Java

Core Java

* Java 8 Features
* Functional Interfaces
* Lambda Expression
* Stream API
* Method References
* Collection Framework
* OOPS
* Generics
* Exception Handling
* Checked Exception
* Unchecked Exception
* Runtime Exception
* Object Class Methods
* hashCode and equals
* wait, notify and notifyAll
* toString
* getClass
* clone
* finalize
* Keywords
* volatile
* transient
* final
* synchronized

Spring Framework

* Beans
* Singleton
* Prototyoe
* Request
* Session
* Global Session

Spring Boot

* Spring Boot Annotations

@Component: Tells Spring IOC Container to create a Spring Bean for annotated class

Micro services

* Service Discovery
* API Gateway
* Spring Cloud Config Server
* Load Balancing
* Scalability

Micro services architecture Design Patterns

* Decomposition Patterns
* Decompose by Business capability
* Decompose by Sub domain
* Decompose by Transactions
* Strangler Pattern
* Bulkhead Pattern
* Sidecar Pattern
* Integration Patterns
* API Gateway Pattern
* Aggregator Pattern
* Proxy Pattern
* Gateway Routing Pattern
* Chained Microservice Pattern
* Branch Pattern
* Client Side UI Composition Pattern
* Database Patterns
* Database per service
* Shared Database per service
* CQRS
* Event Sourcing
* Saga Pattern
* Observability Patterns
* Log Aggregation
* Performance Metrics
* Distributed Tracing
* Health Check
* Cross Cutting Patterns
* External Configuration
* Service Discovery Pattern
* Circuit Breaker Pattern
* Blue-Green Deployment Pattern

Design Patterns

* Creational
* Singleton
* Prototype
* Factory
* Abstract Factory
* Builder
* Object Pool Pattern
* Structural
* Adapter
* Bridge
* Composite
* Decorator
* Façade
* Flyweight
* Proxy
* Behavioural
* Chain of Responsibility
* Command
* Interpreter
* Iterator
* Mediator
* Memento
* Observer
* State
* Strategy
* Template Method
* Visitor
* J2EE
* Presentation Layer
  + Intercepting Filter Pattern
  + Front Controller Pattern
  + View Helper Pattern
  + Composite View Pattern
* Business Layer
  + Business Delegate Pattern
  + Service Locator
  + Session Façade Pattern
  + Transfer Object Pattern
* Integration Layer
  + Data Access Object / DAO Pattern
  + Web Service Broker Pattern

Principles

* 12 Factor App
* Codebase
* Dependencies
* Config
* Backing Services
* Build, Release, Run
* Processes
* Port Binding
* Concurrency
* Disposability
* Dev/Prod Parity
* Logs
* Admin Processes
* SOLID
* Single Responsibility Principle
* Open Closed Principle
* Liskov Substitution Principle
* Interface Segregation Principle
* Dependency Inversion Principle
* DRY
* KISS

JPA

* Hibernate

Data Structures

Algorithms

Project Architecture

System Design

Caching Techniques

Concurrent Collections

Synchronization

Time Complexity and Space Complexity

Authentication Mechanism

Authorization Mechanism

Collision Resolution Techniques

* Separate Chaining or Open Hashing
* Open Addressing or Closed Hashing

Competitive Programming (Leetcode, Hackerrank)

Java 8 Features

Functional Interface

* Functional Interface is an interface which contains one and only one abstract method but can have multiple default and static methods

