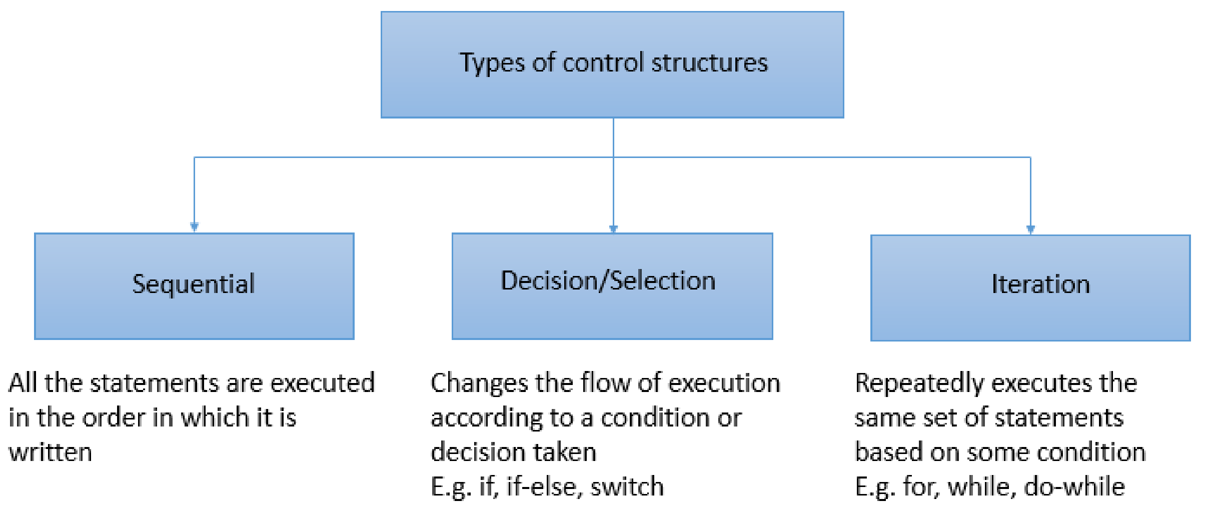
Control Structures

The instructions are usually executed line by line. Sometimes, all the statements in a program may not be executed. There can be change in the flow of control and can be implemented using control structures.

Remember that you have already learnt about control structures while discussing algorithms.


Types of control structures
Sequential
Decision/Selection
Iteration
All the statements are executed in the order in which it is written
Changes the flow of execution according to a condition or decision taken
E.g. if, if-else, switch
Repeatedly executes the same set of statements based on some condition
E.g. for, while, do-while

If Condition:

Selection control structure is implemented using if statement.

An "if" statement contains a relational and logical expression followed by a block of statements. Based on the result of the expression, the corresponding statements/code blocks get executed or skipped.

Syntax:

```
if (<condition>) {  // Curly braces are not required if there is only one statement inside the block

        <statements>;

}
```

The statements inside the "if" block gets executed only when the condition evaluates to true.

Let's have a look at a scenario.

