

**Encapsulation:-** In Java, encapsulation refers to wrapping variables and methods as a single unit. Class variables are hidden away from other classes and can only be accessed through their current class methods. The mechanism of encapsulation is also referred to as 'data hiding' as it hides data from other classes. Encapsulation is a key concept of object-oriented programming, along with abstraction, polymorphism, and inheritance.

**Polymorphism:-** Polymorphism in Java refers to the ability of objects to transform into different forms. Objects which pass upwards of one IS-A test are called polymorphic. Polymorphism enables developers to perform individual actions in varying ways. Programmers can define a single interface and have multiple executions.

**Abstraction:-** Abstraction is a fundamental concept in Object-Oriented Programming (OOP). It means hiding unnecessary details and showing only the essential features of an object. This helps users focus on what an object does rather than how it does it. It Minimizes the complexity of observing things, Increases reusability and code duplication, and Raises application security by hiding details, Multiple related classes can be grouped as siblings.

**INHERITANCE:-** Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system). The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class.

**Heap and Stack Memory Allocation:-** In Java, memory is divided into two main areas: Heap Memory and Stack Memory, each serving distinct purposes. Heap Memory is used for storing objects and static variables. It is shared across all threads in the application and managed by the Garbage Collector, which deallocates memory for objects no longer in use.

On the other hand, Stack Memory is used for method-specific data, such as local variables, method parameters, and references to objects in the heap. It operates in a Last In, First Out (LIFO) manner, with each method call creating a new frame in the stack. When the method execution completes, its frame is automatically removed, ensuring efficient memory management.

**String vs String builder vs String buffer:-** In Java, String, StringBuilder, and StringBuffer are classes used for handling and manipulating sequences of characters, but they differ in terms of mutability, performance, and thread safety. A String is immutable, meaning once created, its value cannot be changed. Any modification creates a new string object, making it less efficient for frequent updates but ideal for fixed or read-only text. Strings are stored in the String Constant Pool, which allows reuse of objects and optimizes memory usage.

**String Constant Pool:-** The String Constant Pool in Java is a special memory area within the Heap where string literals are stored. When a string literal (e.g., "Hello") is created, Java first checks the pool to see if an identical string already exists. If it does, the new reference points to the existing string, avoiding duplication and saving memory. If the string does not exist, it is added to the pool. This mechanism ensures efficient memory usage and enhances performance when working with strings.

**Final and Static Keyword:-** The final and static keywords serve distinct purposes but are often used together for specific use cases. The final keyword is used to impose restrictions. When applied to a variable, it makes the variable a constant, meaning its value cannot be changed once assigned. When used with a method, it prevents the method from being overridden by subclasses, ensuring its behavior remains unchanged. Applied to a class, it prevents the class from being inherited, making it immutable or unextendable. For example, the String class in Java is final to maintain its immutability and security.

**Collection Framework:-** The Collection Framework in Java is a unified architecture that provides a set of interfaces, classes, and algorithms for storing and manipulating groups of data. It is part of the java.util package and includes various classes for working with different types of data structures, such as lists, sets, maps, and queues. The primary goal of the collection framework is to provide a standard way to handle and organize data, making it easier to work with large sets of data efficiently.

**Auto wired annotations:-** The @Autowired annotation in Spring is used to automatically inject dependencies into your classes. Instead of manually creating or setting up objects (dependencies) within a class, Spring does this for you. When you use @Autowired on a field, constructor, or setter method, Spring looks for a matching bean in the Spring container and automatically provides it.

**Bean:-** Bean is simply a Java object that is managed by the Spring Framework. It follows specific conventions, such as having a no-argument constructor, being serializable, and allowing access to its properties via getter and setter methods. In the context of Spring, a bean represents a component or service that is created, configured, and managed by the Spring container. The container takes care of the lifecycle of the bean, such as instantiating it, setting its dependencies, and handling its destruction when no longer needed.

**Configuration:-** configuration refers to the process of setting up and defining the necessary components, properties, and behaviors of an application. In the context of the Spring Framework, configuration typically involves defining how beans (objects) are created, how they interact with each other, and how various settings are applied to the application. This can be done using XML configuration files or more commonly now, Java-based configuration using @Configuration annotations.

## **Spring boot application annotations:-**

**@SpringBootApplication:** This is the main annotation that combines three important annotations: **@Configuration**, **@EnableAutoConfiguration**, and **@ComponentScan**. It marks the entry point of a Spring Boot application, typically used on the main class to launch the application.

**@RestController:** Used in RESTful web services, this annotation combines **@Controller** and **@ResponseBody**, meaning that the class is a controller and the return values of its methods are automatically serialized into the HTTP response body.

**@RequestMapping:** This annotation is used to map HTTP requests to handler methods of MVC controllers. It can be customized with HTTP methods (GET, POST, etc.) and request paths, allowing for flexible routing in web applications.

**@Autowired:** Automatically injects beans into a class. This annotation reduces the need for manual configuration by Spring's dependency injection system, automatically providing required dependencies based on the type.