**Short Answers based Questions:**

**Question 2:**

**Explain the terms JDK , JVM , JRE**

**Answer:**

**The Java Development Kit (JDK), Java Virtual Machine (JVM), and Java Runtime Environment (JRE) are crucial components in the Java programming ecosystem, each serving a specific purpose. Their significance lies in their roles in the development, compilation, and execution of Java applications.**

**JDK (Java Development Kit):**

Development Tools: The JDK provides a comprehensive set of development tools, including the Java compiler (javac), debugger, and other utilities. These tools are essential for writing, compiling, and debugging Java code.

Libraries and APIs: JDK includes the Java API libraries, which consist of prewritten code and classes that developers can use to perform common tasks without having to write the code from scratch.

Executable Binaries: The JDK includes the JRE, which means it comes with everything needed for both development and execution of Java applications.

**JVM (Java Virtual Machine):**

Platform Independence: JVM plays a crucial role in making Java a platform-independent language. It interprets and executes Java bytecode, allowing Java applications to run on any device or operating system with a compatible JVM.

Memory Management: JVM manages memory, including garbage collection, which helps in automatic memory cleanup and prevents memory leaks.

Security: JVM provides a secure execution environment by enforcing access controls and preventing unauthorized access to system resources.

JVM uses **Class Loader** to load all classes both inbuilt & user defined into the memory, which are available in the form of BYTECODE and uses the **Byte Code verifier** to check if any foreign code entry has taken place since its original inception, thereby implementing security, also the Bytecode verifier enables one to check the type safety, then the **Just In Time Compiler(JIT)** isused to convert the bytecode to the native format.

**JRE (Java Runtime Environment):**

Runtime Environment: JRE is essential for running Java applications. It includes the JVM, libraries, and other components needed to execute Java bytecode.

Deployment: End-users only need the JRE to run Java applications; they don't need the development tools provided by the JDK. This simplifies the deployment process for Java applications.

Web Browsing: The JRE includes a Java plugin that allows Java applets to run in web browsers, enabling the execution of Java-based content on websites.

Overall Significance:

Development Workflow: Developers use the JDK during the development phase to write, compile, and debug Java code.

Distribution and Execution: End-users require the JRE to execute Java applications without needing the development tools provided by the JDK.

Runtime Environment: The JVM provides a runtime environment for executing Java bytecode and ensures that Java applications run consistently across different platforms.

In summary, the JDK, JVM, and JRE work together to enable the development, distribution, and execution of Java applications, making Java a versatile and widely-used programming language.

**===========================================================================**

**Question 3:**

**What is the difference between final, finally and finalize keywords in Java**

**Answer:**

final:

Usage in Java and other programming languages:

In programming languages like Java, the "final" keyword is used to define a constant value that cannot be modified or a class that cannot be extended.

final int constantValue = 10;

final class FinalClass { /\* class definition \*/ }

finally:

Usage in programming (exception handling):

In programming, particularly in languages with exception handling mechanisms (such as Java), the "finally" block is used in a try-catch-finally structure. The code within the "finally" block is guaranteed to execute, whether an exception occurs or not. It is often used for cleanup or resource release.

Example (Java):

try {

// code that may throw an exception

} catch (Exception e) {

// handle the exception

} finally {

// code that always executes, regardless of whether an exception occurred

}

finalize:

In Java, the "finalize" method was historically used for object cleanup and resource release before an object is garbage collected. However, it is considered a bad practice, and starting from Java 9, the "finalize" method has been deprecated. It's generally recommended to use other mechanisms, such as the "AutoCloseable" interface or the try-with-resources statement, for resource management.

Example :

public class MyClass {

// other class members

@Override

protected void finalize() throws Throwable {

// cleanup code before the object is garbage collected

// (deprecated and not recommended)

}

}

**===========================================================================**

**Question 4:**

**Explain What are IOC and DI concepts in spring framework**

**Answer:**

**Dependency Injection (DI):**

Dependency Injection (DI) is a design pattern that allows an object to be dependent on another object and receive its dependencies from an external source, rather than creating them itself. This helps to decouple the objects and make the code more flexible, maintainable, and testable.

With dependency injection, objects are not tightly coupled to their dependencies, but instead they receive their dependencies through a constructor, setter method, or interface. The external source, often called the "injector," is responsible for creating the required objects and passing them to the dependent objects.

