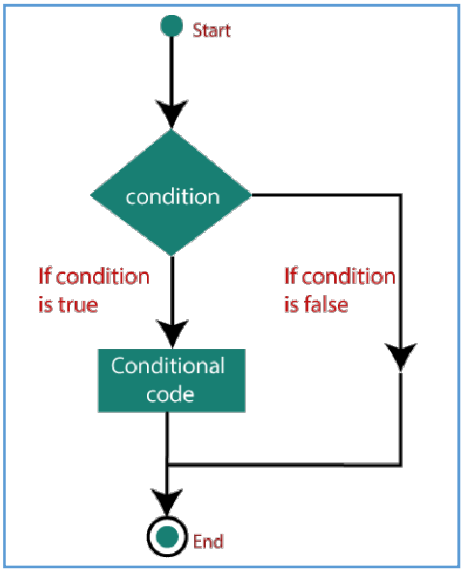
WEEK5
Decision making: if, if..else, switch Iterative: need of iterative statements; types of loops in java; how to use them; Break and continue statements;

Decision making

- Conditional statements are used to perform different actions based on various conditions.
- The conditional statement evaluates a condition before the execution of instructions.



Types of Conditional Statements

The conditional statements in Java are listed below:

- **if statement**
- **if….else statement**
- **if….else if….statement**
- **nested if statement**
- **switch statement**

The if statement

- It is one of the simplest decision-making statements which are used to decide whether a block of Java code will execute if a certain condition is true.

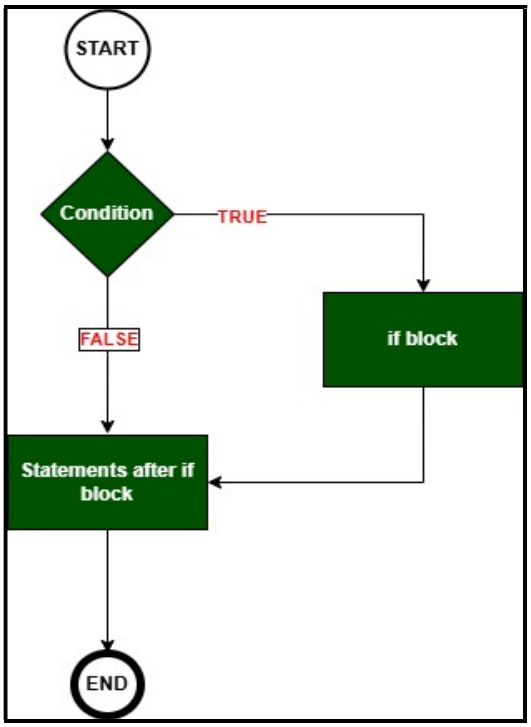
  if (condition)

  {

  // block of code will execute if the condition is true

  }

- If the condition evaluates to true, the code within **if statement** will execute, but if the condition evaluates to false, then the code after the end of **if statement (after the closing of curly braces)** will execute.



- Example:

```
int  x = 78;
if (x>70)
{
   System.out.println ("x is greater");
}
```

Output:

