1. Basic Structure

```java
package com.example; // Package declaration (optional)

import java.util.Scanner; // Import statements

public class Main { // Class declaration
    public static void main(String[] args) { // Main method (entry point)
        System.out.println("Hello, World!"); // Code execution starts here
    }
}
```

---

2. Data Types

Primitive Data Types:

- **Integers**:
    - `byte` (8-bit), `short` (16-bit), `int` (32-bit), `long` (64-bit)

```java
byte smallNum = 10; // 8-bit integer
short mediumNum = 100; // 16-bit integer
int largeNum = 1000; // 32-bit integer
long veryLargeNum = 100000L; // 64-bit integer, use 'L' for long
```

- **Floating-Point**:
    - `float` (32-bit), `double` (64-bit)

```java
float pi = 3.14f; // Use 'f' for float
double bigPi = 3.141592653589793; // 64-bit floating-point
```

- **Character**:
    - `char` (16-bit Unicode)

```java
char letter = 'A'; // 16-bit Unicode character
```

- **Boolean**:
    - `boolean` (`true` or `false`)

```java
boolean isJavaFun = true; // Boolean value
```

Reference Data Types:

- **String**:

```java
String name = "Java"; // String of characters
```

- **Arrays**:

```java
int[] numbers = {1, 2, 3, 4, 5}; // Array of integers
String[] names = new String[3]; // Fixed-size array
```

---

3. Variables

- **Local Variables**:

```java
int localVar = 10; // Declared inside a method
```

- **Instance Variables**:

```java
class MyClass {
    int instanceVar; // Declared inside a class, outside methods
}
```

- **Static Variables**:

```java
class MyClass {
    static int staticVar; // Shared across all instances
}
```

- **Constants**:

```java
final double PI = 3.14159; // Cannot be changed
```

---

4. Operators

- **Arithmetic**:

```java
int sum = 10 + 5;      // Addition
int difference = 10 - 5; // Subtraction
int product = 10 * 5;   // Multiplication
int quotient = 10 / 5;  // Division
int remainder = 10 % 5; // Modulus
```

- **Comparison**:

```java
boolean isEqual = (10 == 5); // Equal to
boolean isNotEqual = (10 != 5); // Not equal to
boolean isGreater = (10 > 5); // Greater than
boolean isLess = (10 < 5); // Less than
boolean isGreaterOrEqual = (10 >= 5); // Greater than or equal to
boolean isLessOrEqual = (10 <= 5); // Less than or equal to
```

- **Logical**:

```
boolean resultAnd = (true && false); // Logical AND
boolean resultOr = (true || false);  // Logical OR
boolean resultNot = !true;           // Logical NOT
```

- **Bitwise**:

```
int resultAnd = 5 & 3; // Bitwise AND
int resultOr = 5 | 3;  // Bitwise OR
int resultXor = 5 ^ 3; // Bitwise XOR
int resultNot = ~5;    // Bitwise NOT
int resultLeftShift = 5 << 1;  // Left shift
int resultRightShift = 5 >> 1; // Right shift
int resultUnsignedRightShift = 5 >>> 1; // Unsigned right shift
```

- **Ternary**:

```
int max = (a > b) ? a : b; // If a is greater than b, then assign a to max, else b to max
```

---

5. Control Flow

If-Else:

```
if (condition) {
    // Code
} else if (anotherCondition) {
    // Code
} else {
    // Code
}
```

Switch:

```
switch (variable) {
    case value1:
        // Code
        break;
    case value2:
        // Code
        break;
    default:
        // Code
}
```

Loops:

- **For Loop**:

```
for (int i = 0; i < 10; i++) {
    // Code
}
```

- **Enhanced For Loop**:

```
int[] numbers = {1, 2, 3};
for (int num : numbers) { // Explain: For each number in numbers
    System.out.println(num);
}
```

- **While Loop**:

```
while (condition) {
    // Code
}
```

- **Do-While Loop**:

```
do {
    // Code
} while (condition);
```

---

6. Arrays

- **Single-Dimensional Array**:

```
int[] numbers = {1, 2, 3}; // Single-dimensional array
```

