Java Syntax and Fundamentals

Table of Contents

Keywords

What are Keywords?

Keywords are the reserved words in Java. We cannot use a keyword as an identifier (e.g., variable name, class name, method name). Keywords are case-sensitive and have predefined meanings in Java. There are 50 reserved keywords in Java.

List of Keywords:

`abstract`, `assert`, `boolean`, `break`, `byte`, `case`, `catch`, `char`, `class`, `const`, `continue`, `default`, `do`, `double`, `else`, `enum`, `extends`, `final`, `finally`, `float`, `for`, `goto`, `if`, `implements`, `import`, `instanceof`, `int`, `interface`, `long`, `native`, `new`, `package`, `private`, `protected`, `public`, `return`, `short`, `static`, `strictfp`, `super`, `switch`, `synchronized`, `this`, `throw`, `throws`, `transient`, `try`, `void`, `volatile`, `while`.

Example of Keywords:

```
class Keywords {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        int c = a + b;
        System.out.println("The sum of a and b is: " + c); // The sum of a and b
is: 30
    }
}
```

In the above example, we have used the keywords `class`, `public`, `static`, `void`, `main`, `String`, and `System.out.println`. These are reserved words in Java and have specific meanings.

Identifiers

What are Identifiers?

Identifiers are the names given to entities like variables, classes, methods, and packages. They are user-defined names consisting of a sequence of characters.

Example of Identifiers:

```
class Identifiers {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        int c = a + b;
        System.out.println(c); // 30
    }
}
```

In the above example, `Identifiers` is a class name, `main` is a method name, and `args`, `a`, `b`, and `c` are variable names. These are user-defined names called identifiers.

Data Types

What are Data Types?

Data types specify the different sizes and values that can be stored in a variable. There are two types of data types in Java:

1. **Primitive Data Types:** `boolean`, `char`, `byte`, `short`, `int`, `long`, `float`, `double`
2. **Non-primitive Data Types:** `Classes, Interfaces, Arrays`

Example of Data Types:

```
class DataTypes {
    public static void main(String[] args) {
        int a = 10;
        float b = 10.5f;
        char c = 'A';
        boolean d = true;
        System.out.println("The value of a is: " + a); // The value of a is: 10
        System.out.println("The value of b is: " + b); // The value of b is: 10.5
        System.out.println("The value of c is: " + c); // The value of c is: A
        System.out.println("The value of d is: " + d); // The value of d is: true
    }
}
```

In the above example, we have used the `int`, `float`, `char`, and `boolean` data types. These are primitive data types in Java.

- The `int` data type is used to store integer values.
- The `float` data type is used to store floating-point values.
- The `char` data type is used to store character values.
- The `boolean` data type is used to store boolean values.

Operators

Types of Operators:

1. **Arithmetic Operators:** `+`, `-`, `*`, `/`, `%`
2. **Relational Operators:** `==`, `!=`, `>`, `<`, `>=`, `<=`
3. **Logical Operators:** `&&`, `||`, `!`
4. **Bitwise Operators:** `&`, `|`, `^`, `~`, `<<`, `>>`, `>>>`

Example of Operators:

```
class Operators {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;

        // Arithmetic Operators
        System.out.println("a + b = " + (a + b)); // 30
        System.out.println("a - b = " + (a - b)); // -10

        // Relational Operators
        System.out.println("a == b: " + (a == b)); // false
        System.out.println("a != b: " + (a != b)); // true

        // Logical Operators
        System.out.println("(a > 5) && (b > 15): " + ((a > 5) && (b > 15))); //
true
        System.out.println("(a < 5) || (b > 15): " + ((a < 5) || (b > 15))); //
true

        // Bitwise Operators
        System.out.println("a & b: " + (a & b)); // 0
        System.out.println("a | b: " + (a | b)); // 30
    }
}
```

Input/Output Operations

Using Scanner for Input

The `Scanner` class in Java is part of the `java.util` package and is used to get input of primitive types like `int`, `double`, etc., and strings from the user. It is a convenient way to handle user input for various data types.