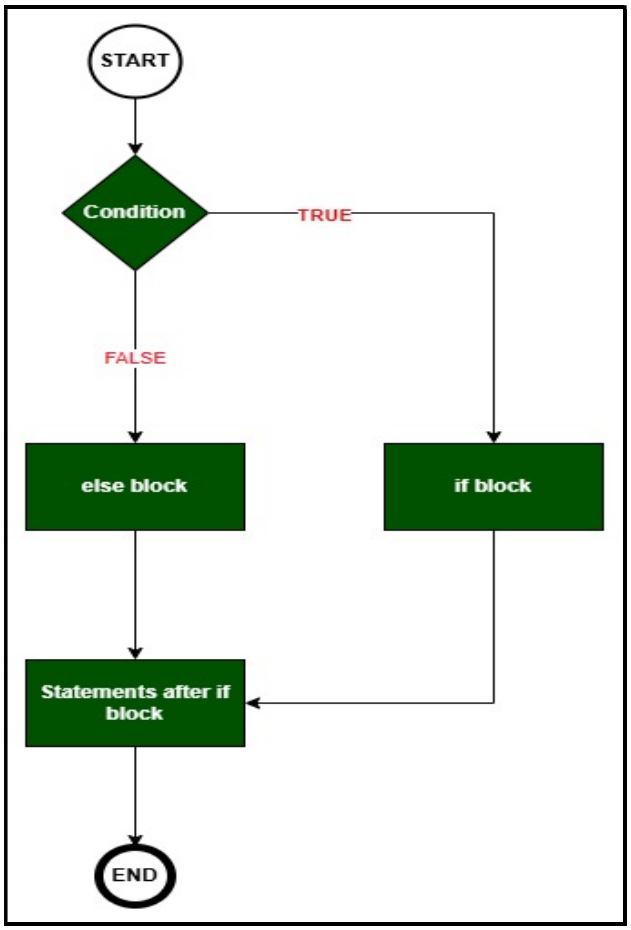
```
x is greater
```

The if….else statement

- An **if….else statement** includes two blocks that are **if block** and **else block.**
- If the condition is true, then the statements inside **if block** will be executed, but if the condition is false, then the statements of the **else block** will be executed.

```
if (condition)

{
        // block of code will execute if the condition is true
}
else
{
        // block of code will execute if the condition is false
}
```



- Example1:

```
Int x = 40, y=20;
if (x < y)
{
        System.out.println ("y is greater");
}
else
{
        System.out.println ("x is greater");
}
```

Output:

```
x is greater
```

- Example2:

```
int a = 10, b = 20, c = 30;
if ( a > b && a > c)
{
        System.out.println ("a is greater");
}
else if ( b > a && b > c )
{
        System.out.println ("b is greater");
}
else
{
        System.out.println ("c is greater");
}
```

Output:

```
c is greater
```

The nested if statement

- It is if statement inside another if statement.

```
if (condition1)
{
        Statement 1; //It will execute when condition1 is true
```

```
        if (condition2)
        {
                Statement 2; //It will execute when condition2 is true
        }
        else
        {
            Statement 3; //It will execute when condition2 is false
        }
}
```

- Example:

```
int num = 20;
if (num > 10)
{
        if (num%2==0)
                System.out.println ( num+ " is greater than 10 and even number");
        else
                System.out.println (num+ " is greater than 10 and odd number");
}
else
{
        System.out.println (num+" is smaller than 10");
}
System.out.println ("After nested if statement");
```

Output:

20 is greater than 10 and even number
After nested if statement

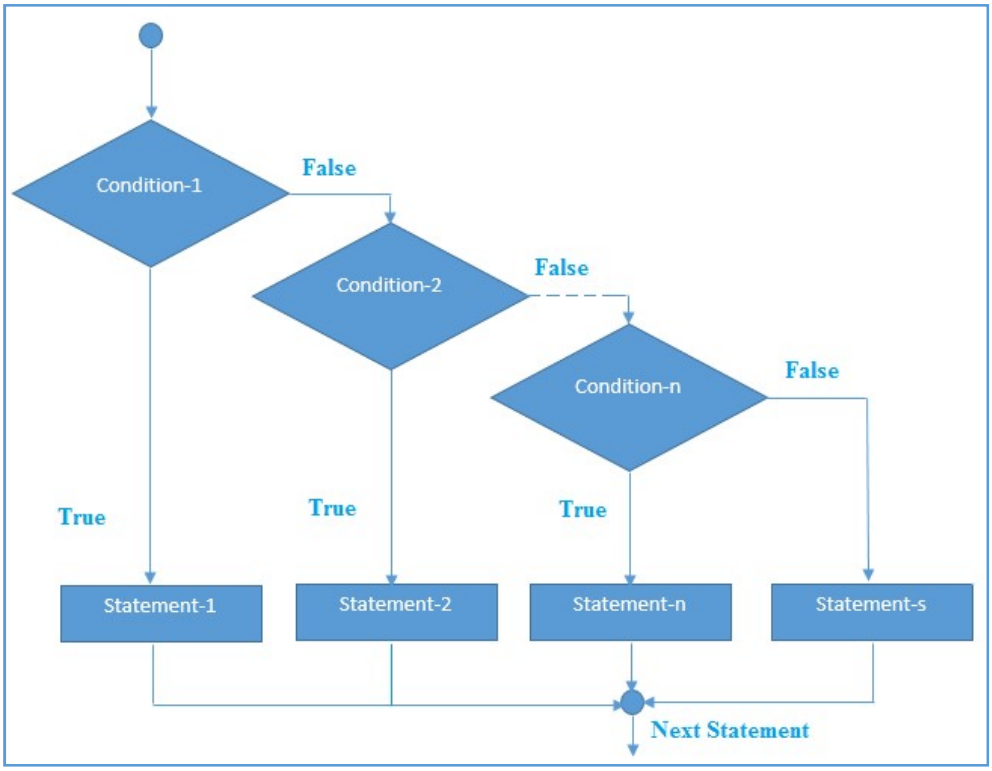
Java if-else-if ladder Statement

- The if-else-if ladder statement executes one condition from multiple statements.

```
        if(condition1)
        {
                //code to be executed if condition1 is true
        }
        else if(condition2)
        {
```

```
        //code to be executed if condition2 is true
}
else if(condition3)
{
        //code to be executed if condition3 is true
}
...
...
...
else

{
        //code to be executed if all the conditions are false
}
```



- Example1:

```
//It is a program of grading system for fail, Second class, First class & FCD.
public class GradingSystem
{
    public static void main(String[] args)
    {
            int marks=65;
            if(marks<35)
            {
                    System.out.println("fail");
            }
            else if(marks>=35 && marks<60)
            {
                    System.out.println("2nd Class");
            }
            else if(marks>=60 && marks<85)
            {
                    System.out.println("First Class");
            }
            else if(marks>=85 && marks<=100)
            {
                    System.out.println("FCD");
            }
            else
            {
                    System.out.println("Invalid marks!");
            }
    }
}
```

- Example2:

Program to check POSITIVE, NEGATIVE or ZERO

```
public class PositiveNegative
{
        public static void main(String[] args)
        {
                int number=-13;
                if(number>0)
                {
                        System.out.println("POSITIVE");
                }
                else if(number<0)
                {
                        System.out.println("NEGATIVE");
                }
                else
                {
                        System.out.println("ZERO");
                }
        }
}
```

The switch statement

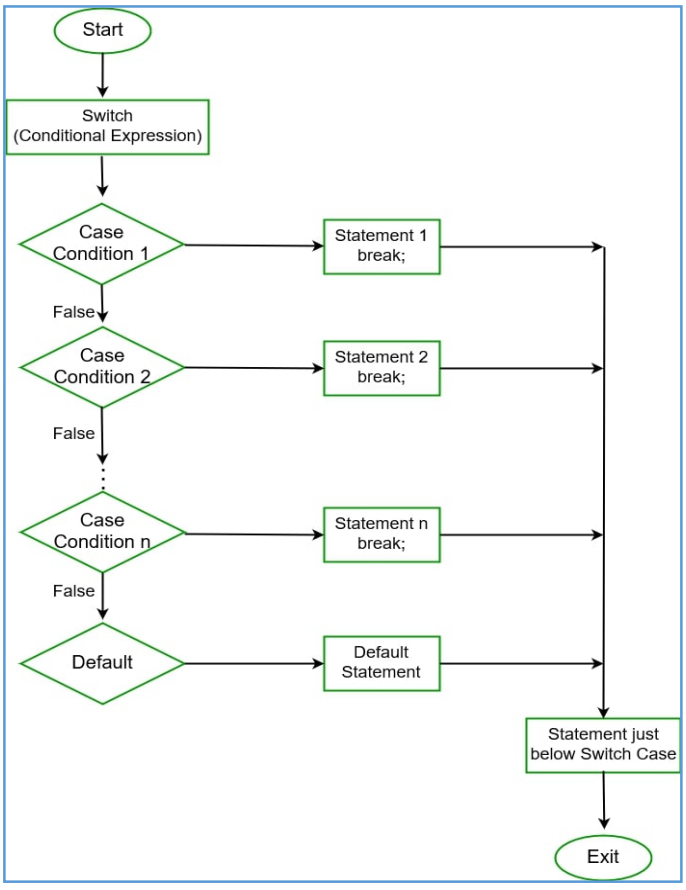
- The **switch statement** is a multi-way branch statement. In simple words, the Java switch statement executes one statement from multiple conditions. It is like an if-else-if ladder statement.
- Basically, the conditional expression can be a byte, short, char, and int primitive data types. It basically tests the equality of variables against multiple values.

