6. Flow Control

If Else Statements

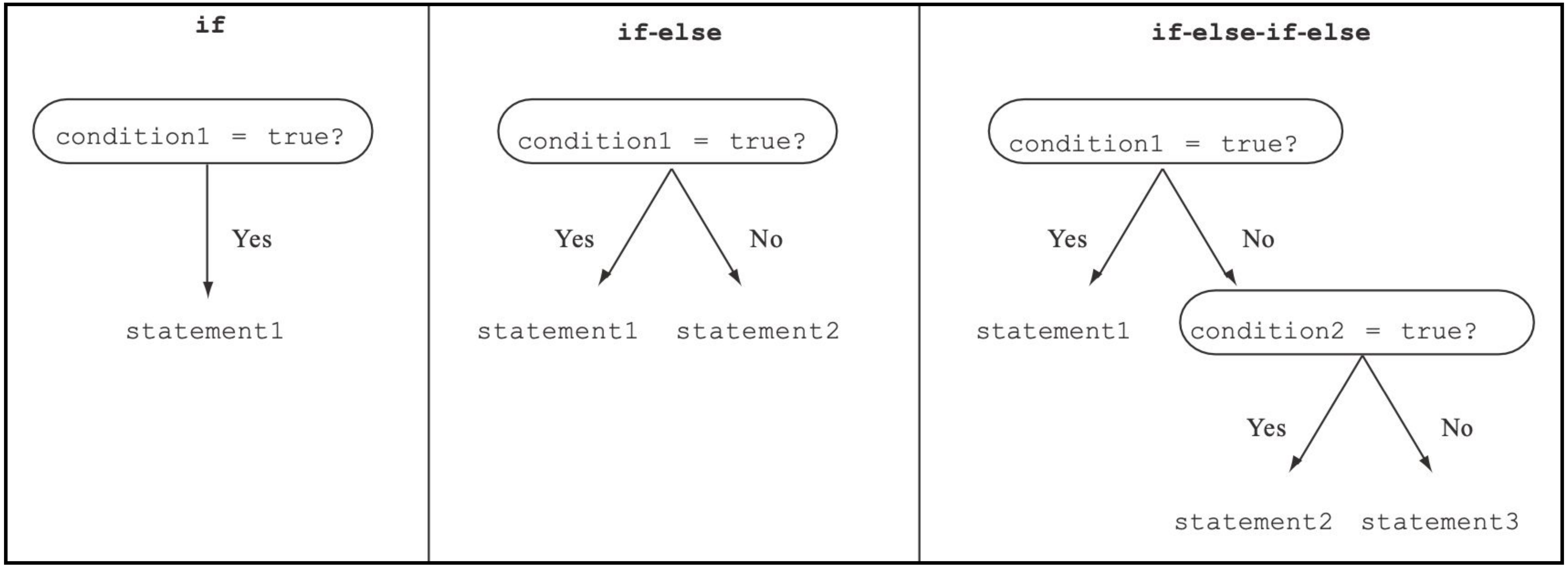
1. The if-else statement is used to make decisions in your code based on certain conditions.
2. It allows you to execute a block of code if a specified condition is true, and another block of code if the condition is false.
3. Syntax:

```
int num = 10;
if (num > 0) {
    System.out.println("Number is positive");
} else {
    System.out.println("Number is non-positive");
}
```

Multiple Conditions

1. You can also use the else if statement to check multiple conditions in sequence:

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


if
condition1 = true?
Yes
statement1
if-else
condition1 = true?
Yes
No
statement1
statement2
if-else-if-else
condition1 = true?
Yes
No
statement1
condition2 = true?
Yes
No
statement2
statement3