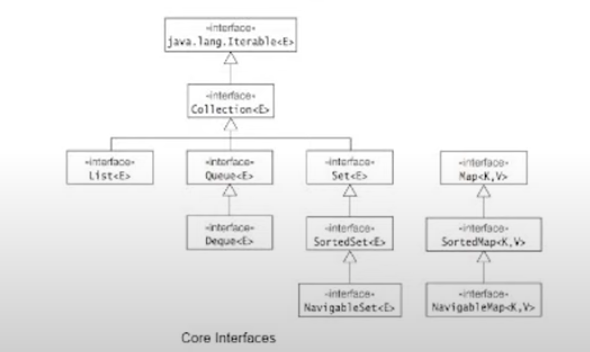
|  |  |
| --- | --- |
| **Interface** | **Single Abstract Method** |
| Runnable | run() |
| Callable | call() |
| Comparable | compareTo() |
| Comparator | compare() |
| Function | apply() |
| Predicate | test() |
| Consumer | accept() |

Lambda Expression

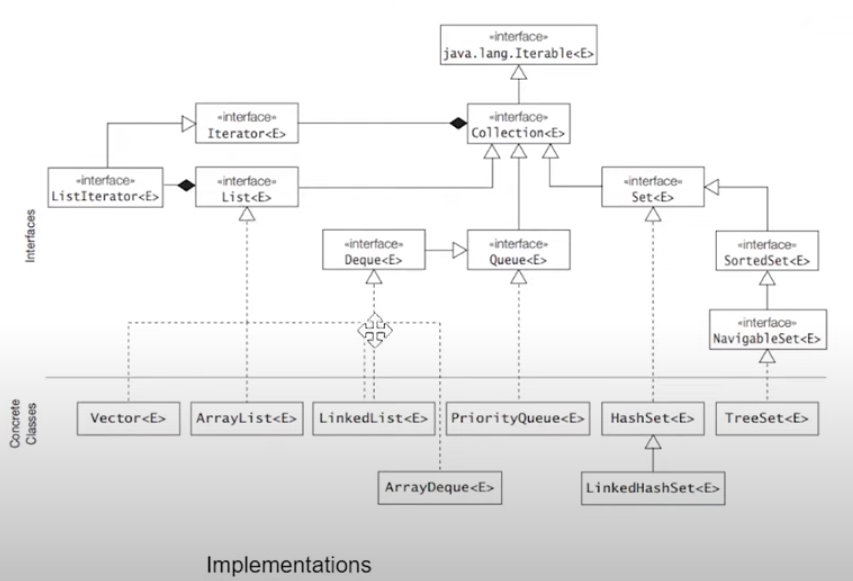
* Lambda Expression is an expression through which we can represent an anonymous function