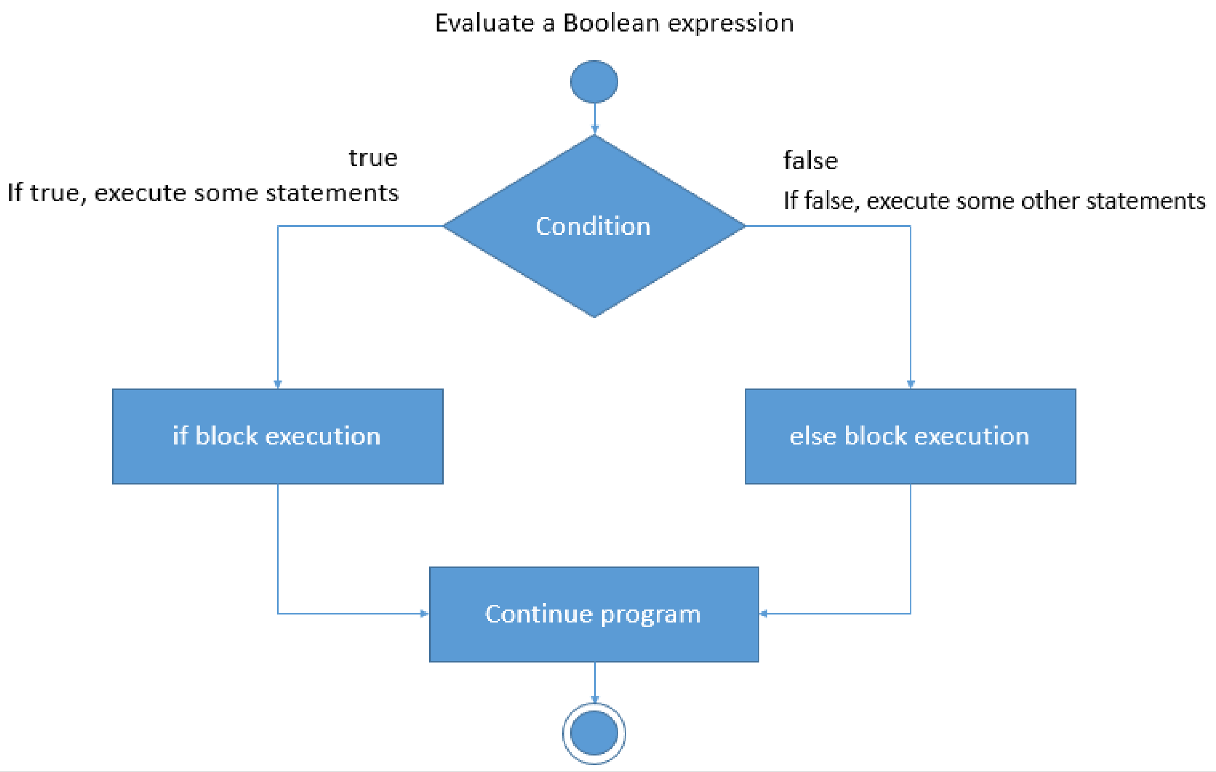
While ordering food in "SwiftFood", when a customer orders food items, you need to calculate the total cost that the customer should pay for the order. The "Regular Customers" are provided with a 5% discount for their orders. Here, we are assuming that each food item costs $10. In the condition, we are checking if the customer type is "Regular". If the condition is true, the statements within the if block get executed to calculate total cost after discount.

```
public class Customer {

  public static void main(String[] args) {

        String customerType = "Regular";

        int quantity = 2;

        int unitPrice = 10;

        int totalCost = 0;

        int discount = 5;

        totalCost = unitPrice * quantity;

        if (customerType == "Regular") {

              totalCost = totalCost - (totalCost * discount / 100);

              System.out.println("You are a regular customer and eligible for 5% discount");

        }

        System.out.println("Total cost: " + totalCost);

  }

}
```

An **if** statement can be written along with an **else** statement. The condition/expression given in the if statement is checked and set of statements are executed based on the outcome of the condition. If the condition is true, the statements written in if block get executed. If the condition is false, then the statements inside else block get executed.

```
if (<condition>) {
        <statements>;
}
else {
        <statements>;
}
```

Flowchart:


Evaluate a Boolean expression
true
If true, execute some statements
Condition
false
If false, execute some other statements
if block execution
else block execution
Continue program

In the below code, we are calculating the total cost that the customers should pay for the order. In the previous example, we considered only Regular customers. The non-regular customers have to pay an additional delivery charge of $5. So, first the customer type is checked. If the customer type is Regular, then the if block gets executed. For other types of customers, the statements of else block get executed.

```
public class Customer {
   public static void main(String[] args) {
        String customerType = "Regular";
        int quantity = 2;
        int unitPrice = 10;
        int totalCost = 0;
        int discount = 5;
        int deliveryCharge = 5;
        totalCost = unitPrice * quantity;
        if (customerType == "Regular") {
            totalCost = totalCost - (totalCost * discount / 100);
            System.out.println("You are a regular customer and eligible for 5% discount");
        } else {
            totalCost = totalCost + deliveryCharge;
            System.out.println("You need to pay an additional delivery charge of $5");
        }
        System.out.println("Total cost: " + totalCost);
    }
}


//Java program to check whether the given number is even or odd
//Observe the output for different values of number variable
class Tester {
    public static void main(String[] args) {
        int number = 5;
        if (number % 2 == 0) {
            // This block will get executed if the if-condition is true

            System.out.println(number + " is an even number");

        } else {

            // This block will get executed if the if-condition is false

            System.out.println(number + " is an odd number");

        }

    }

}
```