if

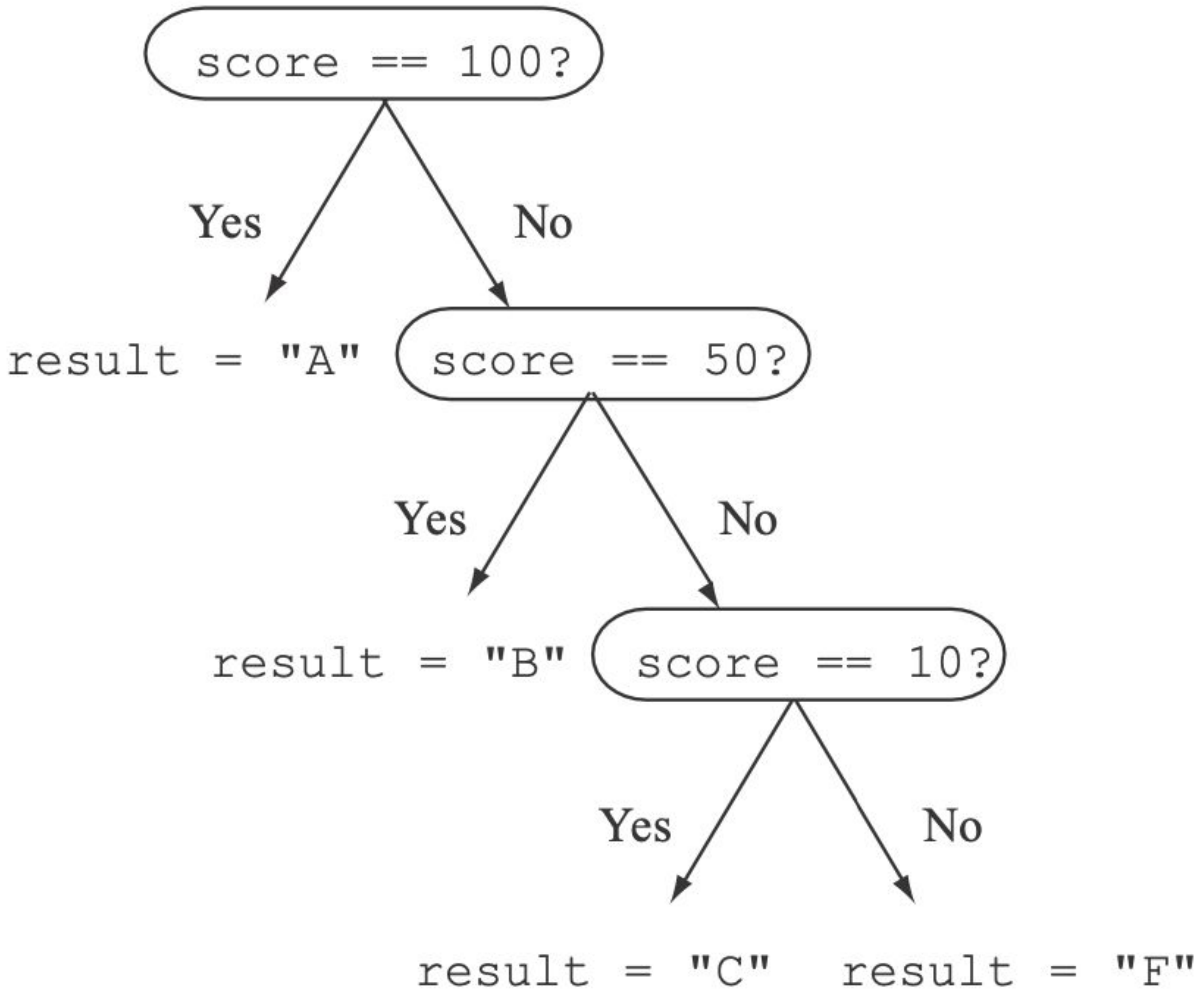
```
if (name.equals("Lion"))
    score = 200;
```

if-else

```
if (name.equals("Lion"))
    score = 200;
else
    score = 300;
```

if-else-if-else

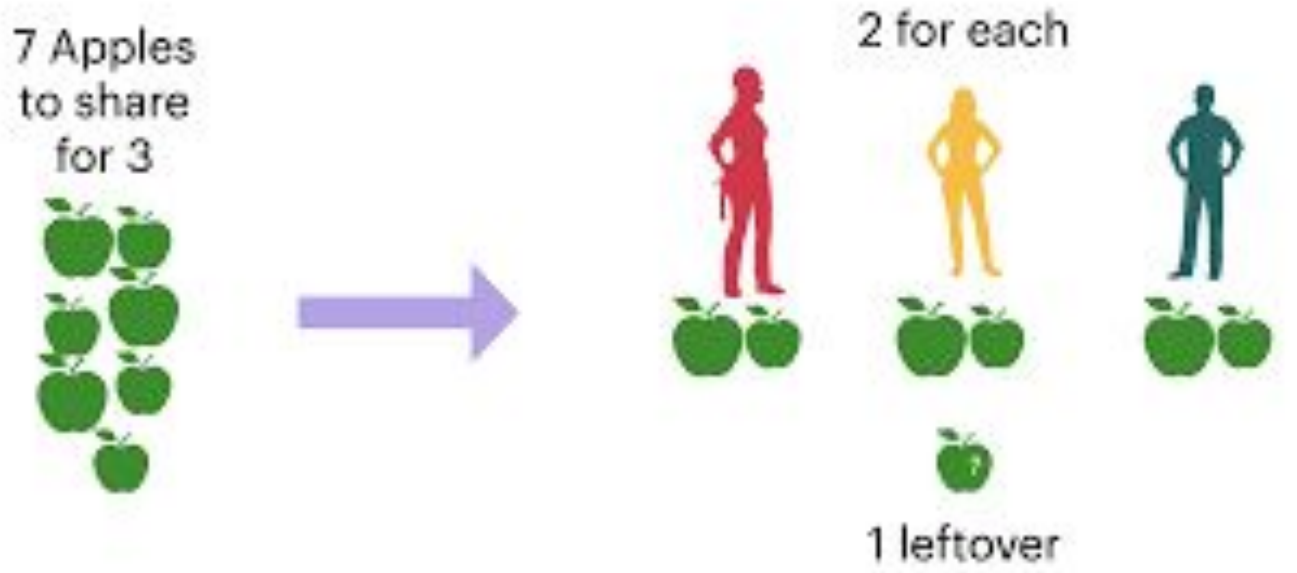
```
if (score == 100)
    result = "A";
else if (score == 50)
    result = "B";
else if (score == 10)
    result = "C";
else
    result = "F";
```


score == 100?
Yes
No
result = "A"
score == 50?
Yes
No
result = "B"
score == 10?
Yes
No
result = "C"
result = "F"

Modulus operator

1. The modulus operator in Java calculates the remainder of a division operation. When you divide one integer by another, the result might not be a whole number. The modulus operator gives you the remainder left over after the division.

