Java

List of key fundamental topics in Java, organised by category, that you should learn to build a strong understanding of the language.

1. Basic Syntax and Structure
- **Java Program Structure:** Understanding the `class` structure, `main()` method, and how Java programs are executed.
- **Statements:** Using semicolons to end statements, blocks of code enclosed in curly braces(`{}`).
- **Comments:** Single-line (`//`), multi-line (`/*...*/`), and documentation comments (`/**...*/`).

2. Data Types and Variables
- **Primitive Data Types:** `int`, `long`, `float`, `double`, `char`, `boolean`, `byte`, and `short`.
- **Reference Types**: Arrays, Strings, and objects of classes.
- **Type Casting**: Implicit and explicit casting (type conversion).
- **Variables:** Declaring variables, final (constants), scope of variables, and initialisation.

3. Control Flow
- **Conditional Statements:** `if`, `else`, `else if`, and `switch`.
- **Loops:** `for`, `while`, `do-while` loops, and loop control statements like `break` and `continue`.
- **Switch-Case:** How to use `switch` for conditional branching with multiple cases.

4. Methods and Functions
- **Defining Methods:** Syntax for defining methods with return types, parameters, and method signatures.
- **Method Overloading:** Creating multiple methods with the same name but different parameters.
- **Method Arguments:** Passing parameters by value, varargs (variable-length arguments).
- **Return Types:** Returning values from methods, `void` methods.

5. Object-Oriented Programming (OOP) Concepts
- **Classes and Objects:** Creating classes, creating objects, constructors, and the `new` keyword.
- **Encapsulation:** Using access modifiers (`private`, `public`, `protected`) to protect data.
- **Inheritance:** `extends` keyword, inheriting properties and methods from a superclass.
- **Polymorphism**: Method overriding and method overloading, dynamic method dispatch.
- **Abstraction**: Abstract classes, abstract methods, and interfaces.
- **Interfaces**: How to define and implement interfaces.
- **Access Modifiers**: `private`, `protected`, `public`, and default access.

6. Arrays and Collections
- **Arrays:** Defining, initialising, and accessing elements in an array. Multidimensional arrays.
- **ArrayList**: Dynamic arrays in the `java.util` package.
- **Other Collections:** Introduction to `LinkedList`, `HashSet`, `TreeSet`, `HashMap`, `TreeMap`, `Stack`, `Queue`, and `PriorityQueue`.
- **Generics:** Type safety in collections (e.g., `ArrayList<String>`), wildcard (`?`), and generic methods.

7. Exception Handling
- **Try-Catch:** Using `try`, `catch`, and `finally` blocks to handle exceptions.
- **Throw and Throws:** Throwing exceptions manually using the `throw` keyword and declaring exceptions in method signatures with `throws`.
- **Custom Exceptions:** Creating your own exception classes by extending `Exception` or `RuntimeException`.
- **Checked vs Unchecked Exceptions:** Understanding the difference between them.

8. Input and Output (I/O)
- **File I/O:** Reading from and writing to files using classes like `FileReader`, `FileWriter`, `BufferedReader`, and `BufferedWriter`.
- **Streams:** Understanding input and output streams (`InputStream`, `OutputStream`, `Reader`, `Writer`).
- **Serialization:** Saving and reading Java objects from files using serialization (`Serializable` interface).

9. String Manipulation
- **String Class:** Creating and manipulating strings using methods like `length()`, `charAt()`, `substring()`, `equals()`, `concat()`, `toUpperCase()`, etc.
- **StringBuilder and StringBuffer:** Working with mutable strings to optimize performance when modifying strings.

10. Java Libraries and APIs
- **Java Standard Library:** Commonly used utility classes in `java.util`, such as `Date`, `Calendar`, `Random`, `Math`, etc.
- **Utility Classes:** `Arrays`, `Collections`, `Objects`, `Optional`, etc.
- **Lambda Expressions:** Introduction to functional programming style with lambda expressions.
- **Streams API:** Processing collections with streams, filtering, mapping, reducing, etc.

11. Multithreading and Concurrency
- Thread Basics: Creating and managing threads using `Thread` class or implementing `Runnable`.
- **Synchronisation**: Using the `synchronized` keyword and locks to prevent race conditions.
- **Executor Service**: Using `ExecutorService` for managing threads.
- **Deadlocks**: Understanding and avoiding deadlocks.
- **Concurrency Utilities**: Understanding `CountDownLatch`, `CyclicBarrier`, `Semaphore`, etc.

12. Java 8 Features and Beyond
- **Lambda Expressions**: Using anonymous functions and improving code readability.
- **Stream API**: Working with collections and sequences of data in a functional programming style.
- **Optional Class**: Handling null values without explicit null checks.
- **Default Methods in Interfaces**: Methods with a body in interfaces.
- **Method References**: Shortening lambda expressions with method references.

13. Java Memory Management
- **Heap vs Stack:** Understanding how memory is allocated for objects and primitives.
- **Garbage Collection:** How the JVM handles memory management and garbage collection.
- **Finalization:** Using the `finalize()` method to release resources before object destruction.

14. Java Development Tools
- **JDK, JRE, and JVM:** Understanding the roles of the Java Development Kit (JDK), Java Runtime Environment (JRE), and Java Virtual Machine (JVM).
- **Build Tools:** Using tools like Maven and Gradle to manage dependencies and build projects.
- Integrated Development Environments (IDEs)**: Working with IDEs such as IntelliJ IDEA, Eclipse, or NetBeans.
- **Debugging:** Using breakpoints, watch variables, and stepping through code to debug Java programs.

15. Design Patterns
- **Creational Patterns:** Singleton, Factory, Builder, Abstract Factory, etc.
- **Structural Patterns:** Adapter, Composite, Decorator, etc.
- **Behavioral Patterns:** Observer, Strategy, Command, State, etc.

Next Steps:
Once you've mastered the fundamental topics, you can explore advanced topics like:
- **Networking**: Using `Socket` and `ServerSocket` for client-server communication.
- **JavaFX / Swing**: Building desktop applications with Java GUI frameworks.
- **Web Development**: Using Java for web development (Servlets, JSP, Spring Framework).
- **Database Connectivity**: Working with JDBC to interact with relational databases.
- **Unit Testing**: Writing unit tests using JUnit and TestNG.
- **Spring Framework**: Learning about Dependency Injection, AOP, and the Spring ecosystem for building enterprise applications.

Conclusion:
The fundamentals of Java provide a solid foundation that you can build on as you progress to more advanced topics. Mastering these concepts will allow you to write efficient, clean, and scalable Java programs.