- **Multi-Dimensional Array**:

```
int[][] matrix = {{1, 2}, {3, 4}}; // Multi-dimensional array
```

```
int length = numbers.length; // Get the length of the array
Arrays.sort(numbers); // Sort the array
Arrays.fill(numbers, 0); // Fill the array with a specific value
int[] copy = Arrays.copyOf(numbers, numbers.length); // Copy the array
boolean isEqual = Arrays.equals(numbers, copy); // Check if two arrays are equal
String arrayString = Arrays.toString(numbers); // Convert the array to a string
```

---

7. Methods

- **Method Declaration**:

```
public static int add(int a, int b) { // Adds two integers
    return a + b;
}
```

- **Var_args (Variable Arguments)**:

```
public static int sum(int... numbers) { // Variable number of arguments
    int total = 0;
    for (int num : numbers) {
        total += num;
    }
    return total;
}
```

- **Method Overloading**:

```
public int add(int a, int b) {
    return a + b;
}
public double add(double a, double b) {
    return a + b;
}
```

---

8. Classes and Objects

- **Class Definition**:

```
class Dog {
    // Fields
    String name;
    int age;

    // Constructor
    public Dog(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Method
    public void bark() {
        System.out.println("Woof!");
    }
}
```

- **Object Creation**:

```
Dog myDog = new Dog("Buddy", 3);
myDog.bark();
```

---

9. Inheritance

- **Parent Class**:

```
class Animal {
    void eat() {
        System.out.println("Eating...");
    }
}
```

- **Child Class**:

```
class Dog extends Animal {
    void bark() {
        System.out.println("Barking...");
    }
}
```

- **Super Keyword**:

```
class Dog extends Animal {
    void eat() {
        super.eat(); // Call parent method
        System.out.println("Dog is eating...");
    }
}
```

---

10. Polymorphism

- **Method Overriding**:

```
class Animal {
    void sound() {
        System.out.println("Animal sound");
    }
}

class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Bark");
    }
}
```

---

11. Encapsulation

- **Private Fields with Getters/Setters**:

```
class Person {
    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

---

12. Abstraction

- **Abstract Class**:

```
abstract class Shape {
    abstract void draw();
}

class Circle extends Shape {
    @Override
    void draw() {
        System.out.println("Drawing Circle");
    }
}
```

- **Interface**:

```
interface Drawable {
    void draw();
}

class Circle implements Drawable {
    @Override
    public void draw() {
        System.out.println("Drawing Circle");
    }
}
```

---

13. Exception Handling

- **Try-Catch-Finally**:

```
try {
    int result = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Cannot divide by zero");
} finally {
    System.out.println("This will always execute");
}
```

- **Custom Exception**:

```
class MyException extends Exception {
    public MyException(String message) {
        super(message);
    }
}
```

---

14. Collections Framework

- **ArrayList**:

```
import java.util.ArrayList;

ArrayList<String> list = new ArrayList<>();
list.add("Java");
list.add("Python");
System.out.println(list.get(0));
```

- **HashMap**:

```
import java.util.HashMap;

HashMap<String, Integer> map = new HashMap<>();
map.put("Java", 1);
map.put("Python", 2);
System.out.println(map.get("Java"));
```

---

15. Generics

- **Generic Class**:

```
class Box<T> { // Box is a generic class with type parameter T
    private T item;

    public void setItem(T item) {
        this.item = item;
    }

    public T getItem() {
        return item;
    }
}
```

---

16. Lambda Expressions

- **Functional Interface**:

```
interface Greeting {
    void greet(String message);
}

// Lambda expression
Greeting greeting = (message) -> System.out.println(message);
greeting.greet("Hello, Lambda!");
```

---

17. Streams API

- **Filter and Map**:

```
import java.util.Arrays;
import java.util.List;

List<String> languages = Arrays.asList("Java", "Python", "C++");
languages.stream()
        .filter(lang -> lang.startsWith("J"))
        .map(String::toUpperCase)
        .forEach(System.out::println);
```

---

18. Annotations

- **Built-in Annotations**:

```
@Override
public String toString() {
    return "This is an overridden method";
}
```

---

19. Multithreading

- **Thread Creation**:

```
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread is running");
    }
}

MyThread thread = new MyThread();
thread.start();
```

---

20. File I/O

- **Reading from a File**:

```
import java.io.File;
import java.util.Scanner;

try {
    File file = new File("file.txt");
    Scanner scanner = new Scanner(file);
    while (scanner.hasNextLine()) {
        System.out.println(scanner.nextLine());
    }
    scanner.close();
} catch (IOException e) {
    e.printStackTrace();
}
```