Java Fundamentals

- **Java** is an object-oriented programming language designed to be platform-independent.
- **JVM** executes Java bytecode and manages memory and runtime operations.
- **JDK** contains tools required to develop, compile, and run Java programs.
- **JRE** provides libraries and environment needed to execute Java applications.
- **Bytecode** is an intermediate code that allows Java programs to run on any system.
- **Primitive data types** store simple values directly in memory.
- **Non-primitive data types** store references to objects and complex data.
- **Type casting** converts a variable from one data type to another.
- **Operators** are symbols used to perform operations on data.

Control Flow Statements

- **If-else** executes code blocks based on conditions.
- **Switch** selects execution paths based on a variable's value.
- **For loop** executes a block of code for a fixed number of iterations.
- **While loop** repeats execution as long as a condition remains true.
- **Do-while loop** executes code at least once before checking the condition.
- **Break** terminates a loop or switch statement.
- **Continue** skips the current loop iteration and moves to the next.

Object-Oriented Programming

- **Class** is a blueprint used to create objects.

- **Object** is a real-world entity created from a class.
- **Encapsulation** binds data and methods together while restricting access.
- **Inheritance** allows one class to acquire properties of another class.
- **Polymorphism** enables the same method name to perform different behaviors.
- **Method overloading** allows multiple methods with the same name but different parameters.
- **Method overriding** allows a subclass to modify parent class behavior.
- **Abstraction** hides implementation details and exposes only essential features.
- **Interface** defines a contract that classes must implement.
- **Association** represents a relationship between classes.
- **Aggregation** is a weak relationship where objects can exist independently.
- **Composition** is a strong relationship where dependent objects cannot exist alone.
- **this keyword** refers to the current class object.
- **super keyword** refers to the parent class object.
- **final keyword** prevents modification of variables, methods, or classes.

Constructors

- **Constructor** initializes an object when it is created.
- **Default constructor** initializes objects with default values.
- **Parameterized constructor** initializes objects with user-defined values.
- **Constructor overloading** allows multiple constructors in the same class.
- **Constructor chaining** calls one constructor from another.

Strings

- **String** objects are immutable and cannot be changed once created.
- **StringBuilder** allows mutable string operations without thread safety.
- **StringBuffer** allows mutable string operations with thread safety.

Arrays

- **Array** stores fixed-size elements of the same data type.
- **Multidimensional array** stores data in rows and columns format.

Exception Handling

- **Exception** is an event that disrupts normal program execution.
- **Checked exceptions** are checked at compile time.
- **Unchecked exceptions** occur during runtime.
- **Try-catch** handles exceptions safely.
- **Finally** executes code regardless of exception occurrence.
- **Throw** explicitly throws an exception.
- **Throws** declares exceptions in method signature.
- **Custom exception** is user-defined exception handling.

Packages & Access Control

- **Package** groups related classes and interfaces.
- **Public** allows access from anywhere.
- **Private** restricts access within the same class.
- **Protected** allows access within package or subclasses.

- **Default access** allows access only within the package.

Collections Framework

- **Collection** stores dynamic groups of objects.
- **List** allows duplicate elements and maintains order.
- **Set** stores unique elements only.
- **Map** stores data as key-value pairs.
- **ArrayList** uses dynamic array for fast access.
- **LinkedList** uses linked nodes for efficient insertion.
- **HashSet** stores unordered unique elements.
- **TreeSet** stores sorted unique elements.
- **HashMap** stores unordered key-value pairs.
- **Iterator** is used to traverse collections.
- **Comparable** defines natural sorting order.
- **Comparator** defines custom sorting logic.

Multithreading

- **Thread** allows concurrent execution of tasks.
- **Runnable** defines a task to be executed by a thread.
- **Thread lifecycle** defines states from creation to termination.
- **Synchronization** controls access to shared resources.
- **Deadlock** occurs when threads wait indefinitely for resources.
- **Executor framework** manages thread pools efficiently.

File Handling

- **File class** represents files and directories.
- **Byte streams** handle binary data.
- **Character streams** handle text data.
- **Buffered streams** improve I/O performance.
- **Serialization** converts objects into byte streams.
- **Deserialization** converts byte streams back into objects.

Java 8+ Features

- **Lambda expressions** provide concise functional code.
- **Functional interface** contains exactly one abstract method.
- **Stream API** processes collections using functional operations.
- **Method reference** refers to a method using shorthand syntax.
- **Optional** prevents null pointer exceptions.
- **Date-Time API** provides modern date and time handling.

Memory Management

- **Stack memory** stores method calls and local variables.
- **Heap memory** stores objects and class instances.
- **Garbage collector** automatically frees unused memory.
- **JVM memory areas** manage program execution efficiently.

JDBC

- **JDBC** connects Java applications to databases.
- **Driver** enables communication between Java and database.
- **Connection** establishes database connectivity.
- **Statement** executes SQL queries.
- **PreparedStatement** executes precompiled SQL securely.
- **ResultSet** holds query results.

Annotations & Reflection

- **Annotations** provide metadata for code.
- **Reflection** allows runtime access to class details.

Advanced Concepts

- **Generics** provide compile-time type safety.
- **Enum** defines a fixed set of constants.
- **Varargs** allow passing variable number of arguments.
- **Inner class** is defined inside another class.
- **Static block** executes once when class loads.
- **Design patterns** provide reusable solutions to common problems.