**Table of Contents**

[***Introduction - Fundamentals 2***](#_gjdgxs)

[**Why Java 2**](#_30j0zll)

[**What is JDK, JRE and JVM 2**](#_1fob9te)

[**Java Memory Model**](#_4d34og8) **4**

[**What is Class, Abstract Class, Object, Interface**](#_1ksv4uv) **8**

[**Polymorphism**](#_s2xxlvbb244g) **8**

[**Exception Handling 1**](#)**2**

[**Multithreading 1**](#_3as4poj)**6**

[**Collection Framework**](#_3o7alnk) **19**

[**I/O Streams 2**](#_32hioqz)**2**

[**Java 8 2**](#_vx1227)**3**

[**JDBC. 2**](#)**5**

# Introduction - Fundamentals

## Why Java

* Platform independent – Write once, run anywhere.
* Object Oriented
* Open source, Rich API and good documentation.
* Strong Community Support
* Powerful IDE support/Development tools
* Security

## What is JDK, JRE and JVM

* JDK

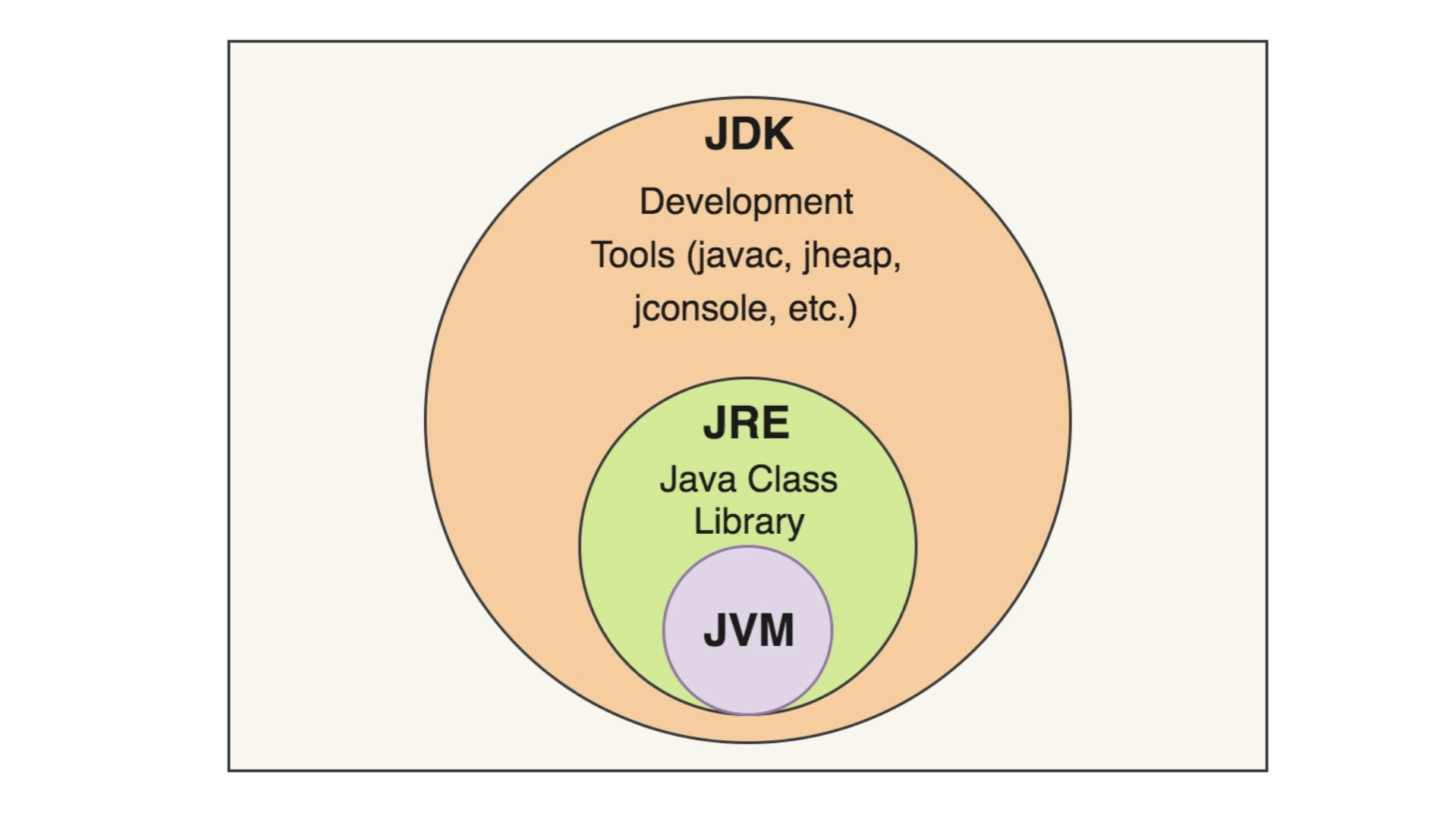
The Java Development Kit (JDK) is a cross-platformed software development environment that offers a collection of tools and libraries necessary for developing Java-based software applications and applets.

