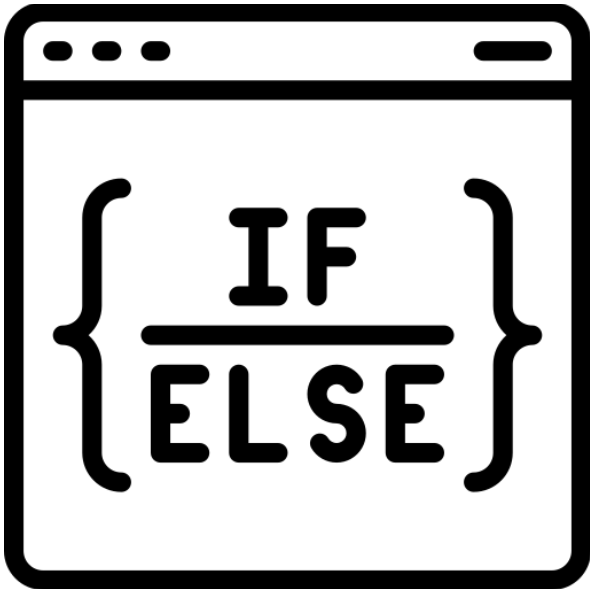
Unit 3 : Control Statements

Control statements are a fundamental part of programming that allow you to dictate the flow of execution in your code. They can be broken down into three main categories: Conditional Statements, Looping Statements, and Jump Statements.



1. Conditional Statements

Conditional statements enable a program to execute specific blocks of code based on whether a certain condition is met (evaluates to true or false). This introduces decision-making capabilities into your programs.

1 : `if` statement

2 : `if-else` statement

`3:if-else-if` ladder

4 : Nested `if` statement

5 : `switch` statement

- **1 : `if` Statement**
  - **Explanation**: This is the most basic decision-making statement. It executes a block of code only if the specified condition is true.
  - **Syntax**:
    Java

```
if (condition) {
    // Block of code to be executed if the condition is true
}
```

  - **Real-time Example**: A mobile banking app might use an `if` statement to check if a user's balance is sufficient before allowing a transaction to proceed.

Code :

```
public class MobileBankingApp {
 public static void main(String[] args) {
    double balance = 5000.00;   // User's current balance
    double transactionAmount = 3000.00; // Requested transfer
amount

    if (balance >= transactionAmount) {
        balance -= transactionAmount;
        System.out.println("Transaction successful!");
        System.out.println("Remaining Balance: ₹" + balance);
    } else {
        System.out.println("Transaction failed: Insufficient balance");
        System.out.println("Available Balance: ₹" + balance);
    }
  }
}
```

Output :

Transaction successful!

Remaining Balance: ₹2000.0

2 : `if-else` Statement

- **Explanation**: This statement provides a secondary path of execution for when the `if` condition is false.
- **Syntax**:
  Java

```
if (condition) {
    // Code to execute if condition is true
} else {
    // Code to execute if condition is false
}
```

- **Real-time Example**: A login system. **`if`** the password is correct, the user is logged in. **`else`**, it displays an "Invalid Credentials" message.

Code :

```
import java.util.Scanner;

public class LoginSystem {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      String username = "admin";
      String password = "1234";

      System.out.print("Enter username: ");
      String inputUser = sc.nextLine();
      System.out.print("Enter password: ");
      String inputPass = sc.nextLine();

      if(inputUser.equals(username) && inputPass.equals(password)) {
         System.out.println("Login Successful!");
      } else {
         System.out.println("Invalid Credentials");
      }
   }
}
```

Output :

Enter username : suhana

Enter password : parul@123

Login Successful!

3 : `if-else-if` Ladder

- **Explanation**: This structure is used to decide among several options. The `if` statements are evaluated from top to bottom. As soon as one of the conditions is true, the statement associated with it is executed, and the rest of the ladder is bypassed.
- **Syntax**:

Java

```
if (condition1) {
    // Code to execute if condition1 is true
} else if (condition2) {
    // Code to execute if condition1 is false and condition2 is true
} else {
    // Code to execute if all conditions are false
}
```

- **Real-time Example**: A shipping-cost calculator. **if** the package weight is under 1 kg, the cost is $5. **else if** the weight is between 1 kg and 5 kg, the cost is $10. **else**, the cost is $20.