Syntax:

```
switch (expression)
{
        case value1:
                // Code block for value1
        break; // Optional, prevents fall-through
```

```
        case value2:
            // Code block for value2
        break;
            // Add more cases as needed
        default:
            // Code block if no cases match
        break;
}
```

- **Expression:** The expression in the switch must evaluate to an int, char, byte, short, String, or an enum in Java.
- **Case Labels:** Each case label must be a constant or literal. Duplicate case values are not allowed.
- **Break Statement:** The break keyword prevents "fall-through" (execution of subsequent cases). If omitted, the execution continues to the next case.
- **Default Case:** The default block is optional and executed if no cases match. It can appear anywhere in the switch but is typically placed last.



Example 1: Imagine you're in a restaurant and you're choosing a drink. The waiter asks for your choice, and you respond with a number:

- If you say **1**, they bring you **Water**.
- If you say **2**, they bring you **Juice**.
- If you say **3**, they bring you **Soda**.
- If you say any other number, they say, "Sorry, we don't have that."

The **switch statement** in Java works exactly like this! It checks a value (your drink number) and matches it to a specific option (Water, Juice, Soda, etc.).

```
public class DrinkChooser
{
	public static void main(String[] args)
	{
		int choice = 2; // Your drink choice
		switch (choice)
		{
			case 1:
				System.out.println("You chose Water.");
				break;
			case 2:
				System.out.println("You chose Juice.");
				break;
			case 3:
				System.out.println("You chose Soda.");
				break;
			default:
				System.out.println("Sorry, we don't have that.");
				break;
		}
	}
}
```

Example 2: Display the day for the given day number.

```
public class SwitchCase
{
	public static void main(String[] args)
	{
		int day = 5;
		String dayString;

		// Switch statement with int data type
		switch (day)
		{
			case 1:
				dayString = "Monday";
				break;

			case 2:
				dayString = "Tuesday";
				break;

			case 3:
				dayString = "Wednesday";
				break;

			case 4:
				dayString = "Thursday";
				break;

			case 5:
				dayString = "Friday";
				break;

			case 6:
				dayString = "Saturday";
				break;

			case 7:
				dayString = "Sunday";
				break;

			default:
				dayString = "Invalid day";
		}
		System.out.println(dayString);
```

```
        }
}
```

Output:

```
Friday
```

- Example2: Cases without ‘break’ statements

Check the given day number is week day or weed end?

```
public class WithoutBreak
{
        public static void main(String[] args)
        {
                int day = 5;
                String dayType;

                // Switch statement with int data type
                switch (day)
                {
                        case 1:

                        case 2:

                        case 3:

                        case 4:

                        case 5:
                                dayType = "Weekday";
                                break;

                        case 6:

                        case 7:
                                dayType = "Weekend";
                                break;

                        default:
                                dayType = "Invalid day";
                }
                System.out.println(dayType);
        }
}
```

Output:

```
Weekday
```

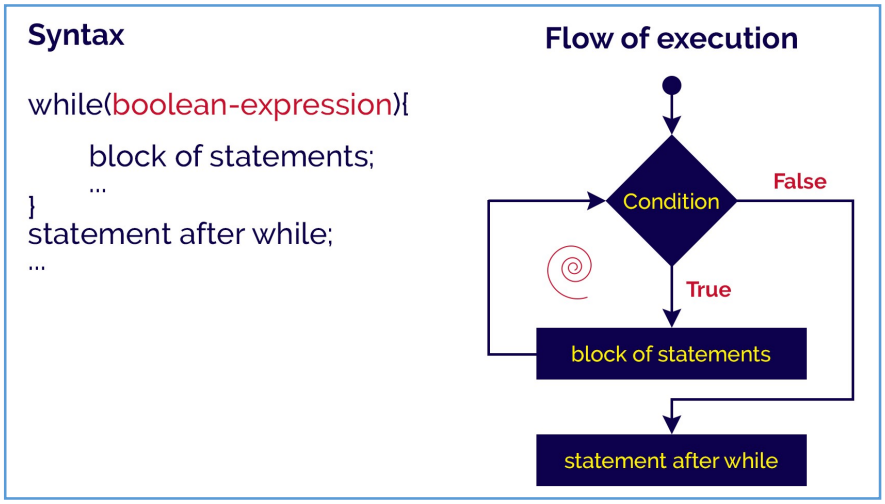
Java Iterative Statements

- The java programming language provides a set of iterative statements that are used to execute a statement or a block of statements repeatedly as long as the given condition is true.
- The iterative statements are also known as looping statements or repetitive statements. Java provides the following iterative statements.

