Complete Java Language Guide (Basic to Advanced)

1. Introduction to Java

Java is a high-level, object-oriented, platform-independent programming language. It was developed by Sun Microsystems (now owned by Oracle). Java applications are compiled into bytecode which runs on the Java Virtual Machine (JVM).

2. Hello World Program

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

3. Data Types and Variables

Java supports primitive types (int, char, boolean, float, etc.) and non-primitive types (String, arrays, objects).

Example:

```
int age = 30;
String name = "John";
boolean isJavaFun = true;
```

4. Operators

Operators are special symbols used to perform operations:

Arithmetic: + - * / %

Relational: > < >= <= == !=

Logical: && || !

Assignment: = += -= *= /= %=

5. Conditional Statements

Used to make decisions in code.

```
if (condition) {
    // code
} else {
    // code
}
switch (expression) {
  case value: break; default:
}
```

6. Loops

Loops repeat a block of code.

For Loop:

```
for (int i = 0; i < 5; i++) { }
```

While Loop:

```
while (condition) { }
```

Do-While:

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

7. Arrays

Arrays store multiple values of the same data type.

```
int[] nums = {1, 2, 3};
String[] names = new String[3];
```

8. Object-Oriented Programming (OOP)

Java follows OOP principles:

- Class: Blueprint for objects

- Object: Instance of class
- Inheritance: Reuse code
- Polymorphism: Many forms
- Abstraction: Hiding details
- Encapsulation: Data protection

9. Classes and Objects

```
class Car {
    String model;
    void drive() {
        System.out.println("Driving");
    }
}
Car c = new Car();
c.drive();
```

10. Constructors

Constructor initializes objects when created.

```
class Person {
    String name;
    Person(String n) {
        name = n;
    }
}
```

11. Inheritance

Allows one class to inherit fields and methods from another.

```
class Animal { void eat() {} }
class Dog extends Animal { void bark() {} }
```

12. Polymorphism

Polymorphism allows methods to behave differently based on object type.

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

class Circle extends Shape { void draw() {} }
```

13. Encapsulation

Encapsulation hides data using private variables and provides access via public methods.

```
class Student {

    private int age;

    public void setAge(int a) { age = a; }

    public int getAge() { return age; }

}
```

14. Abstraction and Interfaces

Abstraction hides complex implementation details.

```
abstract class Animal { abstract void sound(); }

interface Flyable { void fly(); }
```

15. Exception Handling

Used to handle runtime errors using try-catch blocks.

```
try {

    int a = 10 / 0;

} catch (ArithmeticException e) {

    System.out.println("Error");

}
```

16. File Handling

Used to read/write data to files.

```
FileWriter writer = new FileWriter("file.txt");

writer.write("Hello");

writer.close();
```

17. Multithreading

Multithreading allows concurrent execution of two or more threads.

```
class MyThread extends Thread {

    public void run() {

        System.out.println("Running");

    }

}
```

18. Java Collections Framework

Provides classes like ArrayList, LinkedList, HashMap to store groups of objects.

```
ArrayList<String> list = new ArrayList<>();

HashMap<String, Integer> map = new HashMap<>();
```

19. Java 8 Features

Java 8 introduced Lambda expressions, Stream API, Functional Interfaces.

```
Runnable r = () -> System.out.println("Run");
```

20. JDBC (Java Database Connectivity)

JDBC connects Java with databases.

```
Connection con = DriverManager.getConnection(url, user, pass);

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery("SELECT * FROM users");
```