Other Concepts	
Dependency Injection	Dependency Injection is a design pattern used in software development to manage dependencies between objects. It allows the dependencies of a class to be provided externally, rather than having the class create or manage them internally. This pattern promotes loose coupling and makes the code more modular, testable, and maintainable.
Dependency Injection in Spring	Dependency Injection (DI) is a fundamental concept in the Spring framework, which provides a powerful and flexible way to manage dependencies in a Java application. Spring's DI container, also known as the Spring IoC (Inversion of Control) container, is responsible for instantiating and wiring dependencies for your application.
Lifecycle of a Spring bean	**1. Bean Definition**: In this stage, the bean configuration is defined in either XML or Java-based configuration. It includes specifying the bean class, dependencies, and other properties.
	2. Instantiation: During this stage, the Spring container creates an instance of the bean based on the bean definition. The container uses the bean's constructor or a factory method to create the object.
	3. Dependency Injection: Once the bean is instantiated, the container injects any required dependencies into the bean. This can be done through constructor injection, setter injection, or field injection.
	4. Bean Post-Processing: After dependency injection, Spring applies any registered BeanPostProcessors to modify the bean instance. BeanPostProcessors can perform tasks such as initializing proxy objects or adding additional behavior to the bean.

Other Concepts (cont)

	5. Initialization: At this stage, any initialization logic specified for the bean is executed. This can involve implementing the InitializingBean interface, defining custom initialization methods using annotations, or specifying initialization methods in the bean configuration.
	6. Ready for Use: After initialization, the bean is ready for use. It can now respond to requests and perform its designated tasks.
	7. Usage: During this stage, the bean is actively used by other components or services in the application. It carries out its assigned functionality and can be accessed and manipulated as needed.
	8. Destruction: When the bean is no longer needed or when the application is shutting down, the container triggers the destruction of the bean. This involves executing any defined destruction logic, such as implementing the DisposableBean interface, specifying custom destruction methods using annotations, or defining destruction methods in the bean configuration.
Spring IoC (Inversion of Control) container	The Spring IoC (Inversion of Control) container is a core component of the Spring framework that manages the lifecycle and dependencies of objects (beans) in a Spring application. The IoC container is responsible for creating, configuring, and wiring the beans, allowing developers to focus on writing the business logic of their application.

Basic

Basic (cont)

Package	group of similar classes, interface and sub package.
	java.lang package is imported implicitly(Throwable, Iterable, Comparable, Object).

STRING

Java String	Immutable
	Literals — stored in string constant pool(inside heap)intern()
	Object — stored directly in heap
	When the intern() method is executed then it checks whether the String equals to this String Object is in the pool or not. If it is available, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.
String-Buffer	mutable
	thread safe and synchronized
	less efficient than StringBuilder
String-Builder	mutable
	non-synchronized,i.e., not thread safe
	more efficient than StringBuffer

Object

Every class in Java is directly or indirectly derived from the Object class.

toString() : provides String representation of an Object. The default toString() method for class Object returns a string consisting of class name+@+unsigned hexadecimal representation of the hash code of the object.

hashCode() : For every object, JVM generates a unique number which is hashcode.

equals(Object obj) : Compares the given object to “this” object

finalize() method : This method is called just before an object is garbage collected. It is called by the Garbage Collector on an object when garbage collector determines that there are no more references to the object.

clone() : It returns a new object that is exactly the same as this object

wait(), **notify() notifyAll()** are related to Concurrency.

OOP

Java language and platform	Java language : High level, Platform Independent, Portable
	Java platform : JRE and API
JVM, JRE & JDK	JVM : VM that provide specification for JRE.
	JRE : implementation of JVM where the byte code get executed.
	JDK : JRE + Tools (javac, javadoc, jar).
Static	Static Variable : belong to class and get memory only once in class area at the time of class loading.
	Static Method : belong to class, cant use non static variables and methods inside if it is not known
	Static Block : initialize static variables and executed before main method at the time of class loading.
	Static Import : access any static member of a class directly. There is no need to qualify it by the class name in program.
Access Modifiers	Public > Protected > Default > Private
Final	variable (can't change, constant), method(can't override), class (can't inherit)

Object	real time entity with state and behavior.
Class	collection of similar Objects.
	Constructor : special function used to initialize state of an object. No return Type but returns current instance of the class. Not inherited so cant make final. Called by super() method from child class.Cant have this() and Super() together in constructor as both should be the first statement.
	this : points current object, it is final type, can be used in synchronized block

OOP (cont)

Encapsulation	wrapping data and associated function into single unit implements data hiding using private property accessed using getter and setter methods.
Inheritance	mechanism by which one class acquire properties and behavior of another on class, Code re-usability.
	Super : points to parent class object.
Polymorphism	same message can be processed in more than one form.
	Method Overloading : same function name but differ in number and type of arguments within the same class, Readability , Compile time.
	Method overriding : specific implementation of method in child class which is already defined in defined in parent class, Run time(only method not property).
	Covarient return type : child class method return type should be sub type of return type of parent class.
Abstraction	implementation hiding using Abstract class and Interface.
	Abstract class : cant be instantiated, should have at least one abstract method, can have constructor(called by extended class constructor during object creation).
	Interface : no constructor and instance, public static final members.

