Step 1: Setting Up the Environment

1. **Install Java Development Kit (JDK):**
   - Download and install the latest JDK from the Oracle website.
2. **Install an Integrated Development Environment (IDE):**
   - Popular IDEs for Java include IntelliJ IDEA, Eclipse, and NetBeans. Download and install one.

Step 2: Understanding Basic Syntax

1. **Hello World Program:**

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

   - Compile: `javac HelloWorld.java`
   - Run: `java HelloWorld`

Step 2: Understanding Basic Syntax

1. **Java Program Structure:** Every Java program has at least one class, and the `main` method inside a class is the entry point of any Java application.

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

2. **Comments:** Comments are used to explain code and are ignored by the compiler.
   - **Single-line Comment:**

```
// This is a single-line comment
System.out.println("Single-line comment");
```

   - **Multi-line Comment:**

```
/*
 This is a multi-line comment.
 It can span multiple lines.
*/
System.out.println("Multi-line comment");
```

3. **Data Types and Variables:** Java is a strongly-typed language, which means every variable must be declared with a data type.
   - **Primitive Data Types:**

```
int age = 25;
double salary = 85000.50;
char grade = 'A';
```

```
boolean isEmployed = true;
```

- **Non-Primitive Data Types:**

```
String name = "John";
int[] numbers = {1, 2, 3, 4, 5};
```

4. **Basic Output:** Using `System.out.println` to print output to the console.

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
        int age = 25;
        System.out.println("Age: " + age);
    }
}
```

Basic Output Methods in Java

1. **print:**
   - The `print` method outputs text to the console without a newline at the end. Subsequent calls to `print` will continue on the same line.

```
public class Main {
    public static void main(String[] args) {
        System.out.print("Hello, ");
        System.out.print("World!");
        // Output: Hello, World!
    }
}
```

2. **println:**
   - The `println` method outputs text to the console followed by a newline. Each call to `println` will print the text on a new line.

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
        System.out.println("This is a new line.");
        // Output:
        // Hello, World!
        // This is a new line.
    }
}
```

3. **printf:**
   - The `printf` method is used for formatted output. It allows you to format strings with placeholders for variables and specify the format of the output.

```
public class Main {
    public static void main(String[] args) {
        String name = "Alice";
        int age = 30;
        double salary = 85000.75;

        // Using printf for formatted output
```

```
            System.out.printf("Name: %s, Age: %d, Salary: %.2f\n", name,
    age, salary);
            // Output: Name: Alice, Age: 30, Salary: 85000.75
        }
    }
```

Detailed Examples

Example 1: Using `print`

```
public class Main {
    public static void main(String[] args) {
        System.out.print("Java ");
        System.out.print("is ");
        System.out.print("fun!");
        // Output: Java is fun!
    }
}
```

Example 2: Using `println`

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Learning Java");
        System.out.println("is fun!");
        // Output:
        // Learning Java
        // is fun!
    }
}
```

Example 3: Using `printf`

```
public class Main {
    public static void main(String[] args) {
        String name = "Bob";
        int age = 25;
        double score = 95.5;

        System.out.printf("Name: %s, Age: %d, Score: %.1f\n", name, age,
score);
        // Output: Name: Bob, Age: 25, Score: 95.5
    }
}
```

Example 4: Combining `print`, `println`, and `printf`

```
public class Main {
    public static void main(String[] args) {
        // Using print
        System.out.print("Hello ");
        System.out.print("World!");
        // Output: Hello World!

        // Using println
        System.out.println();
        System.out.println("Welcome to Java programming.");
        // Output:
        // Welcome to Java programming.

        // Using printf
        String course = "Java";
```

```
        int duration = 4;
        System.out.printf("Course: %s, Duration: %d weeks\n", course,
duration);
        // Output: Course: Java, Duration: 4 weeks
    }
}
```

Formatting with `printf`

`printf` is particularly powerful because it allows detailed formatting options:

- **String (`%s`):** Prints a string.
- **Integer (`%d`):** Prints an integer.
- **Floating-point (`%f`):** Prints a floating-point number.
- **Character (`%c`):** Prints a character.
- **Percentage (`%%`):** Prints a literal percentage sign.

Examples:

```
public class Main {
    public static void main(String[] args) {
        // String formatting
        System.out.printf("Hello, %s!\n", "Alice");

        // Integer formatting
        System.out.printf("Age: %d\n", 30);