To use the `Scanner` class, you need to import it at the beginning of your Java file:

```
import java.util.Scanner;
```

Example of Input/Output Operations

Basic Example:

This example demonstrates how to take integer and string input from the user and print it.

```
import java.util.Scanner;

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

        // Taking integer input
        System.out.print("Enter an integer: ");
        int num = scanner.nextInt();
        System.out.println("You entered: " + num);

        // Taking string input
        System.out.print("Enter a string: ");
        String str = scanner.next();
        System.out.println("You entered: " + str);

        scanner.close();
    }
}
```

In this example:

- `scanner.nextInt()` is used to read an integer value from the user.
- `scanner.next()` is used to read a single word string input from the user.
- `System.out.println()` is used to print the output to the console.

Detailed Example with Multiple Data Types:

This example demonstrates how to take various types of input from the user, including integers, doubles, and strings.

```
import java.util.Scanner;

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

        // Taking integer input
        System.out.print("Enter an integer: ");
        int num = scanner.nextInt();
        System.out.println("You entered integer: " + num);

        // Taking double input
        System.out.print("Enter a double: ");
```

```
        double dbl = scanner.nextDouble();
        System.out.println("You entered double: " + dbl);

        // Taking string input (single word)
        System.out.print("Enter a single word string: ");
        String str1 = scanner.next();
        System.out.println("You entered single word string: " + str1);

        // Taking string input (whole line)
        scanner.nextLine(); // consume the leftover newline character
        System.out.print("Enter a full line string: ");
        String str2 = scanner.nextLine();
        System.out.println("You entered full line string: " + str2);

        scanner.close();
    }
}
```

In this example:

- `scanner.nextDouble()` is used to read a double value from the user.
- `scanner.nextLine()` is used to read an entire line of text from the user. Note that we call `scanner.nextLine()` after `scanner.next()` to consume the newline character left behind.

Example with Multiple Inputs in One Line:

This example demonstrates how to take multiple inputs in one line separated by spaces.

```
import java.util.Scanner;

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

        // Taking multiple inputs in one line
        System.out.print("Enter an integer, a double, and a string: ");
        int num = scanner.nextInt();
        double dbl = scanner.nextDouble();
        String str = scanner.next();

        System.out.println("You entered integer: " + num);
        System.out.println("You entered double: " + dbl);
        System.out.println("You entered string: " + str);

        scanner.close();
    }
}
```

In this example:

- The user can enter an integer, a double, and a string separated by spaces.
- The `Scanner` class reads each input one by one in the order they are entered.

Using BufferedReader for Input

An alternative to `Scanner` is `BufferedReader` from the `java.io` package, which can be more efficient for reading large amounts of input.

Example of BufferedReader:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

class BufferedReaderExample {
    public static void main(String[] args) throws IOException {
        BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));

        // Taking integer input
        System.out.print("Enter an integer: ");
        int num = Integer.parseInt(reader.readLine());
        System.out.println("You entered: " + num);

        // Taking string input
        System.out.print("Enter a string: ");
        String str = reader.readLine();
        System.out.println("You entered: " + str);
    }
}
```

In this example:

- `BufferedReader` is used to read text from an input stream (the console in this case).
- `reader.readLine()` is used to read a line of text.
- `Integer.parseInt()` converts the string input to an integer.

Using System.out for Output

The `System.out.println()` and `System.out.print()` methods are used to print output to the console.

- `System.out.print()` prints the text without a newline.
- `System.out.println()` prints the text with a newline.

Example:

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

```
        System.out.println("Welcome to Java.");
    }
}
```

In this example:

- `System.out.print("Hello, ")` prints `Hello,` without a newline.
- `System.out.println("World!")` prints `World!` with a newline.
- `System.out.println("Welcome to Java.")` prints `Welcome to Java.` with a newline.

By using these methods, you can handle a variety of input and output operations in Java, making your programs interactive and user-friendly.

By covering these topics, you will have a solid foundation in Java syntax and fundamentals. Each section includes explanations and code examples with expected output to help you understand the concepts better.