Advantages of using Dependency Injection include:

1. Loose Coupling: Dependency injection allows objects to be loosely coupled, making it easier to change or replace components without affecting the rest of the system.

2. Testability: Dependency injection makes it easier to test individual components in isolation, since their dependencies can be mocked or stubbed.

3. Reusability: Objects can be reused in different contexts, since they are not tightly coupled to their dependencies.

4. Flexibility: Dependency injection allows objects to be wired together at runtime, making it easier to change or replace components as needed.

Dependency Injection is a feature , where the dependencies of a class (i.e., the objects it relies on) are injected into the class by an external entity, rather than the class creating or managing its dependencies. This reduces the coupling between components and makes the system more modular, flexible, and testable.

Role in Spring: In Spring, DI is the mechanism by which the Spring IoC container provides the necessary dependencies to a class when it is being created. The dependencies are typically provided through constructor injection, setter injection, or method injection.

**Inversion Of Control:**

Inversion of Control (IoC) is a core principle in the Spring Framework and is used extensively in Spring MVC. IoC refers to a design pattern in which the control of objects and their dependencies is inverted. Instead of the objects themselves managing their own dependencies, the control of the dependencies is handed over to a container, which manages the lifecycle and dependencies of the objects.

In a Spring MVC application, the IoC container is responsible for managing the components and their dependencies. This includes creating and initializing the components, managing their lifecycle, and resolving their dependencies. The components are defined as beans in the configuration of the IoC container, and the container takes care of managing the relationships between the components and their dependencies.

The use of IoC in Spring MVC provides several key benefits, including:

Loose Coupling: By managing the dependencies between components, IoC helps to reduce the coupling between components and makes it easier to change the implementation of a component without affecting other parts of the application.

Flexibility: The IoC container provides a flexible and configurable mechanism for managing components and their dependencies, allowing developers to easily change the configuration of the components and the relationships between them.

Testability: By managing the dependencies between components, IoC makes it easier to test the components in isolation, without having to worry about the dependencies and their state.

Reusability: The IoC container makes it easier to reuse components and their dependencies, by making it simple to configure and manage the relationships between components.

In summary, IoC is a key principle in the Spring Framework and is used extensively in Spring MVC to manage the components and their dependencies in a flexible and configurable manner. The use of IoC in Spring MVC helps to reduce coupling between components, improve testability, and increase reusability.

**===========================================================================**

**Question 5:**

**Explain Checked and Unchecked exceptions in brief along with examples**

**Answer:**

# Checked Exception vs Unchecked Exception:

Checked Exceptions are those exceptions which are detected at the time of compilation but encountered only during the runtime, but UnChecked exceptions are those which are both detected and encountered only at the runtime.

All Exceptions defined under RuntimeException & Error classes are all UnChecked exceptions all other Exceptions are Checked Exceptions.

Which means that activities for example File Handling , Database related tasks would  give rise to Checked exceptions, whenever we write a piece of code for File Handling, Database connectivity or Network related tasks or even any Resource oriented tasks, the Compiler cautions us by providing  compilation error, asking us to guard the block with try catch mechanism, so that in the runtime if the respective resource is failed/not found, it can be handled appropriately, without abrupt halting of the application.

Whereas for the activities like Array Manipulation or Division it is the responsibility of the developer to handle exceptions and the compiler may not give the caution message, which are having the potential to generate UnChecked exceptions.

Checked exceptions are exceptions that are checked at compile-time. This means that the compiler ensures that these exceptions are either caught using a try-catch block or declared in the method's throws clause.

Examples: Examples of checked exceptions include IOException, SQLException, and FileNotFoundException.

Handling: Developers are required to handle checked exceptions either by using a try-catch block to catch the exception or by declaring the exception in the method's throws clause. Failure to do so results in a compilation error.

Unchecked exceptions, also known as runtime exceptions, are exceptions that are not checked at compile-time. They usually indicate programming errors, and the compiler does not enforce handling or declaration of these exceptions.

Examples: Examples of unchecked exceptions include NullPointerException, ArrayIndexOutOfBoundsException, and ArithmeticException.

Handling: While it is not mandatory to handle or declare unchecked exceptions, it is generally a good practice to do so for robust and reliable code. Developers can choose to catch these exceptions or let them propagate up the call stack.

==================================EOD====================================