Code :

```
import java.util.Scanner;

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

        System.out.print("Enter package weight (in
kg): ");
        double weight = sc.nextDouble();

        double cost;

        if (weight < 1) {
            cost = 5.0;
        } else if (weight >= 1 && weight <= 5) {
            cost = 10.0;
        } else {
            cost = 20.0;
        }

        System.out.println("Shipping cost: $" +
cost);
    }
}
```

Output :

```
Enter package weight (in kg): 0.8
Shipping cost: $5.0
```

4 : Nested if-else in Java: Explanation

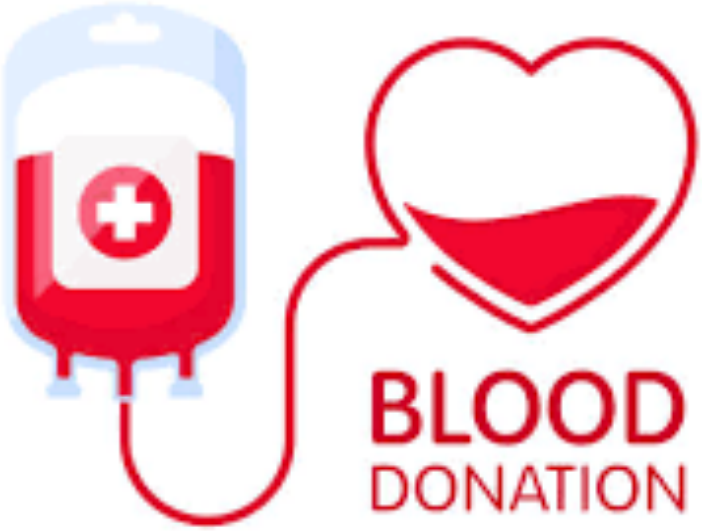
🔷 **Definition:**

A **nested if-else** is an `if` or `else if` statement inside another `if` or `else` block. It allows checking **multiple conditions** in a **hierarchical or dependent manner**.

🔶 **Syntax:**

```
if (condition1) {
    // Executes when condition1 is true
    if (condition2) {
        // Executes when condition2 is also true
    } else {
        // Executes when condition2 is false
    }
} else {
    // Executes when condition1 is false
}
```

Real time example : Eligibility for blood donation .



Code :

```
public class NestedIfElseExample {
    public static void main(String[] args) {
        int age = 20;
        int weight = 55;

        if (age >= 18) {
            if (weight > 50) {
                System.out.println("Eligible to donate blood");
            } else {
                System.out.println("Not eligible due to weight");
            }
        } else {
            System.out.println("Not eligible due to age");
        }
    }
}
```

Output :

Eligible to donate blood

5 : `switch` Statement

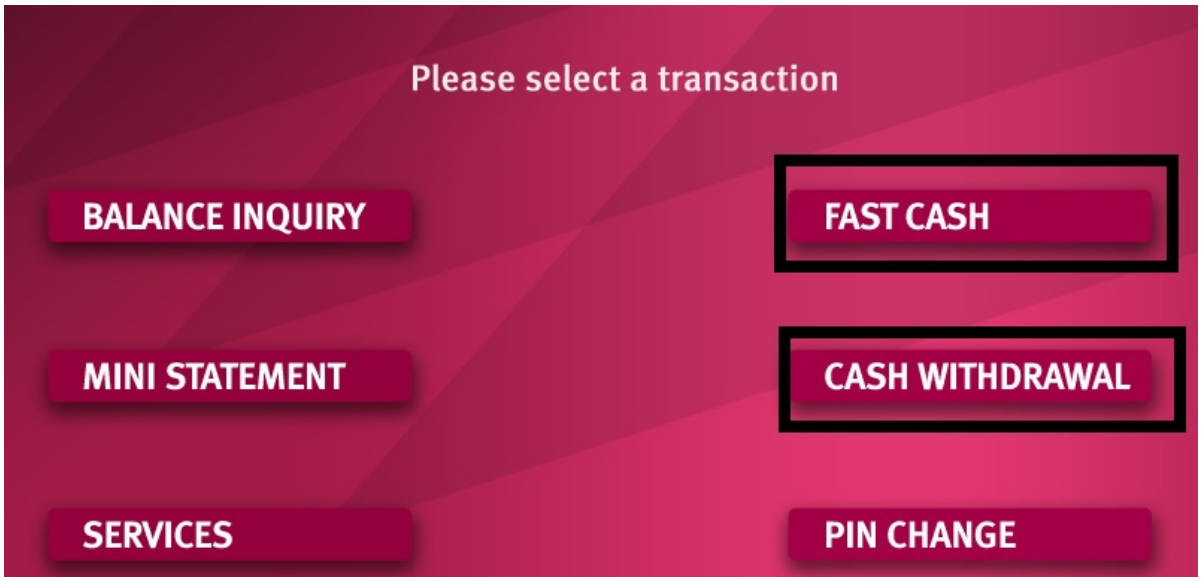
- **Explanation**: The `switch` statement allows a variable to be tested for equality against a list of values. It provides a cleaner alternative to long `if-else-if` ladders.