2. Lets suppose you have 7 apples and you want to divide them by three, you can use modulus operator to find the reminder(leftover) using this expression 7 % 3, the result would be one.



Nested if-else

1. You can also nest if-else statements within each other to handle more complex conditions:

```
int num = 10;
if (num > 0) {
    if (num % 2 == 0) {
        System.out.println("Number is positive and even");
    } else {
        System.out.println("Number is positive and odd");
    }
} else if (num < 0) {
    System.out.println("Number is negative");
} else {
    System.out.println("Number is zero");
}
```

Short-hand If-Else (Ternary Operator)

1. Java also supports a short-hand version of the if-else statement known as the ternary operator (?:), which can be used to assign a value based on a condition:
2. You can have multiple nested conditions using ternary operator but it’s not recommended since it makes your code harder to understand.

```
int num = 10;

String result = (num > 0) ? "Positive" : "Non-positive";

System.out.println("Number is " + result);
```

Switch

1. The switch statement is used to execute one block of code from multiple options based on the value of an expression.

2. It's a useful alternative to using multiple if-else statements when you have a single expression that you want to compare against multiple possible values.

Switch statement syntax

```
switch (expression) {
    case value1:
        // Code to be executed if expression matches value1
        break;
    case value2:
        // Code to be executed if expression matches value2
        break;
    // Add more cases as needed
    default:
        // Code to be executed if expression doesn't match any case
}
```

Key points about the switch statement

1. The expression must evaluate to a byte, short, char, int, String, or enum.
2. Each case statement specifies a value to compare the expression against.
3. The break statement is used to exit the switch block after a case is executed. If break is omitted, the execution will "fall through" to the next case.
4. The default case is optional and is executed if none of the case values match the expression.
5. In the following example, the switch statement sets the dayName based on the value of dayOfWeek.
    1. If dayOfWeek is 2, the output will be "Day is Monday". If dayOfWeek is not in the range 1-7, the default case will set dayName to "Invalid day".

Example switch statement

```
int dayOfWeek = 2;
String dayName;

switch (dayOfWeek) {
   case 1:
      dayName = "Sunday";
      break;
   case 2:
      dayName = "Monday";
      break;
   case 3:
      dayName = "Tuesday";
      break;
   case 4:
      dayName = "Wednesday";
      break;
   case 5:
      dayName = "Thursday";
      break;
   case 6:
      dayName = "Friday";
      break;
   case 7:
      dayName = "Saturday";
      break;
   default:
      dayName = "Invalid day";
}

System.out.println("Day is " + dayName);
```

Switch Expression (OPTIONAL) Java17

1. In java 17, switch statement are more shorter and concise.
2. You no longer need to use the break keyword within each case block as you would with traditional switch statements. Instead, each case block can be treated as an expression, and the value of the chosen case will b directly returned. Example:

```
int day = 3;
String dayName = switch (day) {
    case 1 -> "Monday";
    case 2 -> "Tuesday";
    case 3 -> "Wednesday";
    case 4 -> "Thursday";
    case 5 -> "Friday";
    case 6, 7 -> "Weekend";
    default -> throw new IllegalArgumentException("Invalid day: " + day);
};
```

if-else statement

```
String day = "SUN";
if (day.equals (“MON") || day.equals (“TUE”) || day.equals ("WED") ||day.equals ("THU"))
   System.out.printin("Time to work");
else if (day.equals ("FRI"))
   System.out.printin("Nearing weekend");
else if (day equals ("SAT") || day equals ("SUN"))
   System.out.println( "Weekend!");
else
   System.out.println("Invalid day?");
```

Switch statement

```
String day = "SUN";
switch (day) {
    case "MON":
    case “TUE":
    case "WED" :
    case "THU": System.out.println("Time to work");
        break;
    case "FRI": System. out.printin( "Nearing weekend" );
        break;
    case "SAT":
    case "SUN": System. out.printin( "Weekend!") ;
        break;
    default: System.out.println("Invalid day?");
    }
```

Comparing a switch statement with if-else

1. The two preceding snippets of code perform the same function of comparing the value of the variable day and printing an appropriate value.