        // Floating-point formatting
        System.out.printf("Salary: %.2f\n", 85000.75);

        // Multiple placeholders
        String product = "Laptop";
        int quantity = 5;
        double price = 999.99;
        System.out.printf("Product: %s, Quantity: %d, Price: %.2f each\n",
product, quantity, price);
        // Output: Product: Laptop, Quantity: 5, Price: 999.99 each
    }
}
```

5. **Basic Input:** Using `Scanner` class to get user input.

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = scanner.nextLine();
        System.out.println("Hello, " + name);
    }
}
```

Basic Input with Scanner

1. **Setting Up the Scanner:**
    - To use the `Scanner` class, you need to import it at the beginning of your program.
    - Create an instance of `Scanner` to read input from the standard input stream (keyboard).

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Code to read input goes here
    }
}
```

Examples

Example 1: Reading a String

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = scanner.nextLine();
        System.out.println("Hello, " + name);
    }
}
```

Example 2: Reading an Integer

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter your age: ");
        int age = scanner.nextInt();
        System.out.println("You are " + age + " years old.");
    }
}
```

Example 3: Reading a Double

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter your salary: ");
        double salary = scanner.nextDouble();
        System.out.println("Your salary is $" + salary);
    }
}
```

Example 4: Reading Multiple Values

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = scanner.nextLine();
        System.out.print("Enter your age: ");
        int age = scanner.nextInt();
        System.out.print("Enter your GPA: ");
        double gpa = scanner.nextDouble();

        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("GPA: " + gpa);
    }
}
```

Example 5: Reading a Character

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter your initial: ");
        char initial = scanner.next().charAt(0);
        System.out.println("Your initial is " + initial);
    }
}
```

Example 6: Reading a Boolean

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Are you a student (true/false): ");
        boolean isStudent = scanner.nextBoolean();
        if (isStudent) {
            System.out.println("You are a student.");
        } else {
            System.out.println("You are not a student.");
        }
    }
}
```

Handling Different Input Types in Sequence

When reading different types of input in sequence, especially after reading a number and then a string, be cautious of the newline character left in the buffer by `nextInt()`, `nextDouble()`, etc. Use `scanner.nextLine()` to clear the buffer.

Example 7: Reading Multiple Types with Buffer Clearing

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter your age: ");
        int age = scanner.nextInt();
        scanner.nextLine(); // Clear the buffer

        System.out.print("Enter your name: ");
        String name = scanner.nextLine();

        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
```

Example with Enhanced Input Validation

To ensure the program does not crash on invalid input, you can add input validation.

Example 8: Enhanced Input Validation

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int age = 0;
        while (true) {
            System.out.print("Enter your age: ");
            if (scanner.hasNextInt()) {
                age = scanner.nextInt();
                scanner.nextLine(); // Clear the buffer
                break;
            } else {
                System.out.println("Invalid input. Please enter a valid
integer.");
                scanner.nextLine(); // Clear the invalid input
            }
        }

        System.out.print("Enter your name: ");
        String name = scanner.nextLine();

        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
```

Step 3: Variables and Data Types

1. **Primitive Data Types:**

```
int number = 10;
char letter = 'A';
boolean isTrue = true;
double decimal = 5.99;
```

2. **Non-Primitive Data Types:**
   - Strings, Arrays, Classes

Data Types and Variables in Java

Java is a statically typed language, which means you must declare the type of a variable before you can use it. There are two categories of data types in Java: primitive data types and non-primitive (reference) data types.

1. Primitive Data Types

Primitive data types are the most basic data types available in Java. There are 8 primitive data types:

1. **byte**:
   - Size: 1 byte (8 bits)
   - Range: -128 to 127

```
public class Main {
    public static void main(String[] args) {
        byte byteVar = 100;
        System.out.println("byte: " + byteVar);
    }
}
```

2. **short**:
   - Size: 2 bytes (16 bits)
   - Range: -32,768 to 32,767

```
public class Main {
    public static void main(String[] args) {
        short shortVar = 10000;
        System.out.println("short: " + shortVar);
    }
}
```

3. **int**:
   - Size: 4 bytes (32 bits)
   - Range: -2^31 to 2^31-1

```
public class Main {
    public static void main(String[] args) {
        int intVar = 100000;
        System.out.println("int: " + intVar);
```

```
    }
}
```

4. **long**:
    - Size: 8 bytes (64 bits)
    - Range: -2^63 to 2^63-1