- **Syntax**:
  Java

```
switch (expression) {
```

```
    case value1:
        // Code to be executed
        break; // Exits the switch block
    case value2:
        // Code to be executed
        break;
    default: // Optional
        // Code to be executed if expression doesn't match any case
}
```

- **Real-time Example**: An ATM machine. The `switch` statement takes the user's menu choice (e.g., "Withdraw," "Deposit," "Check Balance"). Based on the choice, it executes the corresponding function.



Code :

```
import java.util.Scanner;

public class ATM {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      double balance = 1000.0; // initial balance

      System.out.println("===== ATM Menu =====");
      System.out.println("1. Withdraw");
      System.out.println("2. Deposit");
      System.out.println("3. Check Balance");
      System.out.print("Enter your choice (1-3): “);
      int choice = sc.nextInt();

      switch (choice) {
         case 1:
            System.out.print("Enter amount to withdraw: ");
            double withdraw = sc.nextDouble();
            if (withdraw <= balance) {
               balance -= withdraw;
               System.out.println("Withdrawal successful!");
               System.out.println("Remaining Balance: $" + balance);
            } else {
               System.out.println("Insufficient balance!");
            }
            break;

         case 2:
            System.out.print("Enter amount to deposit: ");
            double deposit = sc.nextDouble();
            balance += deposit;
            System.out.println("Deposit successful!");
            System.out.println("Updated Balance: $" + balance);
            break;

         case 3:
            System.out.println("Your current balance is: $" + balance);
            break;

         default:
            System.out.println("Invalid choice. Please select 1-3.");
      }
   }
}
```

Output :

===== ATM Menu =====

1. Withdraw

2. Deposit

3. Check Balance

Enter your choice (1-3): 1

Enter amount to withdraw: 300

Withdrawal successful!

Remaining Balance: $700.0

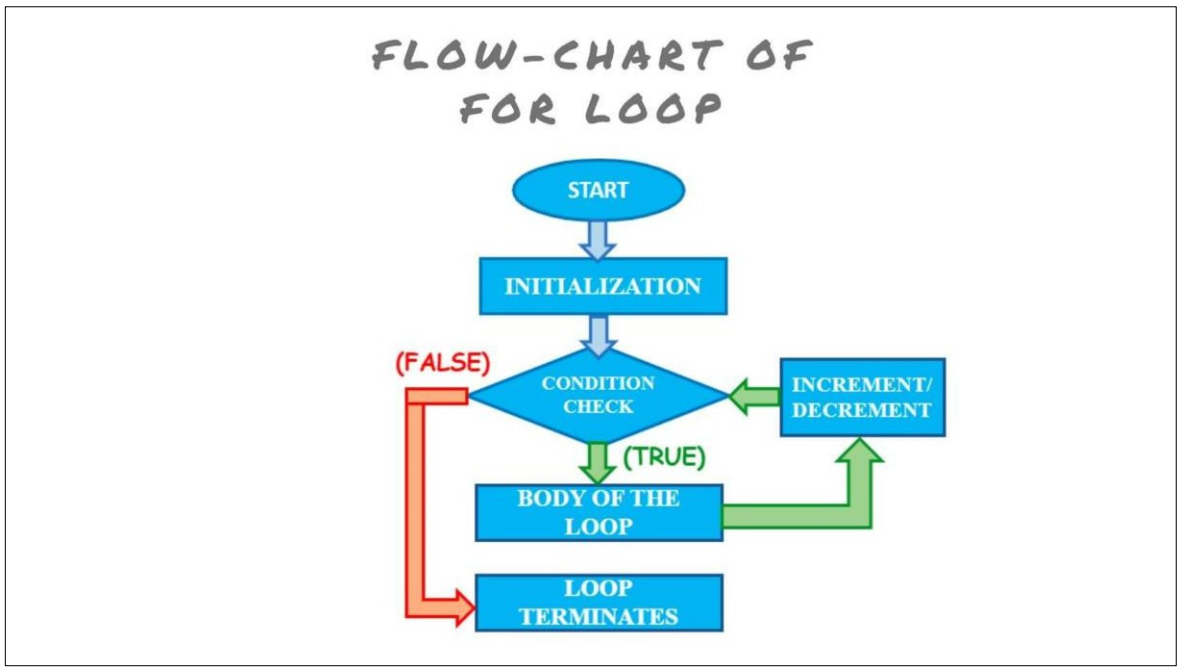
Loops


```
class Demo
{
  public static void main(String [] args)
  {
    System.out.println("JAVA");
    System.out.println("JAVA");
    System.out.println("JAVA");
    System.out.println("JAVA");
    System.out.println("JAVA");
  }
}
```

