- **Java Inheritance**
  Java inheritance is a mechanism in which one class acquires the properties (fields) and behaviors (methods) of another class. It allows for hierarchical classification, where a subclass inherits features from its parent class, promoting code reuse and method overriding. The extends keyword is used to establish an inheritance relationship, enabling polymorphism and dynamic method dispatch.
- **Polymorphism**
  Polymorphism in Java allows objects to be treated as instances of their parent class rather than their actual class. The two types of polymorphism are compile-time (method overloading) and runtime (method overriding). This concept enables one interface to be used for a general class of actions, facilitating flexibility and integration of code.
- **Method Overloading in Java**
  Method overloading in Java occurs when multiple methods in the same class have the same name but different parameters (different type, number, or both). It allows methods to perform similar functions with different input parameters, improving code readability and usability. The return type of methods can be different but it doesn't affect overloading.
- **Method Overriding in Java**
  Method overriding in Java is when a subclass provides a specific implementation for a method that is already defined in its superclass. The overridden method must have the same name, return type, and parameters. This feature supports runtime polymorphism, allowing dynamic method invocation based on the object's actual type.
- **Java Abstraction**
  Java abstraction is a process of hiding the implementation details from the user and only exposing the essential features of an object. It can be achieved using abstract classes and interfaces. Abstract classes can have both abstract (without body) and concrete methods, while interfaces can only have abstract methods (until Java 8 introduced default methods).
- **Java Encapsulation**
  Java encapsulation is a mechanism of wrapping the data (variables) and code (methods) together as a single unit. It restricts direct access to some of the object's components, which can be achieved by making the variables private and providing public getter and setter methods. This helps in protecting the data from unauthorized access and modification.
- **Rules for Java Method Overriding**
  - The method in the subclass must have the same name, return type, and parameters as in the superclass.
  - The overriding method cannot have a more restrictive access modifier than the overridden method.
  - A subclass can only override methods that are inherited from the superclass (i.e., not private or final).
  - The overriding method can throw narrower or fewer checked exceptions than the overridden method.
- **Constructors in Java**

Constructors in Java are special methods that are called when an instance of a class is created. They have the same name as the class and do not have a return type. Constructors are used to initialize objects, setting initial values for fields. Java provides a default constructor if no constructors are defined explicitly.

- **Types of Java Constructors**
    - **Default Constructor**: A constructor with no parameters provided by Java if no other constructors are defined.
    - **Parameterized Constructor**: A constructor that accepts arguments to initialize an object with specific values.
    - **Constructor Overloading in Java**

      Constructor overloading in Java is a technique of having more than one constructor in a class with different parameters. It allows objects to be initialized in different ways. The constructors must have a unique parameter list, enabling the creation of objects with different sets of initial values.
- **Interface in Java**
  An interface in Java is a reference type, similar to a class, that can contain only constants, method signatures, default methods, static methods, and nested types. Interfaces provide a way to achieve abstraction and multiple inheritance in Java. A class that implements an interface must provide implementations for all its methods. Interfaces are declared using the interface keyword.

Exception Handling

- **Hierarchy of Exception classes:** Java's exception classes are part of an inheritance hierarchy that starts with the Throwable class. This branches into Error (for serious errors) and Exception (for conditions that a reasonable application might want to catch).
- **Types of Exception:** Java exceptions are categorized into Checked exceptions (subclasses of Exception excluding RuntimeException) that are checked at compile-time, and Unchecked exceptions (subclasses of RuntimeException) that are checked at runtime.
- **Try:** The try block contains code that might throw an exception. If an exception occurs, it is thrown to the nearest enclosing catch block.
- **Catch:** The catch block is used to handle exceptions that occur in the associated try block. Multiple catch blocks can be used to handle different types of exceptions.
- **Finally:** The finally block contains code that is always executed, regardless of whether an exception is thrown or not. It is typically used for resource cleanup.
- **Throw:** The throw keyword is used to explicitly throw an exception. It is followed by an instance of Throwable.
- **Throws:** The throws keyword is used in method signatures to declare the exceptions that the method can throw. It provides a warning to the caller of the method about potential exceptions.

- **Multiple catch block:** Java allows multiple catch blocks to handle different types of exceptions thrown from the same try block. This helps in specific exception handling.
- **Exception Handling with method Overriding:** In method overriding, if the superclass method declares an exception, the subclass overridden method can declare the same exception or its subclass but cannot declare a broader exception.

Java Collection Framework

- **Hierarchy of Collection Framework:** The Java Collection Framework provides a unified architecture for representing and manipulating collections. It includes interfaces like Collection, Set, List, Queue, and Map, and their implementations like ArrayList, LinkedList, HashSet, TreeSet, and HashMap.
- **Collection interface:** The Collection interface is the root interface of the collection hierarchy. It represents a group of objects, known as elements, and provides methods to add, remove, and query elements.
- **Iterator interface:** The Iterator interface provides methods to iterate over any collection. It includes hasNext(), next(), and remove() methods to traverse and modify the collection.
- **Set:** A collection that does not allow duplicate elements (e.g., HashSet, TreeSet).
- **List:** An ordered collection that can contain duplicate elements (e.g., ArrayList, LinkedList).
- **Queue:** A collection used to hold multiple elements prior to processing, typically in a FIFO order (e.g., PriorityQueue).
- **Map:** An object that maps keys to values, with no duplicate keys (e.g., HashMap, TreeMap).
- **Comparator:** An interface for defining custom order for sorting (e.g., Collections.sort with a comparator).
- **Comparable interfaces:** An interface used to define the natural order of objects (e.g., String, Integer implement Comparable).
- **ArrayList:** A resizable array implementation of the List interface.
- **Vector:** A synchronized, resizable array implementation of the List interface.
- **LinkedList:** A doubly linked list implementation of the List and Deque interfaces.
- **PriorityQueue:** An unbounded priority queue based on a priority heap.
- **HashSet:** A collection that uses a hash table for storage, ensuring no duplicates.
- **LinkedHashSet:** A HashSet with predictable iteration order.
- **TreeSet:** A NavigableSet implementation based on a red-black tree, sorted in natural order or by a comparator.
- **HashMap:** A hash table-based implementation of the Map interface.
- **ConcurrentHashMap classes:** A thread-safe implementation of the Map interface, optimized for concurrent access.