  - while statement
  - do-while statement
  - for statement

'while' statement in java

- The while statement is used to execute a single statement or block of statements repeatedly as long as the given condition is TRUE.
- The while statement is also known as Entry control looping statement.
- The syntax and execution flow of while statement is as follows.



- Example:

```
public class WhilePgm
{
        public static void main(String[] args)
        {
                int num = 1;
                while(num <= 10)
                {
```

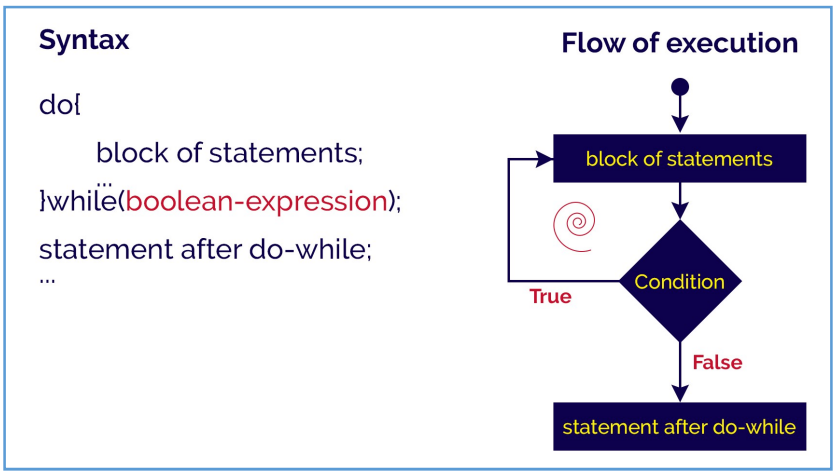
```
                    System.out.println(num);
                    num++;
                }
                System.out.println("Statement after while!");
            }
        }
```

Output:

```
1
2
3
4
5
6
7
8
9
10
Statement after while!
```

'do-while' statement in java

- The do-while statement is used to execute a single statement or block of statements repeatedly as long as given the condition is TRUE.
- The do-while statement is also known as the **Exit control looping statement**.
- The do-while statement has the following syntax.



- Example

```
public class DoWhilePgm
{
```

```
    public static void main(String[] args)
    {
            int num = 1;
            do
            {
                    System.out.println(num);
                    num++;
            } while(num <= 10);
            System.out.println("Statement after do-while!");
    }
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
Statement after while!
```

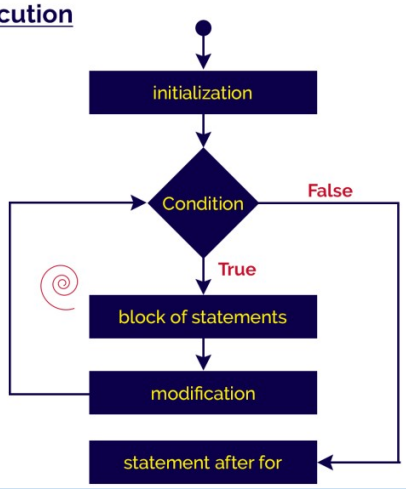
for statement in java

- The for statement is used to execute a single statement or a block of statements repeatedly as long as the given condition is TRUE.
- The for statement has the following syntax and execution flow diagram.

Syntax

```
for(initialization; boolean-expression; modofication){
        block of statements;
        ...
}
statement after for;
...
```

Flow of execution



- In for-statement, the execution begins with the initialization statement. After the initialization statement, it executes Condition. If the condition is evaluated to true, then the block of statements executed otherwise it terminates the for-statement. After the block of statements execution, the modification statement gets executed, followed by condition again.
- Example:

```
public class ForPgm
{
        public static void main(String[] args)
        {
                for(int i = 0; i < 10; i++)
                {
                        System.out.println("i = " + i);
                }

                System.out.println("Statement after for!");
        }
}
```

Output:

```
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
Statement after for!
```

Break and continue statements

The break and continue statements are the jump statements that are used to skip some statements inside the loop or terminate the loop immediately without checking the test expression. These statements can be used inside any loops such as for, while, do-while loop.

