The `switch` statement in Java is used to execute one set of statements from multiple conditions. It acts as an alternative to using multiple `if-else` statements when you need to perform different actions based on the value of a single variable. The `switch` statement is often used with primitive data types, enumerations, and, since Java 7, `String` values.

Syntax of `switch` statement

```
switch (expression) {
    case value1:
        // code to be executed if expression == value1
        break;
    case value2:
        // code to be executed if expression == value2
        break;
    // more cases...
    default:
        // code to be executed if none of the cases match
}
```

- **`expression`**: This is the variable being compared. It must return a value of `int`, `byte`, `short`, `char`, `String`, or an enum type.
- **`case`**: Each `case` specifies a possible value for the expression. If the expression matches a case value, the code in that case will execute.
- **`break`**: Ends the `switch` case, so it doesn't execute code in the following cases.
- **`default`**: (optional) Executes if none of the specified cases match the expression.

Example: Day of the Week

This example will display a message based on the day of the week:

```java
public class SwitchExample {
    public static void main(String[] args) {
        int day = 3; // Let's say 3 represents Wednesday

        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;
        }
    }
}
```

Explanation:

- If `day = 3`, the `switch` statement will match with `case 3`, printing "Wednesday."
- If you remove `break;` after `case 3`, Java will execute all code after that case

until it reaches a `break` or the end of the switch block. This is called **fall-through behavior**.

Using `String` in a `switch`

In Java 7 and later, you can also use `String` values with `switch`, which can be useful for checking commands or options:

```
public class SwitchStringExample {
    public static void main(String[] args) {
        String fruit = "apple";

        switch (fruit) {
            case "apple":
                System.out.println("Apple is red or green.");
                break;
            case "banana":
                System.out.println("Banana is yellow.");
                break;
            case "orange":
                System.out.println("Orange is orange.");
                break;
            default:
                System.out.println("Unknown fruit.");
                break;
        }
    }
}
```

Key Points to Remember

1. The `break` statement stops the code from "falling through" to the next case.
2. If no cases match, the `default` block executes (if provided).
3. The `switch` statement is helpful for simplifying multiple `if-else` conditions based on a single expression.

Scenario: Grading System

Suppose you want to assign a grade based on a student's score. You might use `if-else` statements like this:

```
int score = 85;
char grade;

if (score >= 90) {
    grade = 'A';
} else if (score >= 80) {
    grade = 'B';
} else if (score >= 70) {
    grade = 'C';
} else if (score >= 60) {
    grade = 'D';
} else {
    grade = 'F';
}

System.out.println("Grade: " + grade);
```

Converting `if-else` to `switch`

In this case, you can't directly use `switch` with ranges (e.g., `score >= 90`), but we can modify the code by grouping scores into ranges.

One way is to map each range to a single integer value and use a `switch` statement on that:

```java
int score = 85;
char grade;

switch (score / 10) {
    case 10: // fall-through for perfect scores
    case 9:
        grade = 'A';
        break;
    case 8:
        grade = 'B';
        break;
    case 7:
        grade = 'C';
        break;
    case 6:
        grade = 'D';
        break;
    default:
        grade = 'F';
        break;
}

System.out.println("Grade: " + grade);
```

Explanation

- Here, we use `score / 10` as the expression for `switch`, which allows us to group ranges:
    - Scores in the 90s and 100 map to `case 9` and `case 10`, respectively.
    - Scores in the 80s map to `case 8`, and so on.
- This solution allows you to avoid multiple `if-else` statements while handling ranges in a `switch`.

Another Example: Menu Selection

Imagine a menu-driven program where the user selects an option (1, 2, or 3). Here's how you might write it using `if-else` statements:

```
int option = 2;

if (option == 1) {
    System.out.println("Option 1 selected: View balance.");
} else if (option == 2) {
    System.out.println("Option 2 selected: Deposit funds.");
} else if (option == 3) {
    System.out.println("Option 3 selected: Withdraw funds.");
} else {
    System.out.println("Invalid option selected.");
}
```

Using `switch` Statement

The code above can be simplified with a `switch` statement:

```
int option = 2;

switch (option) {
    case 1:
        System.out.println("Option 1 selected: View balance.");
        break;
    case 2:
        System.out.println("Option 2 selected: Deposit funds.");
        break;
    case 3:
        System.out.println("Option 3 selected: Withdraw funds.");
        break;
    default:
        System.out.println("Invalid option selected.");
        break;
}
```

Key Takeaways

1. **Readability**: `switch` statements often improve readability when you have multiple conditions based on a single variable.
2. **Grouping Conditions**: In cases like grading, you can sometimes use math (like

`score / 10`) to map ranges to a single case value in `switch`.

3. **Efficient for Fixed Values**: `switch` is very effective for fixed values like specific numbers, characters, or strings.