**1. Java Basics**

* **Syntax & Structure**: Learn the basic syntax, structure of a Java program, and how Java code is organized (classes, methods, packages, etc.).
* **Data Types**: Understand primitive data types (int, float, double, char, boolean) and reference types (objects, arrays, strings).
* **Variables & Constants**: Understand variable declarations, data type conversions, constants (final keyword).
* **Operators**: Learn about arithmetic, relational, logical, bitwise, and assignment operators.
* **Control Flow**: Understand if, else, switch, while, do-while, and for loops.

**2. Object-Oriented Programming (OOP) Principles**

* **Classes and Objects**: Learn how to define classes and create objects in Java.
* **Encapsulation**: Understand how to hide data using access modifiers (private, public, protected), and use getter/setter methods.
* **Inheritance**: Learn how to create subclasses, and use extends to inherit properties and methods from a parent class.
* **Polymorphism**: Study method overloading, method overriding, and dynamic method dispatch.
* **Abstraction**: Understand abstract classes and interfaces to implement abstract behavior.
* **Interfaces vs Abstract Classes**: Understand when to use interfaces and abstract classes, and how they differ.

**3. Core Java APIs**

* **String Handling**: Learn how to manipulate strings using String, StringBuilder, and StringBuffer.
* **Arrays**: Learn to create and manipulate arrays, understand multidimensional arrays.
* **Collections Framework**: Understand lists (ArrayList, LinkedList), sets (HashSet, TreeSet), maps (HashMap, TreeMap), and queues (PriorityQueue, LinkedList).
* **Generics**: Learn how to use generics to write type-safe code, particularly with collections.

**4. Exception Handling**

* **Exceptions**: Understand the difference between checked and unchecked exceptions.
* **Try-Catch**: Learn how to handle exceptions using try, catch, finally, and custom exceptions.
* **Throw and Throws**: Understand how to throw exceptions manually and declare exceptions in method signatures.
* **Custom Exceptions**: Learn to create your own exception classes.

**5. Java I/O (Input/Output)**

* **Streams**: Understand byte streams (e.g., FileInputStream, FileOutputStream) and character streams (e.g., FileReader, FileWriter).
* **Buffered Streams**: Learn to use BufferedReader, BufferedWriter for efficient I/O operations.
* **Serialization**: Learn how to serialize and deserialize objects using ObjectOutputStream and ObjectInputStream.
* **File Handling**: Learn how to work with files using the java.nio.file package.

**6. Concurrency (Multithreading)**

* **Threads**: Understand how to create and manage threads using Thread class and Runnable interface.
* **Synchronization**: Learn how to avoid concurrency issues using synchronized blocks and methods.
* **Concurrency Utilities**: Learn about the java.util.concurrent package, which includes tools like ExecutorService, CountDownLatch, Semaphore, and CyclicBarrier.

**7. Java 8 Features (and later)**

* **Lambda Expressions**: Learn to use anonymous functions to simplify code and support functional programming.
* **Streams API**: Understand how to use the Streams API for functional-style operations on collections (map, filter, reduce, etc.).
* **Optional**: Learn about the Optional class for handling null values and avoiding NullPointerExceptions.
* **Default and Static Methods in Interfaces**: Understand how interfaces can have method implementations using default and static.

**8. Memory Management and Garbage Collection**

* **Heap and Stack**: Learn about memory allocation in the heap and stack, and how Java manages memory.
* **Garbage Collection**: Understand how Java performs automatic garbage collection, the role of the garbage collector, and how to manage memory effectively.

**9. Java Reflection**

* Learn how to use the Reflection API to inspect and manipulate classes, methods, fields, and annotations at runtime.

**10. Java Design Patterns (Optional but Useful)**

* **Singleton Pattern**
* **Factory Pattern**
* **Observer Pattern**
* **Strategy Pattern**
* **Decorator Pattern**
* **Iterator Pattern**
* **Facade Pattern**

**11. JVM (Java Virtual Machine) Basics**

* **JVM Architecture**: Learn about the components of the JVM (Class Loader, Bytecode, Heap, Stack, etc.).
* **JRE and JDK**: Understand the difference between Java Runtime Environment (JRE) and Java Development Kit (JDK).
* **JVM Tuning**: Learn how to optimize JVM performance and memory management using JVM flags.

**12. Unit Testing**

* **JUnit**: Learn how to write unit tests for Java code using the JUnit testing framework.
* **Mocking Frameworks**: Get familiar with frameworks like Mockito to mock objects and simulate behaviors in unit tests.

**13. Basic Java Development Tools**

* **IDE**: Learn to use an IDE (e.g., IntelliJ IDEA, Eclipse, or NetBeans) effectively for Java development.
* **Build Tools**: Get familiar with build tools like Maven or Gradle for managing dependencies and building projects.
* **Version Control**: Learn Git for version control to manage source code and collaborate with other developers.

**14. Java APIs and Libraries**

* **JavaFX / Swing** (if you want to build desktop applications)
* **Networking (Sockets, HTTP)**: Learn how to work with networking APIs to build networked applications.
* **Database Connectivity (JDBC)**: Learn how to connect to and interact with databases using JDBC.

**Additional Tips**

* **Practice Regularly**: Implement small projects to reinforce concepts. For example, create a basic CRUD application or a simple banking system.
* **Understand the Core Principles**: Focus on understanding the why behind the concepts. It's easy to follow syntax, but understanding why things work a certain way will help you build better, maintainable code.

Do you want me to dive deeper into any of these areas, or focus on a specific concept you’d like to learn more about?