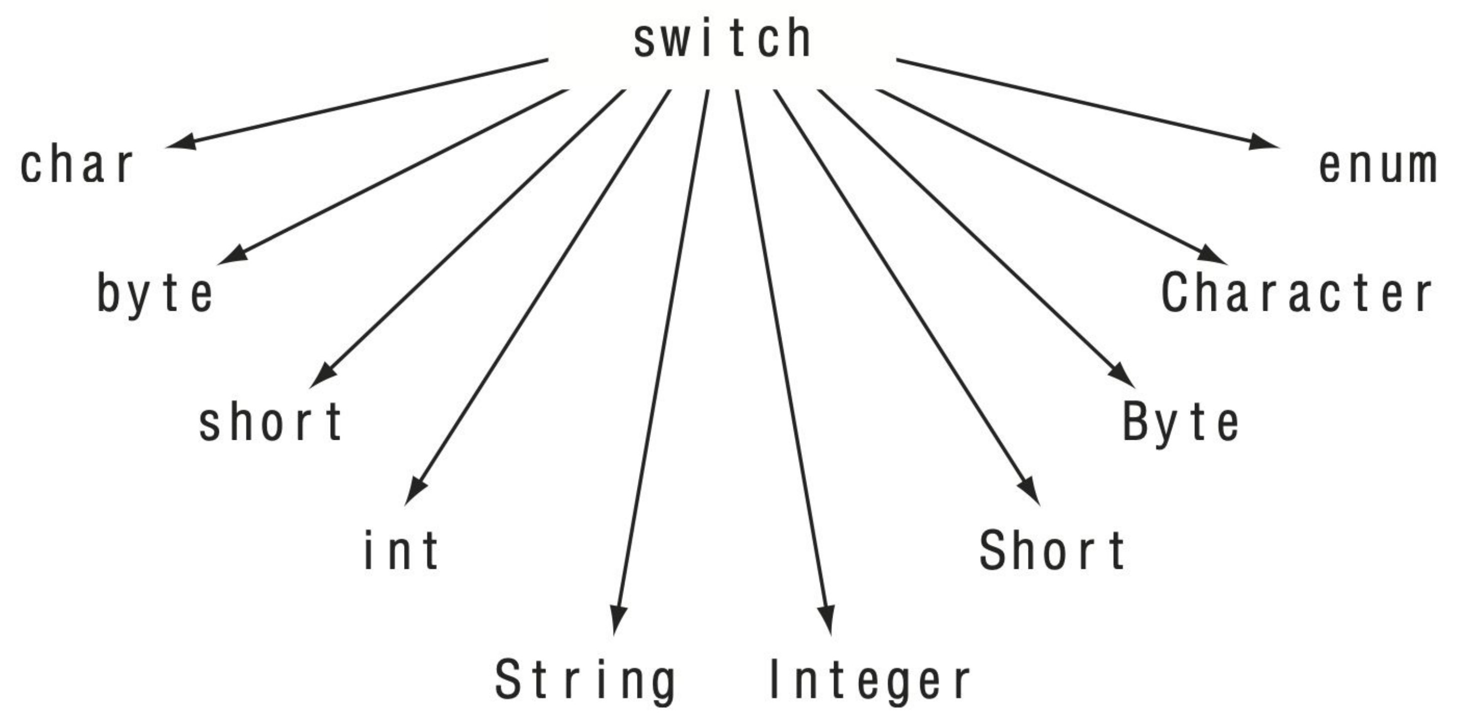
2. But the latter code, which uses the switch statement, is simpler and easier to read and follow.

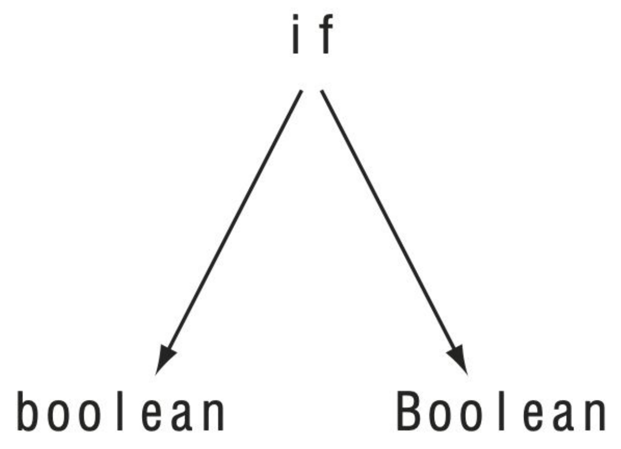
3. Note that the preceding switch statement doesn't define code for all the case values.

4. What happens if the value of the variable day matches TUE?

5. When control of the code enters the label matching TUE in the switch construct, it'll execute all the code until it encounters a break statement or it reaches the end of the switch statement.