Break:

- The break statement in java is used to terminate from the loop immediately. When a break statement is encountered inside a loop, the loop iteration stops there, and control returns from the loop immediately to the first statement after the loop.
- Basically, break statements are used in situations when we are not sure about the actual number of iteration for the loop, or we want to terminate the loop based on some condition.
- **Syntax :**

```
break;
```

- Example:

```
// Java program to demonstrate using
// break to exit a loop
class BreakDemo
{
    public static void main(String[] args)
    {
        // Initially loop is set to run from 0-9
        for (int i = 0; i < 10; i++) {
            // Terminate the loop when i is 5
```

```
                    if (i == 5)
                        break;
                    System.out.println("i: " + i);
                }
                System.out.println("Out of Loop");
            }
        }
```

Output:

```
        i: 0
        i: 1
        i: 2
        i: 3
        i: 4
        Out of Loop
```

Continue:

- The continue statement in Java is used to skip the current iteration of a loop. We can use continue statement inside any types of loops such as for, while, and do-while loop.
- Basically continue statements are used in the situations when we want to continue the loop but do not want the remaining statement after the continue statement.
- **Syntax:**

```
continue;
```

- Example:

```
// Java program to demonstrates the continue
// statement to continue a loop
class ContinueDemo
{
    public static void main(String args[])
    {
        for (int i = 0; i < 10; i++) {
            // If the number is 2
```

```
                    // skip and continue
                    if (i == 2)
                        continue;

                    System.out.println(i);
                }
            }
        }
```

- Output:

```
0
1
3
4
5
6
7
8
9
```

Program No 01:

Write a java program to find the largest of 3 numbers.

```
class Largest
{
	public static void main(String args[])
	{
		int a = 10, b = 20, c = 30;
		if ( a > b && a > c)
		{
			System.out.println ("a is greater");
		}
		else if ( b > a && b > c )
		{
			System.out.println ("b is greater");
		}
		else
		{
			System.out.println ("c is greater");
		}
	}
}
```

Program No 02:

Write a java program to calculate the sum of odd and even numbers till 100

```
class Sum_Odd_Even
{
	public static void main(String[] args)
	{
		int  sumE = 0, sumO = 0;
		for(int i = 0; i <= 100; i++)
		{
			if(i % 2 == 0)
			{
				sumE = sumE + i;
			}
			else
			{
				sumO = sumO + i;
			}
		}
		System.out.println("Sum of Even Numbers:" + sumE);
		System.out.println("Sum of Odd Numbers:" + sumO);
	}
}
```

Output:

Program No 03:

Write a java program of grading system for fail, Second class, First class & FCD.

```
public class GradingSystem
{
    public static void main(String[] args)
    {
        int marks=65;
        if(marks<35)
        {
            System.out.println("fail");
        }
        else if(marks>=35 && marks<60)
        {
            System.out.println("2nd Class");
        }
        else if(marks>=60 && marks<85)
        {
            System.out.println("First Class");
        }
        else if(marks>=85 && marks<=100)
        {
            System.out.println("FCD");
        }
        else
        {
            System.out.println("Invalid marks!");
        }
    }
}
```

Output:

Program No 04:

Write a java program to display the day for the given day number.

```
public class Day
{
	public static void main(String[] args)
	{
		int day = 5;
		String dayString;

		switch (day)
		{
			case 1:
				dayString = "Monday";
				break;

			case 2:
				dayString = "Tuesday";
				break;

			case 3:
				dayString = "Wednesday";
				break;

			case 4:
				dayString = "Thursday";
				break;

			case 5:
				dayString = "Friday";
				break;

			case 6:
				dayString = "Saturday";
				break;

			case 7:
				dayString = "Sunday";
				break;

			default:
				dayString = "Invalid day";
		}
		System.out.println(dayString);
```

```
                }
        }
```

Output:

Program No 05:

Identify and resolve the issues in the following code.

```
class Issue
{
	public static void main(String args[])
	{
		for (int i = start; i <= start+1; i++)
		{
			System.out.println(i);
		}
	}
}
```