Collection Framework

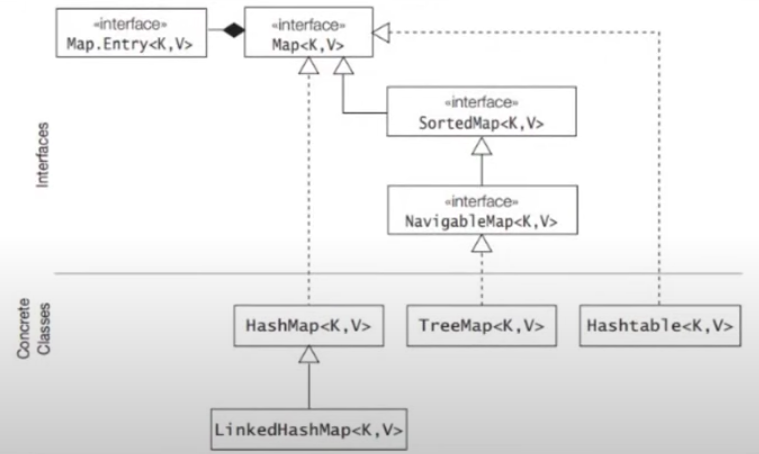
Core Interfaces in Collection Framework



Collection Implementation Classes



Map Interface Implementations

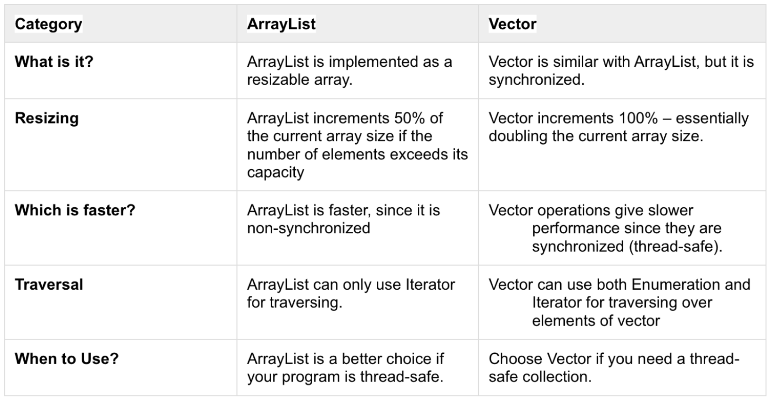


ArrayList

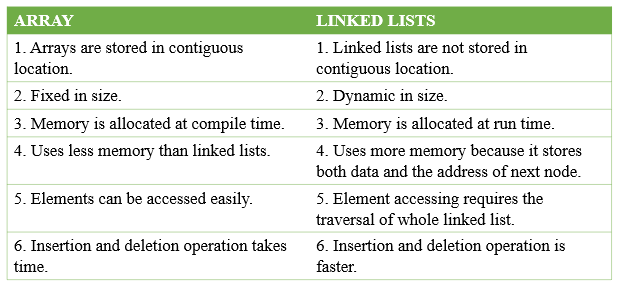
* ArrayList is a dynamically resizable array
* ArrayList is used when we do not know the size of array
* Once 50% of ArrayList capacity is full it copies elements from old array into new array and old array is garbage collected
* ArrayList can only use Iterator for traversal
* ArrayList is not thread-safe

Vector

* Vector also uses a dynamically resizable array
* Once 100% of Vector capacity is full it doubles the current array size
* Vector is slower
* Vector can use both Enumeration and Iterator for element traversal.
* Vector is thread-safe



LinkedList



Differences between or Versus or v/s

Core Java

* JDK v/s JRE vs JVM
* Autoboxing vs Unboxing
* Pass by Value v/s Pass by Reference
* String v/s StringBuffer v/s StringBuilder

Collection Framework

* Array vs ArrayList
* ArrayList vs LinkedList
* HashMap vs LinkedHashMap
* LinkedHashMap vs TreeMap
* HashSet vs LinkedHashSet
* LinkedHashSet vs TreeSet
* Collections.synchronized(map) vs ConcurrentHashMap
* Fail Fast Vs Fail Safe Iterator

Spring

* ApplicationContext vs BeanFactory

Asynchronous Messaging Systems

* Kafka vs RabbitMQ

JPA

* Hibernate vs JPA

SOAP vs REST

Protobuf vs REST

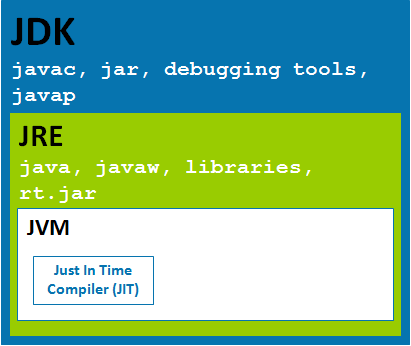
JUNIT 4 vs JUNIT 5

NOSQL vs RDBMS

Synchronous vs Asynchronous Communication

Differences between or Versus or v/s Explanation

* JDK vs JRE vs JVM



Books

* Clean Code by Robert C. Martin
* Designing Data Intensive Applications by Martin Kleppmann