Imagine you have to print "JAVA" 100 times?

Are you going to type the statement 100 times?

Here comes the loop to rescue the programmer and make the job easy. Most fundamental and basic loop is known as FOR loop.



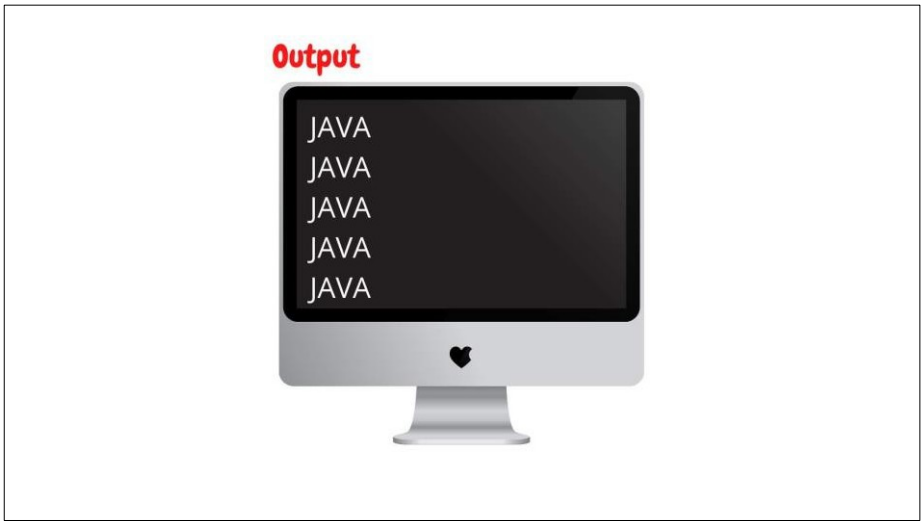
1 : For Loop

Syntax:

for (initialization; condition; increment/ decrement)

Code:

```
class Demo
{
   public static void main (String [] args)
   {
         int i;
         for ( i=1; i<=5; i++)
         {
            System.out.println("JAVA");
         }
    }
}
```



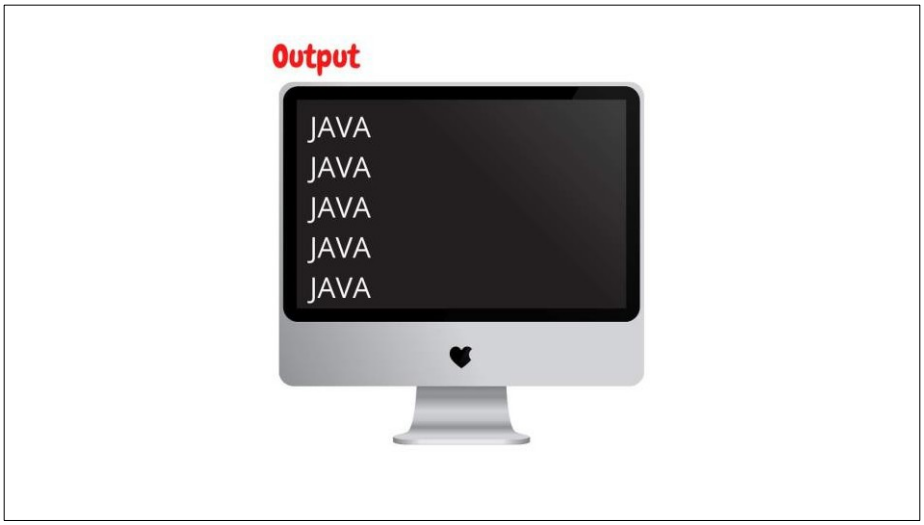
2 : `while` Loop

🔷 **Syntax:**

```
initialization;
while (condition) {
    // statements
    increment/decrement;
}
```