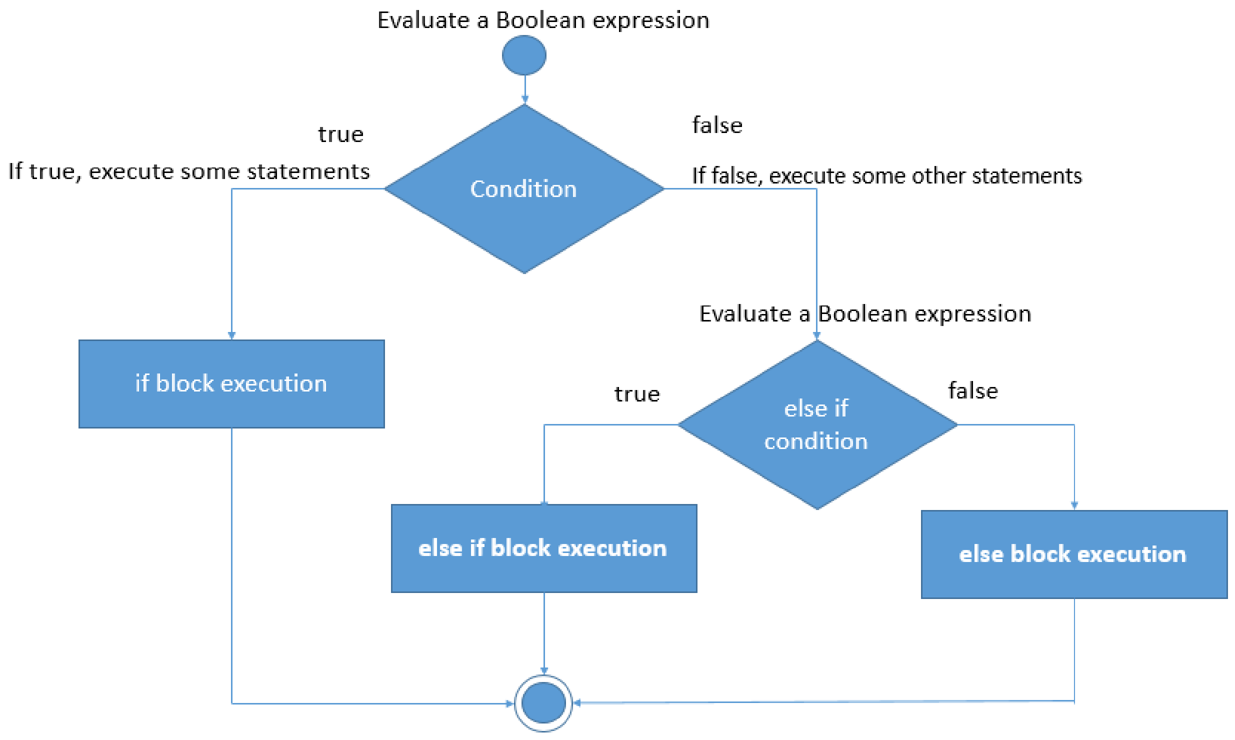
You can also have **else if** statements. As the name suggests, it is a combination of else and if. Like else, it extends an if statement to execute a different set of statements in case the original if expression evaluates to false. Then, the conditions present inside the else if blocks are checked. Once a condition evaluates to true, remaining else if and else statements are skipped.

When all the conditions are false, the else block is executed. Coding the else block is optional.

Syntax:

```
if (<condition 1>) {

        <statements>;

}

else if (<condition 2>) {

        <statements>;

}

else if (<condition 3>) {

        <statements>;

}

else {
        <statements>;
}
```

Flow Chart:


Evaluate a Boolean expression
true
If true, execute some statements
Condition
false
If false, execute some other statements
if block execution
Evaluate a Boolean expression
true
else if condition
false
else if block execution
else block execution

When a customer places an order, the total cost varies according to the customer type. The customer type must be either Regular or Guest. For other values, the program should display an error message. This can be represented using if-else-if statements as shown below.

```
public class Customer {
  public static void main(String[] args) {
        String customerType = "Regular";
        int quantity = 3;
        int unitPrice = 10;
        int discount = 5;
        int deliveryCharge = 5;

        int totalCost = unitPrice * quantity;

        if (customerType == "Regular") {

                totalCost = totalCost - (totalCost * discount / 100);

                System.out.println("You are a regular customer and eligible for 5% discount");

                System.out.println("The total cost to be paid is " + totalCost);

        } else if (customerType == "Guest") {

                totalCost = totalCost + deliveryCharge;

                System.out.println("You need to pay an additional delivery charge of $5");

                System.out.println("The total cost to be paid is" + totalCost);

        } else // If there is only one statement inside a block, {} is optional

                System.out.println("Invalid customer type");

    }

}
```