Books

* Clean Code by Robert C. Martin

Chapter 1: Introduction

Chapter 2: Naming

Chapter 3: Functions

Chapter 4: Comments

Chapter 5: Formatting

Chapter 6: Objects vs Data Structures

Chapter 7: Error Handling

Chapter 8: Quickly Using third party software in clean way

Chapter 9: Clean tests

Chapter 10: Clean classes

Chapter 11: Clean Systems

Chapter 12: Emergence of Great Software Systems

Chapter 13: Clean Concurrent Systems

Chapter 14: Code Refactoring

Chapter 15: Code Refactoring

Chapter 16: Code Refactoring

Chapter 17: Code Smells and Heuristics

* Designing Data Intensive Applications by Martin Kleppmann

Chapter 1: Reliable, Scalable and Maintainable

Chapter 2: Data Models and Data Query Languages

Chapter 3: Storage and Retrieval

Chapter 4: Agile Code evolution and Data Encoding

Chapter 5: Data Replication

Chapter 6: Data Partitioning

Chapter 7: Transactions

Chapter 8: Troubles with Distributed Systems

Chapter 9: Consistency and Consensus

Chapter 10: Batch Processing

Chapter 11: Stream Processing

Chapter 12: Future of Data Systems

Best Practices:

Always code for interfaces

Interview Questions:

Why Java is not a purely object oriented programming language?

Why main method is static in java?

Can we overload main method in java?

Why immutable class is final in java?

Can we declare abstract class as final?

Can a final class extend an abstract class in Java?

Can we execute comments in Java?

What is the order of precedence of access modifiers in Java?

Java uses Pass by Value or Pass by Reference?

What is Type Erasure?

HashMap Interview Questions:

Can we have multiple instances of ApplicationContext in a Spring Boot Application?

Interview Questions Explanation:

Question: Why Java is not a purely object oriented programming language?

Answer:

Reason 1: A pure object oriented language should access methods only using object but java has **static variables and static methods** which do not need object instantiation.

Reason 2: Java supports **primitive data type** which are **not objects**

Question: Why main method is static in java?

Answer: Since main method is static, JVM need not create an object to invoke main method

Question: Can we overload main method in java?

Answer: Yes, we can but there should be one and only one method with signature public static void main (String [] args) {}

Question: Why immutable class is final in java?

Answer: In a HashMap you should make sure that the key of HashMap is an immutable object.

<https://www.youtube.com/watch?v=NsXy97Nqq_E>

Question: Can we declare abstract class as final?

Answer: No, at a time a class can either be abstract or final but not both.

<https://www.youtube.com/watch?v=EPWF8SOIvQY>

Question: Can a final class extend an abstract class in Java?

Answer: Yes, but ChildClass must provide implementation for all abstract methods.

Question: Can we execute comments in Java?

Answer: Yes, Unicode characters “\u000d” carries

<https://www.youtube.com/watch?v=KTIoDe7H2Sc>

Question: What is the order of precedence of access modifiers in Java?

Answer: (Least Restricted) public > protected > default > private (Most Restricted)

<https://www.codejava.net/java-core/the-java-language/java-access-modifiers-examples-public-protected-private-and-default>

Question: Java uses Pass by Value or Pass by Reference?

Answer: Java is pass by value

Question: What is Type Erasure?

Answer: Type Erasure is a process in which compiler replaces a generic parameter with actual class.

Question: Can we have multiple instances of ApplicationContext in a Spring Boot Application?

Answer:

Question:

Answer:

References:

|  |  |  |
| --- | --- | --- |
| **Topic** | **Link** | **Author** |
| Spring Boot Annotations | <https://www.youtube.com/watch?v=AXZkhKTbbWc> | Ramesh Fadtare |
| Java Interview Questions and Answers | <https://www.youtube.com/watch?v=KN9BP9y6FKc> | Ramesh Fadtare |
| Top 20 HashMap Interview Questions in Java | <https://www.youtube.com/watch?v=eKoeKQ2yGjU> | Naveen AutomationLabs |
| Collection Framework | <https://www.youtube.com/watch?v=VE_AAUxTUCY&t=7987s> | Riddhi Dutta |
| Java 8 | <https://www.youtube.com/watch?v=ePJrt5-G8eM> | Java Techie |
|  |  |  |