* JRE

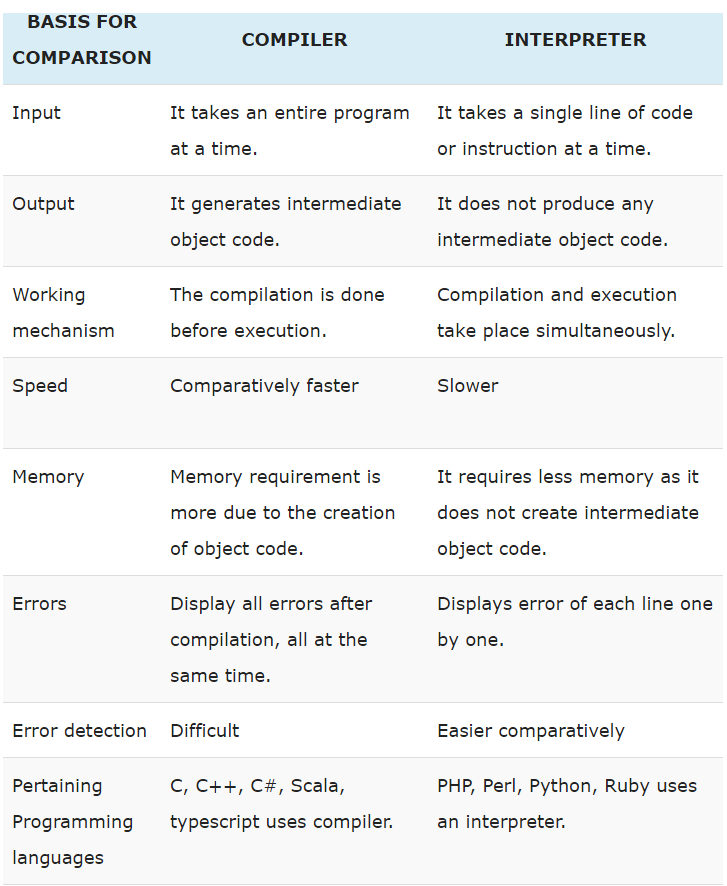
JRE allows Java applications to run. It contains a Java Virtual Machine (JVM), which is a program that executes Java bytecode.

* JVM

A Java Virtual Machine (JVM) is a program that interprets Java bytecode to run as a program by providing a runtime environment that executes this process.



## Interpreter vs Compiler



## 

## Reference Links:

* <https://www.javatpoint.com/features-of-java>
* <https://www.geeksforgeeks.org/differences-jdk-jre-jvm/>

## Interview Questions:

* What is Java?
* Explain the main features of Java.
* Explain the concept of the Java Virtual Machine (JVM).
* What is the difference between JDK, JRE, and JVM?

## Java Memory Model

* Heap Memory
  + Used for dynamic memory allocation.
  + Used to store local variables, method parameters, and return values
  + Objects in the heap can be accessed by multiple threads.
* Stack Memory
  + Used for static memory allocation.
  + Used to store objects
  + Private to each thread
* Garbage Collector
  + Automatic Memory Management
  + Object Lifecycle Management - Objects memory allocation, reference tracking, and deallocation.
  + Clears heap memory when obj is not used for some time.

## Core concepts

* Wrapper Classes
  + Wrapper classes in Java are classes that wrap primitive data types.
  + This means that they provide 100% object-oriented representation of primitive data types.



## Static vs Non Static

**Static members**

* Class-level scope
* Shared by all objects
* Initialization when the class is loaded
* Access using the class name
* Can be accessed by static methods
* Accessed using the class name.
* Used when need of Constant variables

**Static methods**

* Class-level scope
* Can access static variables and methods
* Cannot access non-static variables and methods

**Non-static members**

* Object-level scope
* Unique to each object
* Access using object reference
* Initialization when the object is created
* Can be accessed by both static and non-static methods

**Non-Static Methods**

* Object-level scope
* Can access static and non-static variables and methods

## Access Modifiers

Controls the accessibility of classes, methods, variables, and constructors.