🔷 **Code:**

```
class Demo {
    public static void main(String[] args) {
        int i = 1;
        while (i <= 5) {
            System.out.println("JAVA");
            i++;
        }
    }
}
```



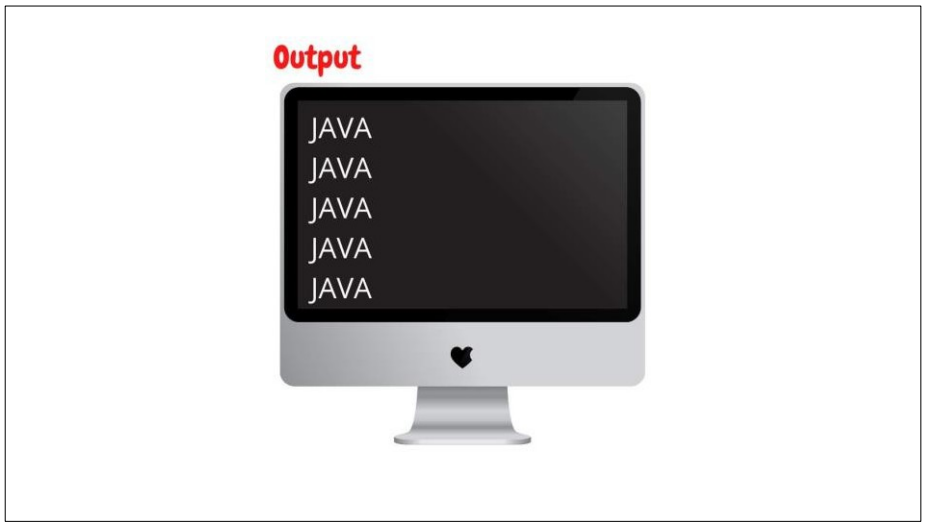
3 : do-while Loop

🔷 **Syntax:**

```
initialization;
do {
    // statements
    increment/decrement;
} while (condition);
```

🔷 **Code:**

```
class Demo {
    public static void main(String[] args) {
        int i = 1;
        do {
            System.out.println("JAVA");
            i++;
        } while (i <= 5);
    }
}
```



Jump Statements

Jump statements unconditionally transfer program control to another part of the program.

- **`1 : break`**
  - **Explanation**: This statement causes an immediate exit from a `switch` statement or a loop (`for`, `while`, or `do-while`).

  - **Code :**
  - public class BreakExample {
  - public static void main(String[] args) {
  - for (int i = 1; i <= 5; i++) {
  - if (i == 3) {
  - break; // exits the loop when i is 3
  - }
  - System.out.println("i = " + i);
  - }
  - }
  - }

- **`2 : continue`**
    - **Explanation**: This statement skips the current iteration of a loop and forces the next iteration to begin.
    - **Real-time Example**: Processing a list of financial transactions. If a transaction record is found to be corrupt, the system can use **`continue`** to skip that specific record and process the next one.

Code :

```
public class ContinueExample {
  public static void main(String[] args) {
    for (int i = 1; i <= 5; i++) {
      if (i == 3) {
        continue; // skips the rest of loop when i is 3
      }
      System.out.println("i = " + i);
    }
  }
}
```

- **`3 : return`**
    - **Explanation**: The `return` statement is used to explicitly exit from a method. It can also be used to pass a value back to the code that called the method.
    - **Real-time Example**: A function that validates a user's password. If the function finds the password is too short, it can immediately **`return false`** without checking other requirements.

Code :

```
public class ReturnExample {
    public static void main(String[] args) {
        for (int i = 1; i <= 5; i++) {
            if (i == 3) {
                return; // exits the method entirely when i is 3
            }
            System.out.println("i = " + i);
        }
    }
}
```