Types of arguments that can be passed





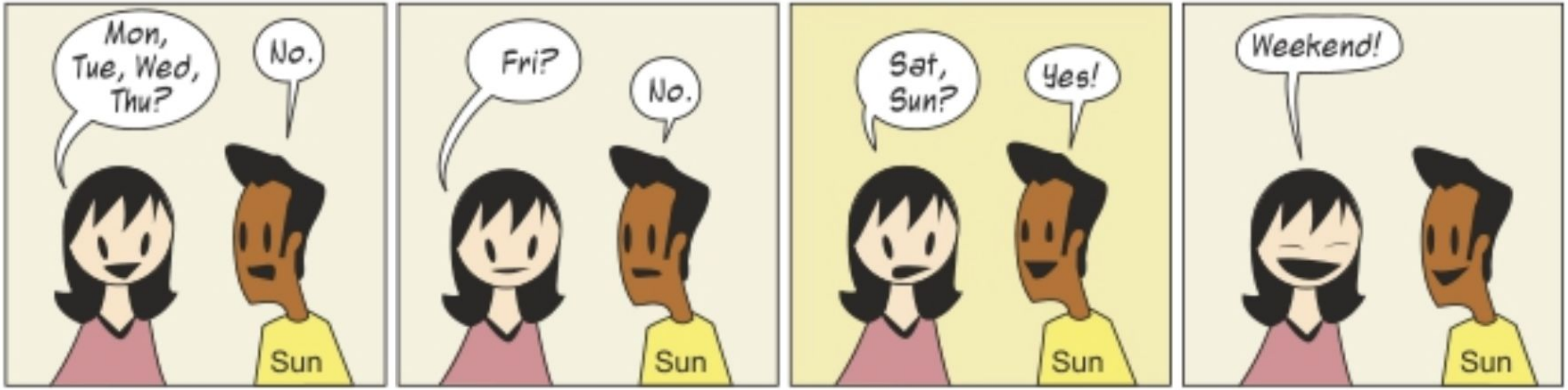
Comparing if-else with switch-case

1. Both if-else statements and switch statements in Java are used for conditional branching, but they have different use cases and syntax.
2. if-else statements:
   1. Used when you have a few conditions to check.
   2. Each if statement can have its own condition.
   3. Can include else if and else blocks to handle multiple conditions.
   4. Conditions can be complex, involving logical operators (&&, ||, etc.).

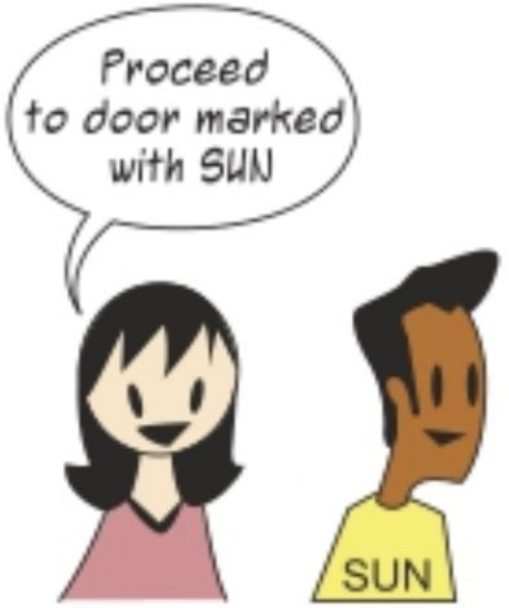
Comparing if-else with switch-case

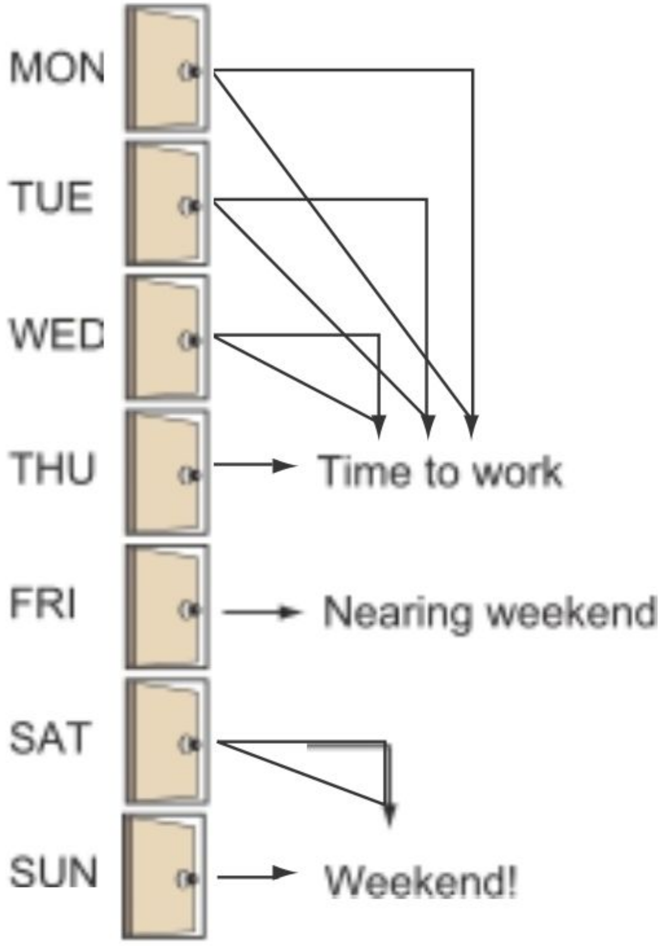
1. Both if-else statements and switch statements in Java are used for conditional branching, but they have different use cases and syntax.
2. switch statements:
    1. Used when you have a single value to compare against multiple possible values.
    2. The value must be a byte, short, int, char, String,or enum.
    3. Each case represents a possible value to compare against.
    4. Must end each case with a break statement to prevent fall-through (executing subsequent cases).
    5. Can include a default case to handle values that do not match any case.

