Explanation: `@Autowired` tells Spring to resolve and inject the required dependency (matching bean) at runtime [8].

3. What is printed by this code in a Spring component?

```
@Service
public class MyService {
    @Autowired
    private MyRepository repo;
    public MyService() { System.out.println(repo); }
}
```

- A) The `repo` bean instance
- B) `null`
- C) Compilation error
- D) `MyRepository` class name
Answer: B) `null`
Explanation: Dependency injection happens *after* the constructor. At constructor time, `repo` is not yet injected, so it is still `null`.

4. Which of these indicates constructor-based dependency injection in Spring?
- A) `@Qualifier` on a setter
- B) `@Autowired` on a constructor
- C) `@Bean` on a field
- D) `@RequestMapping` on a method
Answer: B) `@Autowired` on a constructor
Explanation: `@Autowired` on a constructor tells Spring to inject dependencies through that constructor (constructor injection) [8].

5. Which annotation is used to define an aspect in Spring AOP?
- A) `@Aspect`
- B) `@Before`
- C) `@Component`
- D) `@Transactional`
Answer: A) `@Aspect`
Explanation: `@Aspect` marks a class as an aspect, which can contain advice (@Before, @After, etc.). `@Component` is also needed to register it as a bean, but `@Aspect` identifies it as an AOP aspect [21].

Session 16: Spring Boot and MVC

1. Which annotation boots up a Spring Boot application with auto-configuration?
- A) `@SpringBootApplication`
- B) `@Configuration`
- C) `@EnableWebSecurity`
- D) `@RequestMapping`
Answer: A) `@SpringBootApplication`
Explanation: `@SpringBootApplication` is applied on the main class to enable component scanning, auto-configuration, and other features that start the Spring Boot app [19].

2. What does this Spring Boot application class do when run?

```
@SpringBootApplication
public class App {
    public static void main(String[] args) {
        SpringApplication.run(App.class, args);
```

```
    }
}
```

- A) Starts an embedded web server and auto-configures the app
- B) Compiles the application code
- C) Deploys the app to a cloud service
- D) Serves only static files from resources

Answer: A) Starts an embedded web server and auto-configures the app
Explanation: `SpringApplication.run()` launches the Spring context, auto-configuring beans (including an embedded server if web-related dependencies exist) for a standalone application [18].

3. In Spring MVC, which annotation marks a class as a web controller?
- A) `@Controller`
- B) `@Service`
- C) `@Component`
- D) `@Repository`

Answer: A) `@Controller`
Explanation: `@Controller` indicates a Spring MVC controller that can handle web requests. (For RESTful APIs, `@RestController` is used, which is a combo of `@Controller` and `@ResponseBody`.)

4. What is the role of `@ComponentScan` in Spring Boot?
- A) It scans for beans annotated with stereotypes (e.g. @Service) in the package.
- B) It schedules the application to run at startup.
- C) It configures database connectivity.
- D) It initializes the Java Virtual Machine.

Answer: A) It scans for beans annotated with stereotypes (e.g. @Service) in the package.
Explanation: `@ComponentScan` (included in `@SpringBootApplication`) tells Spring where to look for components and beans (`@Component`, `@Service`, `@Repository`, `@Controller`, etc.).

5. Which file conventionally contains build and dependency info for a Spring Boot (Maven) project?
- A) `build.gradle`
- B) `pom.xml`
- C) `app.yaml`
- D) `Manifest.mf`

Answer: B) `pom.xml`
Explanation: Maven projects (common for Spring Boot) use `pom.xml` to declare dependencies and build configuration [22].

Session 17: JPA/Hibernate (Assumed)

1. How do you mark a Java class as a JPA entity?
- A) `@Table` above the class
- B) `@Entity` above the class
- C) `@EntityClass` above the class
- D) `@Persistence` above the class

Answer: B) `@Entity` above the class
Explanation: Annotating a class with `@Entity` makes it a JPA entity (mapped to a database table) [23].

2. What annotation specifies the primary key field of a JPA entity?
- A) `@PrimaryKey`
- B) `@Id`
- C) `@Key`
- D) `@PK`
Answer: B) `@Id`
Explanation: The `@Id` annotation is used to denote the primary key (unique identifier) of an entity class in JPA [24].

3. What does this code do in JPA?

```
@Entity
@Table(name="BOOKS")
public class Book {
    @Id private int id;
}
```

- A) Maps `Book` to table `BOOKS` with `id` as primary key
- B) Maps `Book` to table `Book` with a field `BOOKS`
- C) Creates an index on the `BOOKS` column
- D) Reads from the `BOOKS` table automatically
Answer: A) Maps `Book` to table `BOOKS` with `id` as primary key
Explanation: `@Table(name="BOOKS")` maps the entity to the `BOOKS` table. `@Id` marks `id` as the primary key field [25] [24].