```
public class Main {
    public static void main(String[] args) {
        long longVar = 100000L;
        System.out.println("long: " + longVar);
    }
}
```

5. **float**:
    - Size: 4 bytes (32 bits)
    - Single-precision 32-bit IEEE 754 floating point
    - Range: approximately ±3.40282347E+38F (6-7 significant decimal digits)

```
public class Main {
    public static void main(String[] args) {
        float floatVar = 10.5f;
        System.out.println("float: " + floatVar);
    }
}
```

6. **double**:
    - Size: 8 bytes (64 bits)
    - Double-precision 64-bit IEEE 754 floating point
    - Range: approximately ±1.79769313486231570E+308 (15 significant decimal digits)

```
public class Main {
    public static void main(String[] args) {
        double doubleVar = 20.99;
        System.out.println("double: " + doubleVar);
    }
}
```

7. **char**:
    - Size: 2 bytes (16 bits)
    - Range: '\u0000' (or 0) to '\uffff' (or 65,535)
    - Used to store characters

```
public class Main {
    public static void main(String[] args) {
        char charVar = 'A';
        System.out.println("char: " + charVar);
    }
}
```

8. **boolean**:
    - Size: 1 bit
    - Values: true or false

```
public class Main {
    public static void main(String[] args) {
        boolean boolVar = true;
        System.out.println("boolean: " + boolVar);
    }
}
```

2. Non-Primitive Data Types

Non-primitive data types (also known as reference types) include classes, arrays, and interfaces. Unlike primitive types, which hold their values directly, reference types store references (addresses) to the actual data.

1. **Strings**:
    - Strings are objects in Java that represent sequences of characters.
    - Unlike primitive types, Strings are immutable (their values cannot be changed once created).

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

Step 4: Operators

1. **Arithmetic Operators:**

```
int a = 10;
int b = 5;
System.out.println(a + b); // 15
```

2. **Comparison Operators:**

```
System.out.println(a > b); // true
```

3. **Logical Operators:**

```
System.out.println(a > b && b < 10); // true
```

Operators: Java provides various operators for different operations.

- **Arithmetic Operators:**

```
int a = 10;
int b = 5;
System.out.println("Addition: " + (a + b));       // 15
System.out.println("Subtraction: " + (a - b));    // 5
System.out.println("Multiplication: " + (a * b)); // 50
System.out.println("Division: " + (a / b));       // 2
System.out.println("Modulus: " + (a % b));        // 0
```

- **Comparison Operators:**

```
System.out.println(a > b);  // true
System.out.println(a < b);  // false
System.out.println(a == b); // false
System.out.println(a != b); // true
System.out.println(a >= b); // true
System.out.println(a <= b); // false
```

- **Logical Operators:**

```
boolean x = true;
boolean y = false;
System.out.println(x && y); // false
System.out.println(x || y); // true
System.out.println(!x);     //
```

Step 5: Control Flow Statements

1. **If-Else:**

```
if (a > b) {
    System.out.println("a is greater than b");
} else {
    System.out.println("a is not greater than b");
}
```

2. **Switch:**

```
int day = 3;
switch (day) {
    case 1:
        System.out.println("Monday");
        break;
    case 2:
        System.out.println("Tuesday");
        break;
    default:
        System.out.println("Other day");
}
```

3. **Loops:**
    - **For Loop:**

```
for (int i = 0; i < 5; i++) {
    System.out.println(i);
}
```

    - **While Loop:**

```
int i = 0;
while (i < 5) {
    System.out.println(i);
    i++;
```

```
        }
```

Control flow statements in Java allow you to dictate the order in which statements are executed based on certain conditions. This includes decision-making statements (`if`, `else if`, `else`, `switch`), looping statements (`for`, `while`, `do-while`), and branching statements (`break`, `continue`, `return`).

Control Flow Statements

1. Decision-Making Statements

if Statement

The `if` statement executes a block of code if its condition evaluates to true.

```
public class Main {
    public static void main(String[] args) {
        int age = 18;

        if (age >= 18) {
            System.out.println("You are an adult.");
        }
    }
}
```

if-else Statement

The `if-else` statement provides an alternative block of code to execute if the condition is false.

```
public class Main {
    public static void main(String[] args) {
        int age = 17;

        if (age >= 18) {
            System.out.println("You are an adult.");
        } else {
            System.out.println("You are a minor.");
        }
    }
}
```

else if Statement

The `else if` statement allows you to check multiple conditions.