The code given below displays the grade of a student based on the marks using if - else if blocks. Execute the code with different values of marks and observe the output.

```
//Observe the output for different values of marks

class Tester {

    public static void main(String[] args) {

        int marks = 90;


        if (marks < 50) {
                System.out.println("Fail");

        } else if (marks >= 50 && marks < 60) {

                System.out.println("D grade");

        } else if (marks >= 60 && marks < 70) {

                System.out.println("C grade");

        } else if (marks >= 70 && marks < 80) {

                System.out.println("B grade");

        } else if (marks >= 80 && marks < 90) {

                System.out.println("A grade");

        } else if (marks >= 90 && marks <= 100) {

                System.out.println("A+ grade");

        } else {

                System.out.println("Invalid!");

        }

   }

}
```

When an if statement is written within another if statement, it is known as nested if statement. It enables us to test multiple criteria. The inner if block condition gets executed only when the condition of the outer if block evaluates to true.

Syntax:

```
if (<condition 1>) {

        if (<condition 2>) {

        <statements>;

        }
```

```
          else {

          <statements>;

          }

}
```

In the below code, the total cost for an order is calculated using nested if blocks. The assumption over here is that each food item costs $10. The Regular customers are provided with a 5% discount for their orders whereas the Guests need to pay an additional delivery charge of $5. First, the customer type is checked. If the customer type is Regular, then the if block gets executed. If the customer type is Guests, then else if block gets executed. Also, for regular customers, if the total cost exceeds $20, a special gift voucher will be provided to the customers. This condition is checked within the outer if block. If customer type is invalid, then the statements of else block get executed.

```
public class Customer {

   public static void main(String[] args) {

          String customerType = "Guest";

          int quantity = 2;

          int unitPrice = 10;

          int totalCost = 0;

          int discount = 5;

          int deliveryCharge = 5;

          totalCost = (unitPrice * quantity);

          if (customerType == "Regular") {

                  totalCost = totalCost - (totalCost * discount / 100);

                  System.out.println("You are a regular customer and have got 5% discount");

                  System.out.println("The total cost to be paid is " + totalCost);

                  if (totalCost >= 20) {
                    System.out.println("You have got a gift voucher!!!!");
                }
            } else if (customerType == "Guest") {
                totalCost = totalCost + deliveryCharge;
                System.out.println("You need to pay an additional delivery charge of $5");
                System.out.println("The total cost to be paid is " + totalCost);
            } else {
                System.out.println("Invalid selection");
            }
        }
    }



class Customer {
  public static void main(String[] args) {
        String customerType = "Guest";
        int quantity = 2;
        int unitPrice = 10;
        int totalCost = 0;
        int discount = 5;
        int deliveryCharge = 5;
        totalCost = (unitPrice * quantity);
        if (customerType == "Regular") {
            totalCost = totalCost - (totalCost * discount / 100);
```

```
                System.out.println("You are a regular customer and have got 5% discount");

                System.out.println("The total cost to be paid is " + totalCost);

                if (totalCost >= 20) {

                        System.out.println("You have got a gift voucher!!!!");

                }

        } else if (customerType == "Guest") {

                totalCost = totalCost + deliveryCharge;

                System.out.println("You need to pay an additional delivery charge of $5");

                System.out.println("The total cost to be paid is " + totalCost);

        } else {

                System.out.println("Invalid selection");

        }

   }

}
```

Similar to if else statements, **switch** statement is also a selection control structure.

The switch statement enables to select a block from a set of options. It allows the flow of execution to be switched according to a value.