Helps to ensure that only authorized code can access certain parts of the program, enhancing the security, modularity, and reusability of the code.

* **Public**: Public access specifier grants unrestricted access to the class member.
* **Private**: Private access specifier restricts access to the class member within the class itself.
* **Protected**: Protected access specifier grants access to the class member within the class, its subclasses, and within the same package.
* **Default**: The default access modifier, also known as package-private, allows access to the class member only within the same package.



## Reference Links:

* <https://www.baeldung.com/java-stack-heap>
* <https://www.baeldung.com/java-wrapper-classes>
* <https://www.geeksforgeeks.org/difference-between-static-and-non-static-variables-in-java/>
* <https://www.geeksforgeeks.org/access-modifiers-java/>

## 

## Interview Questions:

* What is the difference between int and Integer in Java?
* What is the purpose of the "Java Heap"?
* What is the purpose of the "Stack memory"?
* What are stack overflow errors ,out of heap memory errors? how to handle
* How does garbage collection work in Java?
* Why do we need Wrapper Classes?
* What is the “static” keyword used for?
* Explain PSVM.
* When to use static variables in Java?
* How to access static and non-static variables in Java?
* What is the purpose of the final keyword in Java?
* Explain the difference between public, private, protected, and default access modifiers.
* What is the difference between a memory leak and a memory outage?
* What is memory leak and how do we analyze memory leaks.
* Discuss the role of the “static block” and “non-static block” in Java. How and when is it executed?
* Explain the significance of the final keyword when applied to a static variable or method. Provide examples.
* Explain the concept of package-private access. How does it contribute to encapsulation and information hiding?
* Discuss the importance of encapsulation in Java and how access modifiers play a role in achieving it.
* What permgen memory space. What happens If permgen is full?
* Why did permgen get replaced with Metaspace. What are its additional features?

## What is Class, Abstract Class, Object, Interface

* Class   
  It contains **generic** state(data) and behavior (methods-business logic)
* Object

An object is an instance of a Java class, meaning it contains **specific** state and behavior(for a class number of objects can be created). It allows run time memoryallocation.

* Abstract class

It is a class with implementation gaps. Abstract class in Java is a class that cannot be instantiated directly. Instead, it must be extended by other classes. It provides 0-100% abstraction.

* Interface

Interface is a collection of abstract methods (methods without a body). It provides a way to achieve 100% abstraction, allowing you to declare a set of methods that must be implemented by any class that implements the interface.

* Inheritance

Inheritance in Java is a mechanism in which one class can inherit the properties and behaviors of another class.

**1. Single Inheritance :** One parent class is directly support into one child class

**2. Multilevel Inheritance**: One child class and more than one parent class

**3. Multiple Inheritance:** More than one parent class parallelly supports one child class.

## Polymorphism

Polymorphism allows any task to perform a single action in more than one way.

Types of polymorphism in Java:

* **compile-time polymorphism:** The compiler decides which method to call based on the types of the arguments that are passed to the method.

This type of polymorphism is achieved through method overloading.

* + **Method overloading** is a feature that allows a class to have multiple methods with the same name, but with different parameters.
* **Run-time polymorphism:** The compiler does not know which method to call until runtime. The compiler determines which method to call based on the object type of the class that is calling the method.
  + **Method overriding** is a feature that allows a subclass to override a method from its parent class. The subclass provides its own implementation of the method.

## Reference Links:

* <https://www.javatpoint.com/object-and-class-in-java>
* <https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html>
* <https://www.geeksforgeeks.org/inheritance-in-java/>
* <https://www.geeksforgeeks.org/interfaces-in-java/>
* <https://www.geeksforgeeks.org/polymorphism-in-java/>