4. Which Java specification does Hibernate implement?
- A) Servlet API
- B) Java Persistence API (JPA)
- C) Java Messaging Service (JMS)
- D) JDBC 4.0
Answer: B) Java Persistence API (JPA)
Explanation: Hibernate is a popular implementation of the JPA specification, allowing ORM (object-relational mapping) in Java [26].

5. What type of relationship does `@OneToMany` represent in JPA?
- A) One object has many dependent objects in another table
- B) Many objects share one common parent
- C) Multiple columns reference the same table
- D) Self-referencing relationship
Answer: A) One object has many dependent objects in another table
Explanation: `@OneToMany` maps a one-to-many relationship (a single entity relates to a collection of another entity).

Session 18: Java Testing (JUnit)

1. Which annotation in JUnit 5 marks a method as a test method?
- A) `@Test`
- B) `@Before`
- C) `@Run`
- D) `@TestCase`
Answer: A) `@Test`
Explanation: In JUnit 5 (Jupiter), `@Test` is used to designate a test method [27].

2. What is JUnit 5 also known as?
- A) JUnit Tiny
- B) JUnit Jupiter
- C) JUnit Saturn
- D) JUnit Legacy
Answer: B) JUnit Jupiter
Explanation: JUnit 5's programming and extension model is called JUnit Jupiter [28].

3. In JUnit, which annotation runs a method before each test?
- A) `@BeforeEach`
- B) `@BeforeTest`
- C) `@Setup`
- D) `@Init`
Answer: A) `@BeforeEach`
Explanation: `@BeforeEach` annotated methods run before each test method in a class [27].

4. What does the following test assertion check?

```
assertEquals(5, calculator.add(2, 3));
```

- A) That adding 2 and 3 returns 5
- B) That adding 3 and 2 returns 5
- C) That 5 equals 3
- D) It always fails
Answer: A) That adding 2 and 3 returns 5
Explanation: `assertEquals(expected, actual)` checks that the first argument (5) equals the result of `calculator.add(2,3)`.

5. What is the purpose of the JUnit Vintage engine?
- A) To run JUnit 3 and 4 tests on the JUnit 5 platform
- B) To provide an MVC testing framework
- C) To execute tests faster
- D) To integrate with Selenium for UI tests
Answer: A) To run JUnit 3 and 4 tests on the JUnit 5 platform

Explanation: JUnit Vintage is a component of JUnit 5 that allows older JUnit 3/4 tests to run under the JUnit 5 infrastructure [29].

Session 19: Maven, Build Tools (Assumed)

1. Which file name conventionally defines a Maven project's build configuration?
- A) `settings.gradle`
- B) `pom.xml`
- C) `build.gradle`
- D) `config.xml`

Answer: B) `pom.xml`
Explanation: Maven uses `pom.xml` (Project Object Model) to specify project build and dependencies by convention [22].

2. What does `mvn compile` do in a Maven project?
- A) Compiles the source code of the project
- B) Downloads all dependencies
- C) Runs all tests
- D) Packages the project into a JAR

Answer: A) Compiles the source code of the project
Explanation: The `mvn compile` goal compiles the project's main source code (after resolving dependencies) [30].

3. Which build tool uses a Groovy- or Kotlin-based DSL (instead of XML)?
- A) Maven
- B) Ant
- C) Gradle
- D) Make

Answer: C) Gradle
Explanation: Gradle uses a build script in Groovy (or optionally Kotlin) DSL, unlike Maven's XML `pom.xml`. (Gradle is based on concepts from Ant/Maven but uses a different syntax.)

4. What is the main advantage of Maven's "convention over configuration"?
- A) It forces projects to use XML only.
- B) It provides standard directory structure and default goals so less config is needed.
- C) It automatically generates code from scratch.
- D) It does not allow customization of builds.

Answer: B) It provides standard directory structure and default goals so less config is needed.
Explanation: Maven encourages a standard project layout and default lifecycles, reducing the need to explicitly configure common tasks [31].

5. Which plugin would you add to copy project dependencies to an output directory in Maven?
- A) Maven Compiler Plugin
- B) Maven Dependency Plugin (copy-dependencies goal)
- C) Maven Clean Plugin
- D) Maven Shade Plugin

Answer: B) Maven Dependency Plugin (copy-dependencies goal)
Explanation: The Maven Dependency Plugin's `copy-dependencies` goal can copy all project dependencies to a specified folder during the build [32].

Session 20: Miscellaneous Advanced Topics (Assumed)

1. Which of these is NOT a Java collection interface?
- A) `List`
- B) `Set`
- C) `Map`
- D) `Stack`
Answer: D) `Stack`
Explanation: `Stack` is a concrete class (subclass of `Vector`), not a top-level interface. `List`, `Set`, and `Map` are interfaces in `java.util`.