The if-else-if-else construct is like a series of questions and answers



A switch statement is like asking a question and acting on the

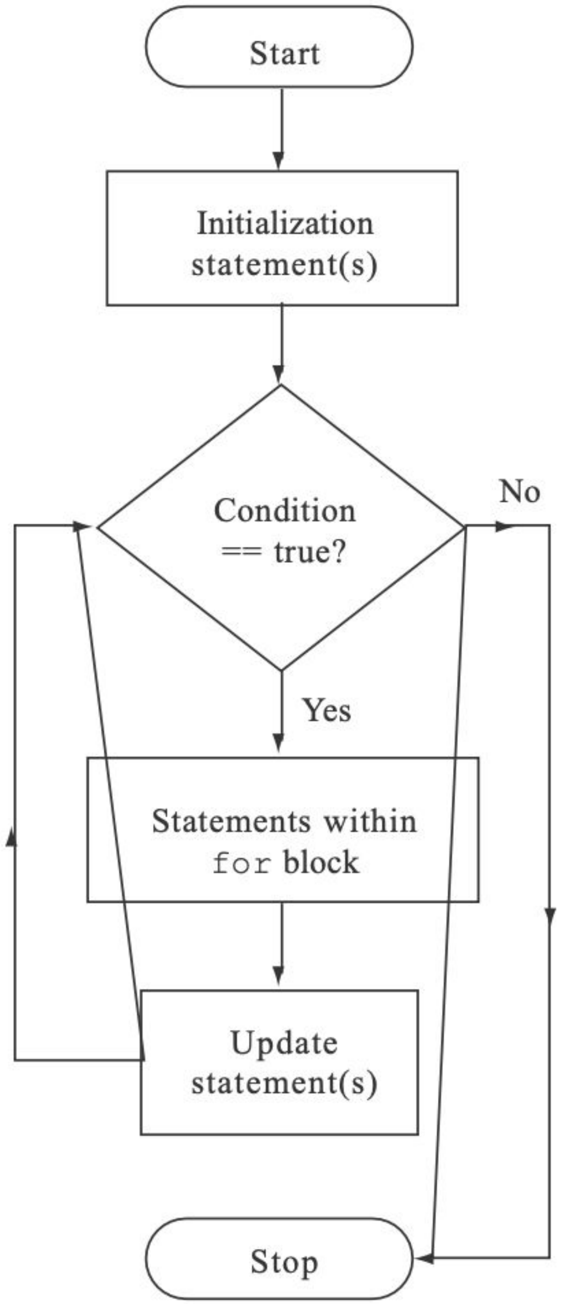




For loop

1. The for loop is used to iterate over a sequence of elements, such as an array or a collection, or to execute a block of code a certain number of times.

2. It provides a concise way to write loops with initialisation, condition checking, and iteration all in one place.



For loop syntax

```
for (initialization; condition; iteration) {
    // Code to be executed
}
```

1. Initialisation - This is where you initialise the loop control variable. It is executed only once when the loop starts.
2. Condition - The condition is evaluated before each iteration. If it evaluates to true, the loop continues. If it evaluates to false, the loop terminates.
3. Iteration - This is where you update the loop control variable. It is executed after each iteration.

For loop example

1. In this example, the for loop will iterate five times, printing "Iteration: 0" to "Iteration: 4" to the console.

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

Enhanced for loop

1. The enhanced for loop, also known as the "for-each" loop, is a special type of loop in Java that is used to iterate over elements in an array or a collection.

2. It provides a more concise and readable way to iterate over such structures compared to traditional for loops.

Enhanced For loop syntax

```
for (element_type element : array_or_collection) {
    // Code to be executed for each element
}
```

1. element_type - The data type of the elements in the array or collection.
2. element - The variable that represents each element in the array or collection.
3. array_or_collection - The array or collection over which to iterate.

Enhanced for loop example with an array

```
int[] numbers = {1, 2, 3, 4, 5};
for (int number : numbers) {
    System.out.println("Number: " + number);
}
```

Enhanced for loop example with a collection (ArrayList)

```
ArrayList<String> names = new ArrayList<>();
names.add("Sam");
names.add("Antonio");
names.add("Ayham”);

for (String name : names) {
    System.out.println("Name: " + name);
}
```

Enhanced for loop

1. In both examples, the enhanced for loop iterates over each element in the array or collection and executes the code block for each element.