## Interview Questions:

* How is a class different from an object in Java?
* What is the purpose of a constructor in Java?
* What is Constructor Overloading?
* What is the purpose of the new keyword and ways to create an object in Java?
* What is the role of the super keyword in a constructor?
* What is an abstract class in Java?
* Can an abstract class have a constructor?
* How is an abstract class different from an interface?
* Can an interface have variables, and if yes, what type are they?
* Can a class implement multiple interfaces in Java?
* Explain the concept of polymorphism in Java.
* What is “this” keyword in Java?
* What is compile-time polymorphism and runtime polymorphism?
* How does method overloading contribute to polymorphism?
* Explain the significance of the @Override annotation in Java.
* Can you achieve runtime polymorphism without using inheritance?
* How does **this** keyword contribute to method overloading and constructor chaining? Provide
* Discuss scenarios where using **this** in a constructor is necessary. How does it prevent naming conflicts?
* In what situations might you encounter a NullPointerException when using **this**, and how can you avoid it?
* Explain the role of the super keyword in Java. How is it used in constructor chaining and method overriding?
* Discuss the differences between super() and this() when used in a constructor.
* Can you provide an example where both this and super are used in the same constructor? What is the purpose of such usage?
* Can method overloading be achieved based on the return type in Java? Why or why not?
* How does the Java compiler determine which overloaded method to call in a given context?
* Discuss the use of the super keyword in method overriding. When is it necessary, and how does it affect the method resolution process?
* Explain the concept of abstract methods and abstract classes in the context of method overriding.
* Discuss the significance of the final keyword in the context of classes. How does it affect class inheritance and method overriding?
* Explain the purpose of the equals() and hashCode() methods in the Object class. How should they be overridden in user-defined classes?
* Discuss the concept of object cloning in Java. How is it implemented, and what considerations should be taken into account?

Strings

* String
  + Strings are objects that represent a sequence of characters.
  + Strings are immutable, which means that once a string is created, its contents cannot be changed.
* StringBuffer
  + A mutable sequence of characters
  + Thread-safe
  + Used for Multi-threaded Application
  + Slower Performance
* StringBuilder
  + A mutable sequence of characters
  + Not Thread-safe
  + Used for Single-threaded Application
  + Faster Performance

## Reference Links:

* <https://www.edureka.co/blog/string-vs-stringbuffer-vs-stringbuilder/>

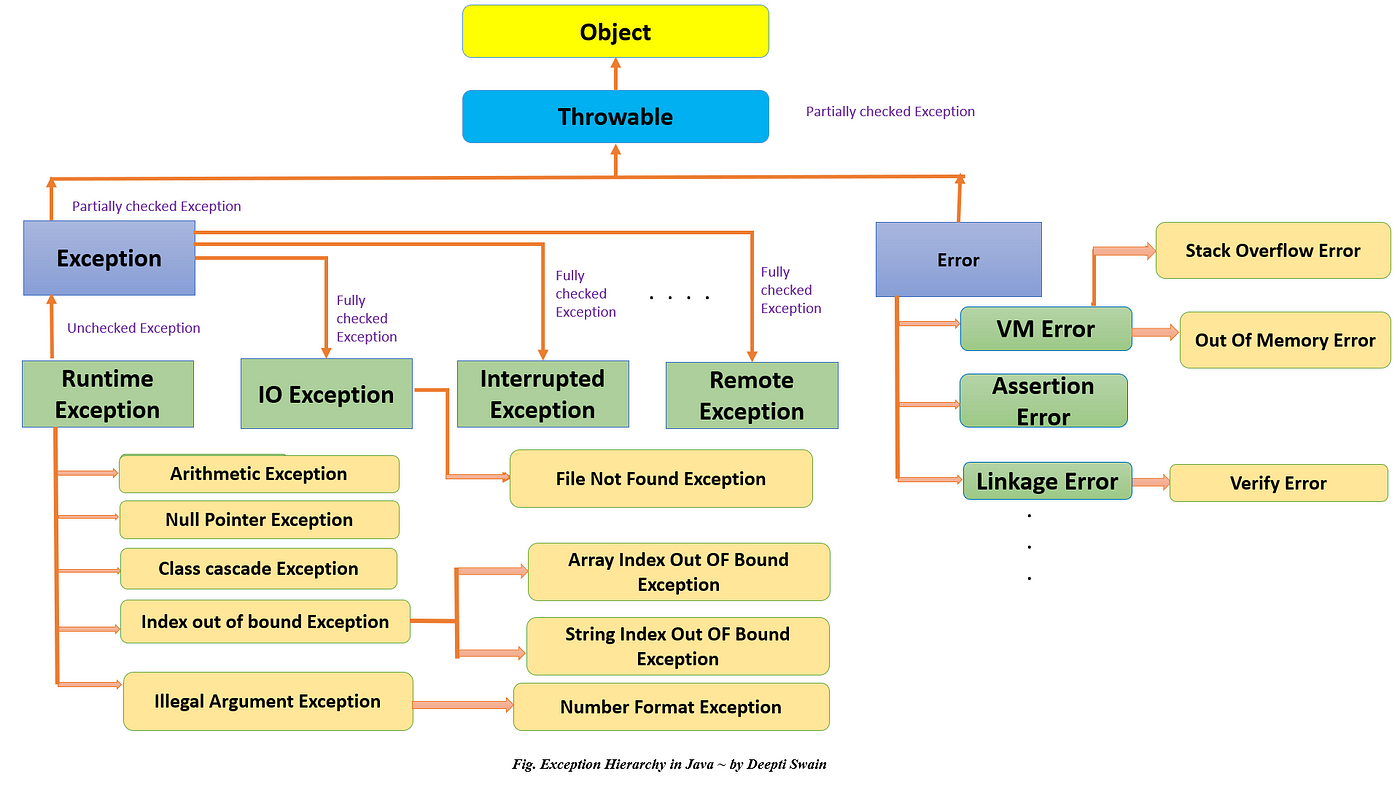
## Interview Questions:

* What is a String in Java?
* Can you modify the content of a String once it is created?
* Explain the concept of String Pool in Java.
* What is the difference between String and StringBuffer in Java?
* What is the StringBuilder class, and how is it different from StringBuffer?
* How does StringBuilder achieve better performance than StringBuffer?
* Difference between “==” and “equals()”.
* Discuss the role of the String pool in Java. How can you explicitly add a String to the pool?
* Explain the importance of the String class being final. How does it relate to security and performance?
* How does the StringBuilder class improve upon the performance of String concatenation? Provide examples of scenarios where StringBuilder is preferable.

## Exception Handling

* **Error:** An error serious problems that cannot be recovered from and typically cause the program to terminate abnormally such as running out of memory or a hardware failure
* **Exception**

An exception in Java is an event that occurs during the execution of a program that disrupts the normal flow of the program.



* **Exception Handling**

Mechanism that allows developers to gracefully manage and respond to runtime errors and exceptional conditions that may occur during the execution of a program.

* **Checked exceptions:** Exceptions that the compiler requires you to handle explicitly
  + **IOException**: Represents any general input-output error.
  + **SQLException**: An error while interacting with a database.
  + **ClassNotFoundException**: Occurs when trying to load a class but the specified class cannot be found.
* **Unchecked exceptions:** Exceptions that the compiler does not require you to handle explicitly such as a hardware failure or a network error.
  + **ArithmeticException**: Thrown when an arithmetic operation results in an undefined or infinite value
  + **NullPointException**: Access an object or invoke a method on a null reference.
  + **InputMismatchException**: Indicates that the input provided does not match the expected type
  + **ArrayIndexOutOfBoundExcepion**: Access an array element outside of the array's bounds.
  + **StringIndexOutOfBoundExcepion**: Access String element outside of the string’s bounds.
  + **NumberFormatException**:Thrown when a method expects a numeric string, but the provided string is not a valid number
* **Handle an Exception:**
  + Try/Catch:
    - The try block encloses the code that might throw an exception.
    - The catch block contains the code to handle the exception if it occurs.
    - Multiple catch blocks can be used to handle different types of exceptions.
  + Throw/Throws:
    - The throw statement is used to explicitly throw an exception.
    - The throws keyword in Java is used to declare that a method can throw one or more exceptions.



* + Custom exceptions
    - Developers can create their own exception classes by extending the Exception class or one of its subclasses.
    - Custom exceptions are useful for handling application-specific errors.

## Reference Links:

* <https://www.geeksforgeeks.org/types-of-exception-in-java-with-examples/>
* <https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html>

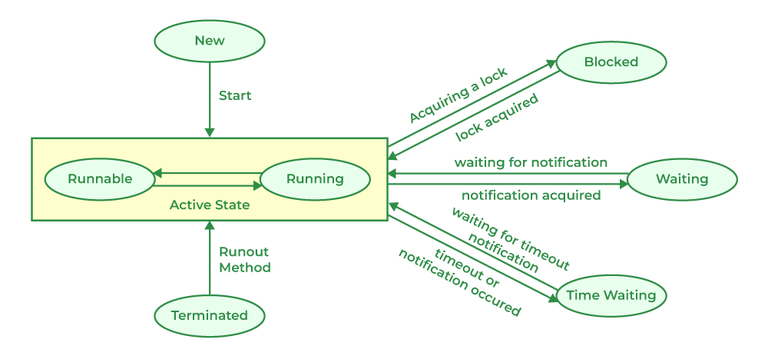
## Interview Questions:

* What is an exception in Java?
* Examine the differences between Error and Exception in Java.
* What is the difference between checked and unchecked exceptions?
* Explain the purpose of the try, catch, and finally blocks in Java.
* What is the purpose of the throws clause in a method signature?
* Explain the difference between the Exception class and the Error class.
* How can you create a custom exception in Java?
* Explain the concept of multiple catch blocks.
* What is the purpose of the finally block?
* What is StackTrace and how do we analyze it?
* Can the finally block be omitted after a try block?
* Explain the difference between throw and throws in Java.
* How does the try-with-resources statement work, and what are its benefits over traditional resource management?
* What is the purpose of the finally block? How does it differ from the catch block?
* Explain the concept of exception chaining in Java. Provide an example demonstrating the use of chained exceptions.
* How does Java handle multi-catch blocks ? What are the advantages and limitations of using multi-catch?
* Discuss the role of the Exception class in the Java exception hierarchy. How can you create custom exceptions, and when would you use them?
* Discuss best practices for logging and tracing exceptions in a Java application. How would you design an effective logging strategy for production systems?
* Explain the concept of exception propagation in Java. How does it impact the design of an application's exception-handling strategy?
* Explain Sql injection error.
* Difference between Classnotfound Exception and NoClassDefFoundError.

## Multithreading

Multithreading in Java refers to the concurrent execution of multiple threads (smaller units of a process) to improve the overall performance and responsiveness of a program.

* Thread:
  + A thread is the smallest unit of execution within a process.
  + In Java, the Thread class is used to create and manage threads.
  + Threads share the same process resources but run independently.
* Thread Lifecycle:
  + **New**: When a thread is created.
  + **Runnable**: When a thread is ready to run.
  + **Running**: The thread is actively executing.
  + **Blocked**: When a thread is waiting for a resource.
  + **Waiting**: The thread is waiting for a specific event to occur.
  + **Timed Waiting**: The thread is waiting for a specified time for an event to occur.
  + **Terminated**: When a thread has completed its execution.



* Thread Priority:

Thread priority allows us to influence the scheduling of threads by assigning them different priority levels.

* + Thread.MIN\_PRIORITY (1)
  + Thread.NORM\_PRIORITY (5)
  + Thread.MAX\_PRIORITY (10)
* Thread Synchronization:

The process of coordinating the execution of multiple threads to ensure that they access shared resources in a safe and consistent manner.

* Creating Threads in Java
  + Extending the Thread Class
  + Implementing the Runnable Interface
  + Using Lambda Expressions (Java 8 and later)

## Reference Links:

* <https://www.tutorialspoint.com/java/java_multithreading.htm>
* <https://www.geeksforgeeks.org/synchronization-in-java/>
* <https://www.javatpoint.com/synchronization-in-java>

## Interview Questions:

* What is multithreading?
* Explain the difference between process and thread.
* How can you create a thread in Java?
* What is the Runnable interface, and how is it different from extending the Thread class?
* What is the wait() and notify() methods used in Java threading?
* How can you achieve synchronization in Java?
* What is a thread pool, and why is it useful?
* Differentiate Multiprocess vs multithreaded
* Explain the concept of thread safety. How can you achieve thread safety in Java?
* Compare synchronized keyword and ReentrantLock. When would you prefer one over the other?
* Discuss the differences between the wait(), notify(), and notifyAll() methods in Java. When would you use each of them?
* Explain the Java Memory Model (JMM) and its significance in the context of multithreading.
* What is a Thread Pool, and why would you use it? Explain the advantages and disadvantages of using thread pools.
* Describe the java.util.concurrent package. Provide examples of classes/interfaces from this package and explain their use in multithreading.
* Discuss the challenges and solutions related to deadlock in multithreading. How can you detect and avoid deadlock situations in Java?
* Explain ExecutorService in Java.
* Differentiate User threads and Daemon threads.