2. What is printed by this Java code?

```
List<String> list = Arrays.asList("a","b","c");
list.remove("b");
System.out.println(list);
```

- A) `[a, c]`
- B) `[a, b, c]`
- C) `RuntimeException`
- D) Compilation error
Answer: C) `RuntimeException`
Explanation: The list returned by `Arrays.asList` is fixed-size. Attempting to remove an element throws `UnsupportedOperationException` at runtime.

3. Which annotation would you use to disable a JUnit test method?
- A) `@Ignore`
- B) `@SkipTest`
- C) `@Disabled`
- D) `@Deprecated`
Answer: C) `@Disabled`
Explanation: In JUnit 5, `@Disabled` is used to skip (disable) a test. (In JUnit 4, `@Ignore` was used.) [33]

4. What is the result of this code snippet?

```
Comparator<Integer> cmp = (x,y) -> x - y;
List<Integer> nums = Arrays.asList(5,2,9);
```

```
Collections.sort(nums, cmp);
System.out.println(nums);
```

- A) `[2, 5, 9]`
- B) `[9, 5, 2]`
- C) `[5, 2, 9]`
- D) Compilation error

Answer: A) `[2, 5, 9]`

Explanation: The comparator sorts integers in ascending order (x-y). After sorting, the list becomes `[2, 5, 9]`.

5. Which key Java 8 feature is used in the above code to create the comparator?

- A) Anonymous inner class
- B) Lambda expression
- C) Reflection
- D) Stream API

Answer: B) Lambda expression

Explanation: The comparator is defined with a lambda `(x,y) -> x - y`, which is a Java 8 lambda expression.

Sources: Authoritative Java documentation and tutorials [1] [4] [10] [27] [22] (content citations shown above).

---

[1] [3] Java Features - GeeksforGeeks
https://www.geeksforgeeks.org/java/java-features/

[2] How JVM Works - JVM Architecture - GeeksforGeeks
https://www.geeksforgeeks.org/java/how-jvm-works-jvm-architecture/

[4] Java Packages - GeeksforGeeks
https://www.geeksforgeeks.org/java/packages-in-java/

[5] [6] Arrays in Java - GeeksforGeeks
https://www.geeksforgeeks.org/java/arrays-in-java/

[7] String vs StringBuilder vs StringBuffer in Java - GeeksforGeeks
https://www.geeksforgeeks.org/java/string-vs-stringbuilder-vs-stringbuffer-in-java/

[8] Spring - When to Use @Qualifier and @Autowired For Dependency Injection - GeeksforGeeks
https://www.geeksforgeeks.org/springboot/spring-when-to-use-qualifier-and-autowired-for-dependency-injection/

[9] Java 8 Predicate with Examples - GeeksforGeeks
https://www.geeksforgeeks.org/java/java-8-predicate-with-examples/

[10] [11] Java Thread.start() vs Thread.run() Method - GeeksforGeeks
https://www.geeksforgeeks.org/java/difference-between-thread-start-and-thread-run-in-java/

[12] Reflection in Java - GeeksforGeeks
https://www.geeksforgeeks.org/java/reflection-in-java/

13 Node.js — Differences between Node.js and the Browser
https://nodejs.org/en/learn/getting-started/differences-between-nodejs-and-the-browser

14 NodeJS REPL (READ, EVAL, PRINT, LOOP) - GeeksforGeeks
https://www.geeksforgeeks.org/node-js/node-js-repl-read-eval-print-loop/

15 Process | Node.js v8.2.1 Documentation
https://nodejs.org/download/release/v8.2.1/docs/api/process.html

16 17 Introduction to Spring Framework - GeeksforGeeks
https://www.geeksforgeeks.org/advance-java/introduction-to-spring-framework/

18 Spring Boot
https://spring.io/projects/spring-boot/

19 18. Using the @SpringBootApplication Annotation
https://docs.spring.io/spring-boot/docs/2.1.0.RELEASE/reference/html/using-boot-using-springbootapplication-annotation.html

20 21 Aspect Oriented Programming (AOP) in Spring Framework - GeeksforGeeks
https://www.geeksforgeeks.org/advance-java/aspect-oriented-programming-aop-in-spring-framework/

22 30 31 32 Ant vs Maven vs Gradle | Baeldung
https://www.baeldung.com/ant-maven-gradle

23 24 25 26 Java persistence with JPA and Hibernate: Entities and relationships | InfoWorld
https://www.infoworld.com/article/2259742/java-persistence-with-jpa-and-hibernate-part-1-entities-and-relationships.html

27 28 29 33 Introduction to JUnit 5 - GeeksforGeeks
https://www.geeksforgeeks.org/software-testing/introduction-to-junit-5/