OOP (cont)

	Tagged/Marker interface: no members defined, used to give mark/tag. eg: serializable, clonable.
Relation	Association : relationship where all objects have their own life-cycle & there is no ownership. eg: teacher-student
	Aggregation : special type of association, separate life cycle but there is ownership eg : department- teacher
	Composition : Special type of aggregation, no separate life cycle and if parent deleted, all child will get delete.

Collection

Collection	Interface in java.util package extended Iterable interface.
List	maintain insertion order, include duplicate elements, can have null entry.
	ArrayList : dynamic array, index based, best for store and fetch, increases size by half
	LinkedList : doubly linked list, best for adding and removing
Queue	PriorityQueue, Dequeue-ArrayDequeue
	PriorityQueue : min/max heap
Set	no duplicate elements
	HashSet : no order maintained, can have single null
	LinkedHashSet : insertion order maintained, can have single null
	TreeSet : sorted, no null value

Collection (cont)

Map	key value pair, unique key
	HashMap : no order, can have single null key
	LinkedHashMap : insertion order, can have single null key
	TreeMap : sorted based on key, can't have any null key
Collections	java.util.Collections utility class
	Sorting : List by Collections.sort(), Set by converting to TreeSet, Map by converting to TreeMap, Need to implement Comparable or Comparator
	Comparable : lang, compareTo(), change base class, single sort logic
	Comparator : util, compare(), don't change base class, multiple sort logic
	Unmodifiable : unmodifiableCollection() return an unmodifiable view of the specified collection
Legacy Class	all are synchronized and thread safe
	Property, Vector(increase size by double), Stack, HashTable(no null key and null value)
Iteration	Iterator : legacy iteration support, list and set, can remove, forward only
	ListIterator : legacy iteration support, list, can remove and add, forward and backward
	Enumerator : only for legacy support

Collection (cont)

To successfully store and retrieve objects from a Hashtable, the objects used as keys must implement the hashCode method and the equals method. Because hashcode used to find the bucket and equals used to replace existing value in that place of bucket.(if equals not overridden then it insert into a new LinkedList node that it use. It it total violation of rule as key are unique in map)

EXCEPTION HANDLING

Throwable	exception and error
Exception	can be handled using try-catch block or throws.
	checked and unchecked exception
Checked exception	found at compile time.
	ClassNotFoundException, SQLException, IOException
Unchecked exception	occur during run time.
	ArithematicException, NumberFormatException, NullPointerException, Arrray-IndexOutOfBoundException, StringIndexOutOfBoundException
Error	cant be handled
	Irrecovarable
	StackOverflowError
Finally	block after try/catch, always executed (not if program exits)

EXCEPTION HANDLING (cont)

Throw	keyword, within method, followed by instance, single, cant propagate checked exception
Throws	keyword, within method signature, followed by class, multiple, can propagate checked exception
Exception Overriding	if parent method not defined exception, child cant define checked exception but can define unchecked
	else child can define only sub class exception
Try with resource	autoclosable

Concurrency

Fail-fast	immediately throws Concurrentmo-dificationException, if any structural modification occur

Generics

Generic	Generics in programming languages, such as Java, allow the creation of classes, interf-aces, and methods that can work with different types, providing flexibility and type safety. It enables the definition of generic algorithms and data structures that can be used with various types without sacrificing type checking at compile time.
Wildca-rd(?)	Lower-bound <? super type>
	Upper-bound <? type>
	Unbound <?>

Memory

Types	Heap Area, Method Area, Stack, Native Method Stack & PC Register
	You can not force Garbage collection in Java. Though you can request it by calling Sytem.gc() or its cousin Runtime.getRuntime().gc(). It's not guaranteed that GC will run immediately as result of calling these method
Immutable Class Creation	Declare the class as final so it can't be extended.
	Make all fields private so that direct access is not allowed.
	Don't provide setter methods for variables
	Make all mutable fields final so that it's value can be assigned only once.
	Initialize all the fields via a constructor performing deep copy.
	Perform cloning of objects in the getter methods to return a copy rather than returning the actual object reference.
Deep Copy and Shallow Copy	The shallow copy is the approach when we only copy field values and therefore the copy might be dependent on the original object.

Memory (cont)

In the deep copy approach, we make sure that all the objects in the tree are deeply copied, so the copy isn't dependent on any earlier existing object that might ever change.

SOLID Principles

Single Responsibility Principle	The Single Responsibility Principle states that a class should have only one reason to change, meaning it should have only one responsibility or job. In other words, a class should have a single purpose or focus.
Open-Closed Principle (OCP)	The Open-Closed Principle states that software entities (classes, modules, functions, etc.) should be open for extension but closed for modification. In other words, the behavior of a software entity should be easily extendable without modifying its existing code.

SOLID Principles (cont)

Liskov Substitution Principle (LSP):	The Liskov Substitution Principle states that objects of a superclass should be replaceable with objects of its subclasses without affecting the correctness of the program. In other words, a subclass should be able to be used wherever its superclass is expected, without causing any unexpected behavior.
Interface Segregation Principle (ISP):	The Interface Segregation Principle states that clients should not be forced to depend on interfaces they do not use. It suggests that interfaces should be specific to the needs of the clients, and no client should be obligated to depend on methods it does not need.

SOLID Principles (cont)

Dependency Inversion Principle (DIP):	The Dependency Inversion Principle states that **high-level modules should not depend on low-level modules. Instead, both should depend on abstractions**. It also states that abstractions should not depend on details; details should depend on abstractions.