```
public class Main {
    public static void main(String[] args) {
        int score = 75;

        if (score >= 90) {
            System.out.println("Grade: A");
        } else if (score >= 80) {
            System.out.println("Grade: B");
        } else if (score >= 70) {
```

```
            System.out.println("Grade: C");
        } else {
            System.out.println("Grade: F");
        }
    }
}
```

Nested if-else Statement

You can nest `if-else` statements within each other.

```
public class Main {
    public static void main(String[] args) {
        int num = -5;

        if (num >= 0) {
            if (num == 0) {
                System.out.println("The number is zero.");
            } else {
                System.out.println("The number is positive.");
            }
        } else {
            System.out.println("The number is negative.");
        }
    }
}
```

switch Statement

The `switch` statement evaluates an expression and executes the corresponding `case` block.

```
public class Main {
    public static void main(String[] args) {
        int day = 3;

        switch (day) {
            case 1:
                System.out.println("Monday");
                break;
            case 2:
                System.out.println("Tuesday");
                break;
            case 3:
                System.out.println("Wednesday");
                break;
            case 4:
                System.out.println("Thursday");
                break;
            case 5:
                System.out.println("Friday");
                break;
            case 6:
                System.out.println("Saturday");
                break;
            case 7:
                System.out.println("Sunday");
                break;
            default:
```

```
                System.out.println("Invalid day");
        }
    }
}
```

2. Looping Statements

for Loop

The `for` loop is used when you know in advance how many times you want to execute a statement or a block of statements.

```
public class Main {
    public static void main(String[] args) {
        for (int i = 0; i < 5; i++) {
            System.out.println("i: " + i);
        }
    }
}
```

Enhanced for Loop

The enhanced `for` loop (also known as the for-each loop) is used to iterate over arrays or collections.

```
public class Main {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5};

        for (int num : numbers) {
            System.out.println("Number: " + num);
        }
    }
}
```

while Loop

The `while` loop is used to execute a block of statements repeatedly as long as a given condition is true.

```
public class Main {
    public static void main(String[] args) {
        int i = 0;

        while (i < 5) {
            System.out.println("i: " + i);
            i++;
        }
    }
}
```

do-while Loop

The `do-while` loop is similar to the `while` loop, but it executes the block of code at least once before checking the condition.

```
public class Main {
    public static void main(String[] args) {
        int i = 0;

        do {
            System.out.println("i: " + i);
            i++;
        } while (i < 5);
    }
}
```

3. Branching Statements

break Statement

The `break` statement terminates the innermost loop or switch statement.

```
public class Main {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            if (i == 5) {
                break; // Exit the loop when i is 5
            }
            System.out.println("i: " + i);
        }
    }
}
```

continue Statement

The `continue` statement skips the current iteration of the loop and moves to the next iteration.

```
public class Main {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            if (i == 5) {
                continue; // Skip the rest of the loop body when i is 5
            }
            System.out.println("i: " + i);
        }
    }
}
```

return Statement

The `return` statement exits from the current method and optionally returns a value.

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Sum: " + add(5, 3));
    }

    public static int add(int a, int b) {
        return a + b; // Exit the method and return the sum
    }
}
```

Comprehensive Example

Here's a comprehensive example that combines multiple control flow statements:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number between 1 and 7: ");
        int day = scanner.nextInt();

        // switch statement
        switch (day) {
            case 1:
                System.out.println("Monday");
                break;
            case 2:
                System.out.println("Tuesday");
                break;
            case 3:
                System.out.println("Wednesday");
                break;
            case 4:
                System.out.println("Thursday");
                break;
            case 5:
                System.out.println("Friday");
                break;
            case 6:
                System.out.println("Saturday");
                break;
            case 7:
                System.out.println("Sunday");
                break;
            default:
                System.out.println("Invalid day");
                break;
        }

        // for loop
        for (int i = 0; i < 5; i++) {
            System.out.println("i: " + i);
        }

        // while loop
        int j = 0;
        while (j < 5) {
            System.out.println("j: " + j);
            j++;
        }

        // do-while loop
        int k = 0;
        do {
            System.out.println("k: " + k);
            k++;
        } while (k < 5);

        // break and continue in for loop
```

```
        for (int l = 0; l < 10; l++) {
            if (l == 3) {
                continue; // Skip the rest of the loop body when l is 3
            }
            if (l == 8) {
                break; // Exit the loop when l is 8
            }
            System.out.println("l: " + l);
        }
    }
}
```