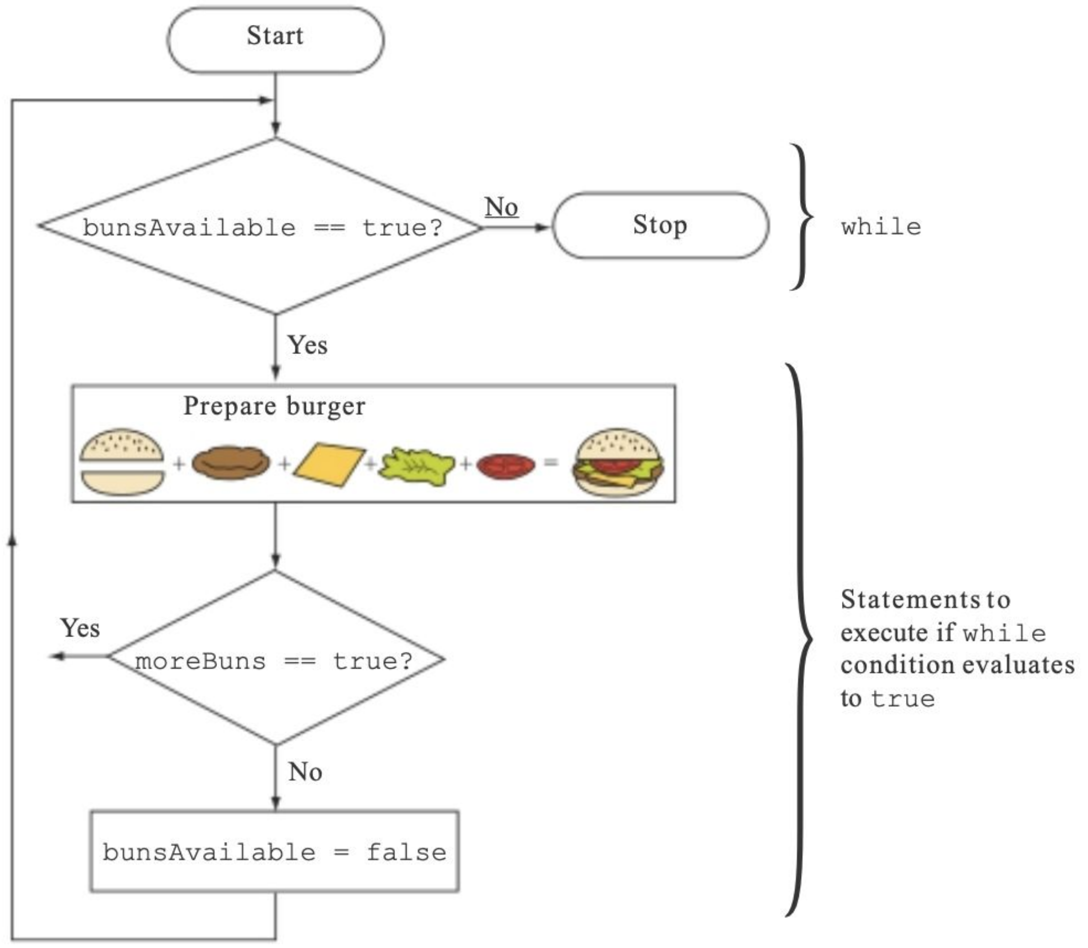
2. The loop variable (number or name) takes on the value of each element in turn.

3. The enhanced for loop simplifies the syntax for iterating over arrays and collections, making the code more readable and less prone to errors.

While loop

1. The while loop is used to repeatedly execute a block of code as long as a given condition remains true.

2. It's useful when you want to execute a block of code an indefinite number of times, based on a condition.



While loop syntax

```
while (condition) {
    // Code to be executed
}
```

1. Condition - The condition is evaluated before each iteration. If it evaluates to true, the code block inside the while loop is executed.
2. If it evaluates to false, the loop terminates, and the program continues with the next statement after the loop.

While loop example

1. In this example, the while loop will iterate five times, printing "Iteration: 0" to "Iteration: 4" to the console.

```
int i = 0;

while (i < 5) {

    System.out.println("Iteration: " + i);

    i++;

}
```