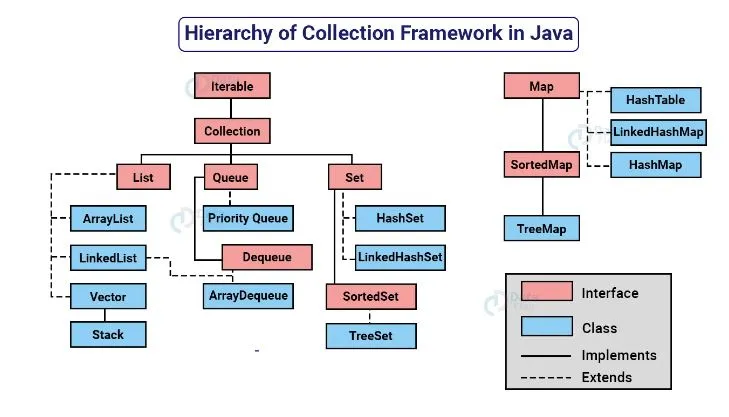
## 

## 

## Collection Framework

The Java Collections Framework provides a set of classes and interfaces for handling and manipulating collections of objects.

* **Consistency and Interoperability:** Provides a standardized way to represent and manipulate collections of objects.
* **Reusable Code:** The framework provides a set of generic, reusable data structures and algorithms.
* **Dynamic Resizing:** Collection classes automatically resize themselves ensuring efficient use of memory.
* **Increased Performance:** The framework provides optimized implementations of common data structures, leading to better performance.
* **Thread-Safety:** The framework includes thread-safe implementations which is crucial in multithreaded environments where multiple threads may access and modify collections simultaneously.



* ArrayList:
  + Dynamic array
  + Implementation of the List interface in the Java Collections Framework.
  + It provides a resizable array-backed data structure
* Map:
  + Collection of key-value pairs.
* Set:
  + Set is an interface that represents a collection of unique elements.
* Queue:
  + Queue is an interface that represents a collection designed for holding elements prior to processing.
  + It follows the First-In-First-Out (FIFO) principle

## Comparator vs Comparable

| **Feature** | **Comparable** | **Comparator** |
| --- | --- | --- |
| Purpose | Defines the natural ordering of the class. | Defines the custom ordering of objects. |
| Implementation | Implemented by the class itself. | Implemented by a separate class or object. |
| Usage | Used to sort collections of the class. | Used to sort collections of different classes or to provide multiple sorting options for a single class. |
| Method | compareTo() | compare() |
| Return value | -1, 0, or 1 to indicate the relationship between two objects. | -1, 0, or 1 to indicate the relative order of two objects. |
| Main Usage | Used for sorting based on the class’s natural order. | Used for more complex sorting that goes beyond the natural order. |

## 

## Reference Link:

* <https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html>

## Interview Questions

* What is the Java Collection Framework?
* What are generics in Java, and why are they used?
* What is the difference between a Collection and a Collections in Java?
* Explain the difference between the List, Set, and Map interfaces.
* Explain the difference between ArrayList and LinkedList in Java.
* What is the purpose of the Iterator interface in Java collections?
* What is the hashCode() method used for in Java collections?
* What is the purpose of the Comparator interface in Java collections?
* Discuss the differences between the Comparator and Comparable interfaces. When would you use one over the other?
* Explain the concept of type erasure in Java generics. How does it impact the behavior of generics at runtime?
* What are bridge methods in Java generics? Provide an example of when a bridge method is generated by the compiler
* What are reifiable types, and how do they differ from non-reifiable types? Provide examples of both
* How does the diamond operator (<>) improve code readability in Java generics? Provide examples.
* Fail-Fast vs. Fail-Safe Iterators
* Explain the internal workings of the HashMap class, including bucketing, hash codes, and collision resolution.
* Discuss the differences between HashMap and ConcurrentHashMap regarding thread-safety. When would you choose one over the other?
* Explain internal implementation of HashMap.
* When do we override Hashcode vs equals()
* Differentiate LinkedList vs hashmap
* Discuss the differences between Comparator and Comparable.

## I/O Streams

Input/Output (I/O) streams provide a mechanism for reading from and writing to different sources, such as files, network connections, and in-memory data structures.

* **Byte Streams:**
  + Operate on raw binary data.
  + Classes with names ending in InputStream or OutputStream are byte streams, like FileInputStream and FileOutputStream.
* **Character Streams:**
  + Deal with character data and automatically handle character encoding and decoding.
  + Classes with names ending in Reader or Writer are character streams, like FileReader and FileWriter.
* **File I/O:**
  + FileInputStream and FileOutputStream are used for reading from and writing to files.
  + FileReader and FileWriter are used for character-based file I/O.