Infinite while loop

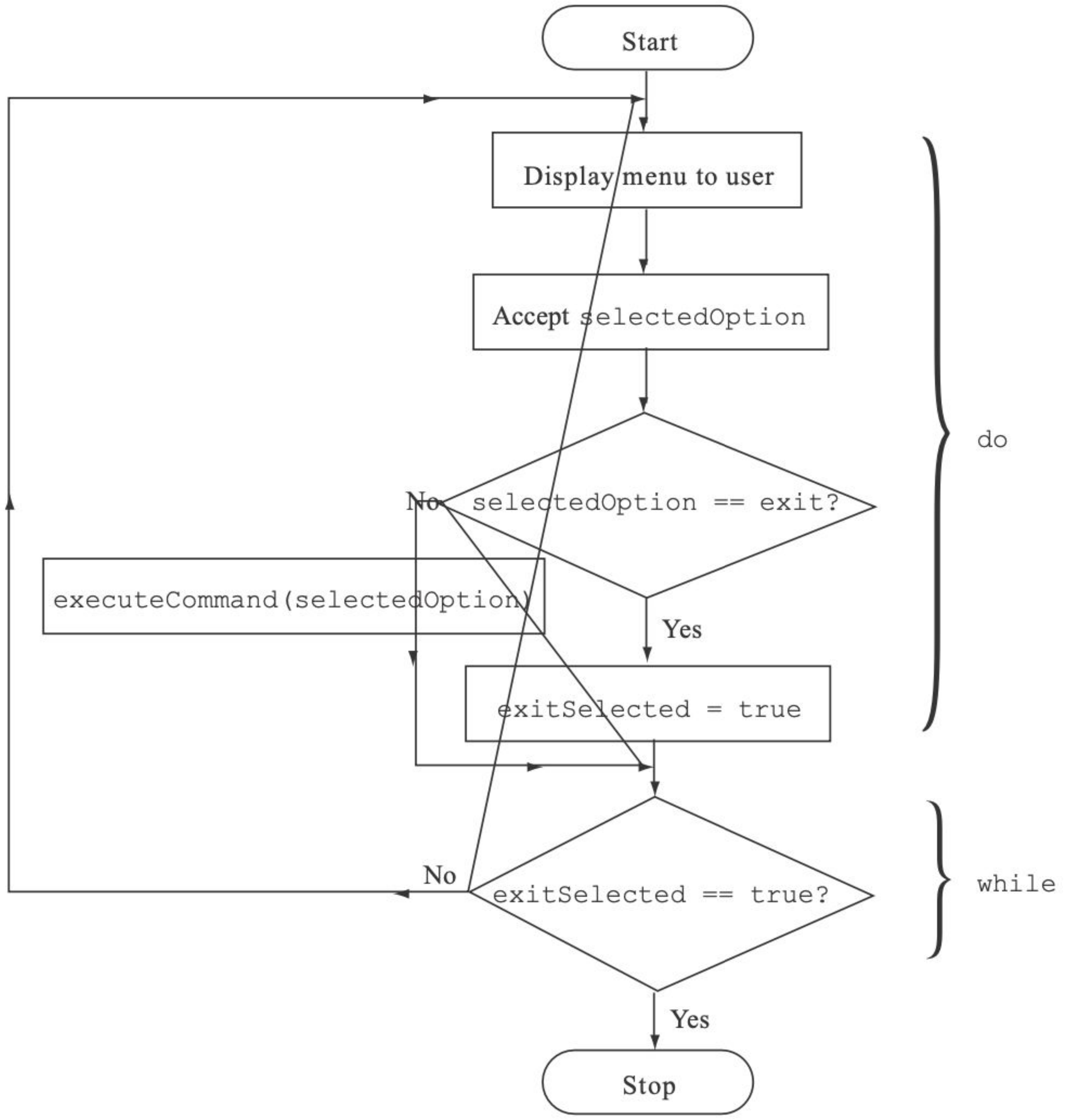
1. Be careful with while loops, as it's easy to create an infinite loop if the condition never becomes false.

2. In this example, the loop will never terminate unless you use a break statement or some other mechanism to exit the loop.

```
while (true) {
  // Infinite loop
}
```

do-while loop

1. The do-while loop in Java is similar to the while loop, but with one key difference:
2. the do-while loop guarantees that the block of code inside the loop is executed at least once, even if the condition is false initially.



do-while loop syntax

```
do {
    // Code to be executed
} while (condition);
```

1. The code block inside the do statement is executed first, and then the condition is evaluated.
2. If the condition is true, the loop continues, and the code block is executed again.
3. If the condition is false, the loop terminates, and the program continues with the next statement after the loop.

do-while loop example

1. In this example, the do-while loop will iterate five times, printing "Iteration: 0" to "Iteration: 4" to the console.

```
int i = 0;
do {
    System.out.println("Iteration: " + i);
    i++;
} while (i < 5);
```

Difference from While Loop

1. The main difference between the do-while loop and the while loop is that the do-while loop guarantees at least one execution of the code block inside the loop, whereas the while loop may not execute the code block at all if the condition is false initially.

do-while loop use cases

1. The do-while loop is useful when you want to execute a block of code at least once, and then continue executing it based on a condition.

2. For example, when you're processing user input, you might want to ask the user for input at least once and then continue based on their response.

Break and Continue

1. break and continue statements are used in loop constructs (for, while, do-while) to control the flow of the loop.

Break

1. The break statement is used to exit a loop prematurely, without completing the remaining iterations.
2. When a break statement is encountered inside a loop, the loop is terminated immediately, and the program continues with the next statement after the loop.
3. In this example, the loop will iterate from 0 to 2, and when i becomes 3, the break statement is executed, and the loop is terminated.

```
// Example of using 'break' to stop the loop when finding a specific value
int searchValue = 3;
boolean found = false;
for (int number : numbers) {
    if (number == searchValue) {
        found = true;
        break; // Exit the loop early when the value is found
    }
}
```

Continue

1. The continue statement is used to skip the current iteration of a loop and continue with the next iteration.
2. When a continue statement is encountered inside a loop, the remaining code in the current iteration is skipped, and the loop continues with the next iteration.
3. In this example, when number % 2 == 0, the continue statement is executed, and the println statement is skipped for that iteration.
4. for (int number : numbers) {

```
    if (number % 2 == 0) {
        continue; // Skip even numbers
    }
    System.out.println(number);
}
```

When to Use break and continue

1. Use break to exit a loop early if a certain condition is met and you no longer need to continue the loop.

2. Use continue to skip the current iteration and continue with the next iteration if a certain condition is met but you still want to continue looping.