## Reference Links:

* <https://www.geeksforgeeks.org/java-io-input-output-in-java-with-examples/>
* [https://intellipaat.com/blog/tutorial/java-tutorial/java-file-io/](https://www.geeksforgeeks.org/java-io-input-output-in-java-with-examples/)

Interview Questions:

* What is a stream in Java?
* What is the difference between InputStream and OutputStream in Java?
* Explain the difference between Reader and Writer in Java.
* What is the purpose of InputStreamReader and OutputStreamWriter?
* What is the purpose of BufferedInputStream and BufferedOutputStream?

## 

## Java 8

FEATURES

* Lambda Expressions:
  + Allow you to write anonymous functions in a concise and expressive way.
  + Passing short blocks of code as arguments to methods
* Functional Interfaces:
  + A functional interface in Java is an interface that contains only one abstract method.
  + Promote functional programming concepts
  + Simplify code by reducing the need for explicit class implementations.
* Stream API:
  + Provides a comprehensive set of methods for processing collections of data.
  + Enables efficient data manipulation, filtering, aggregation, and transformation using streams of elements.
* Date and Time API:
  + Provides a more flexible and consistent way to represent and manipulate date and time values.
* Default Methods:
  + Allow to add new methods to an interface without breaking existing implementations.
* Optional Class:
  + Tool for handling null values safely and effectively.

## Reference Links

* <https://www.interviewbit.com/blog/java-8-features/>
* <https://www.javatpoint.com/java-8-features>

## Interview Questions:

* What are the main features introduced in Java 8?
* Explain the concept of Lambda Expressions in Java 8.
* What is the functional interface in Java 8? Can you provide an example?
* How does the new forEach() method in the Iterable interface work in Java 8?
* What is the Stream API in Java 8? How does it differ from the Collections API?
* What is the difference between a consumer and a supplier in Java 8? Provide examples.
* How does the Optional class help in handling null values in Java 8?
* What is the purpose of the Default Method feature in Java 8 interfaces?
* Explain the Method Reference feature in Java 8. Provide examples.
* What is the flatMap operation in the Stream API? How is it different from a map?
* What is the java.time package in Java 8 and how does it improve date and time handling?
* Explain the concept of the @FunctionalInterface annotation.
* How does the CompletableFuture class in Java 8 help with asynchronous programming?
* What is the purpose of the Collector interface in the Stream API?
* Can you provide an example of using the reduce operation in the Stream API?
* What is the peek method in the Stream API used for?
* Explain the concept of the Default Methods in interfaces.
* How does the filter method work in the Stream API? Provide an example.
* What is the purpose of the @FunctionalInterface annotation in Java 8?

JDBC

## JDBC is an API that allows Java programs to access databases.

## Connect to a database

## Execute SQL statements

## Process the results of SQL statements

## Close the connection to the database

Connection Establishment

* Load the JDBC driver:
  + Class.forName loads the JDBC driver for MySQL.
* Establish the connection:
  + DriverManager.getConnection(JDBC\_URL, USERNAME, PASSWORD) creates a connection to the specified database.
* Perform database operations:
  + Perform database operations like executing SQL queries, updates, inserts, etc..
* Close the connection:
  + Close the connection in the finally block else an exception occurs.

## 

## Reference Links:

* <https://www.geeksforgeeks.org/introduction-to-jdbc/>
* <https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html>

## Interview Questions:

* What is JDBC?
* Explain the steps to establish a connection to a database using JDBC.
* What are the core interfaces of JDBC?
* Differentiate between Statement, PreparedStatement, and CallableStatement.
* What is the purpose of the JDBC Driver? Name some popular JDBC Drivers.
* Explain the difference between ResultSet and Statement in JDBC.
* How do you handle SQLExceptions in JDBC?
* What is connection pooling, and why is it important in JDBC?
* Explain the concept of batch processing in JDBC.
* What is the role of the JDBC DriverManager class?
* What is the significance of the ResultSetMetaData and DatabaseMetaData interfaces?
* How can you retrieve data from a ResultSet in JDBC?
* What are the advantages of using PreparedStatement over Statement in JDBC?
* Explain the transaction management in JDBC.
* How does JDBC support stored procedures?
* Explain the use of the CallableStatement interface in JDBC.
* What is the role of the DriverManager class in JDBC?
* Explain the concept of connection pooling and its benefits in JDBC.
* What is the purpose of the BatchUpdateException class in JDBC?
* How can you handle transactions in JDBC?
* What are the common challenges and best practices in JDBC programming?
* How does JDBC support metadata retrieval from a database?