Syntax:

```
switch (expression or variable) {

  case value1: <statements>;

               break;

  case value2: <statements>;

               break;
  default: <statements>;
}
```

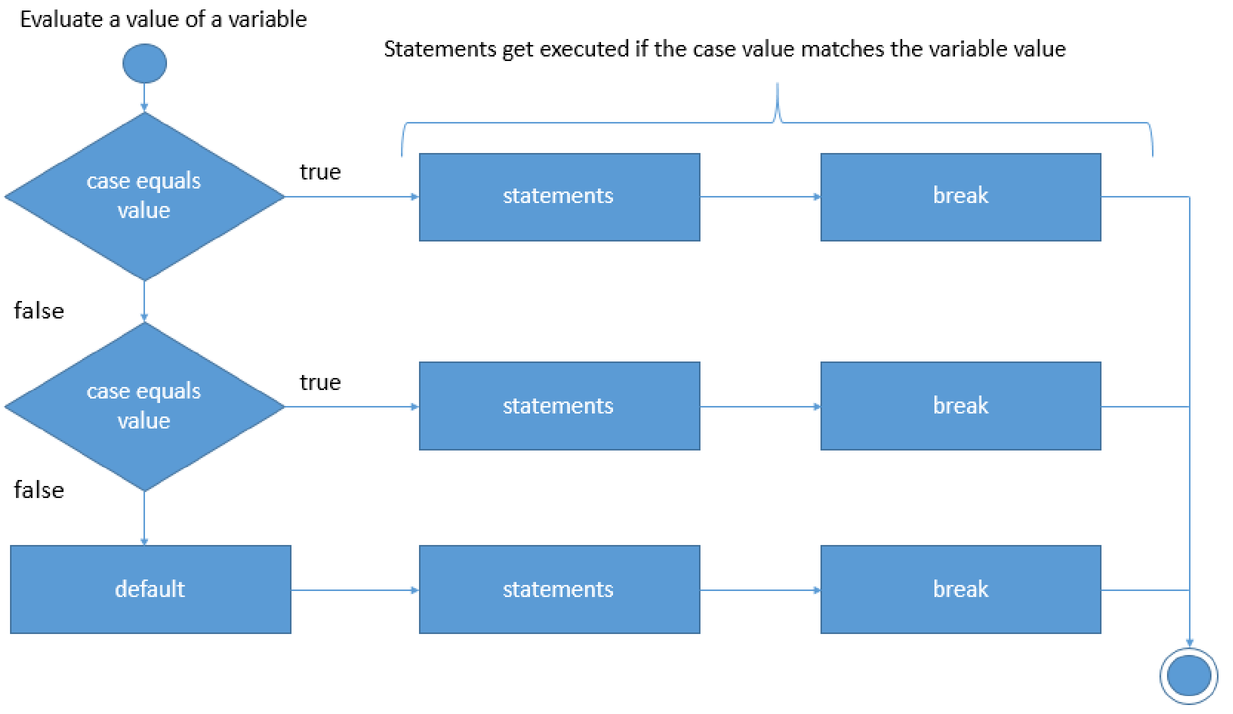
During execution, the result of expression or variable written in the switch statement is compared with the constant values of cases one by one. When a match is found, the set of statements present in that case are executed until a **break** statement is encountered or till the end of switch block, whichever occurs first. In the absence of break statement, the flow of control falls through subsequent cases and executes the statements of all those cases until it reaches a break statement or end of switch block.

The switch block can have a special case called **default**. The default case is executed when none of the cases match with the value of expression/variable. default is optional. If none of the cases match and if there is no default statement, the control comes out of switch block without executing any case.

In Java, the switch block works only for the following data types:

- char and integral datatypes are supported in switch-case statement, but float or double are not supported.
- String datatype is also supported in switch-case statement from Java 7 version onwards.

Flowchart:



In the below example, the food item ordered gets displayed. If the ordered food is anything other than Burger, Pizza and Sandwich, then a message, 'Invalid selection' is displayed from default case.

```
public class Customer {

   public static void main(String[] args) {

          String orderedFood = "Pizza";

          switch (orderedFood) {

          case "Burger":

                 System.out.println("You have ordered Burger. Unit price: $10");

                 break;

          case "Pizza":

                 System.out.println("You have ordered Pizza. Unit price: $15");

                 break;

          case "Sandwich":
                System.out.println("You have ordered Sandwich. Unit price: $8");

                break;

        default:

                System.out.println("Invalid selection");

        }

    }

}
```

Execute the code given below and observe the output. Modify the code to get 5% discount for Regular customer and 10% for Premium customer. Also, try changing sequence of the cases, observe how it works if default is written before other cases.

Iteration control structures are used to execute a set of statements repeatedly. To terminate the repetition, a condition is required.

Let us consider that the girl in the image should jump over the rope repetitively until she is tired. That means, a repeated action has to be performed as long as a condition (getting tired) is met. Also, the number of times the girl is going to jump over the rope cannot be estimated before.

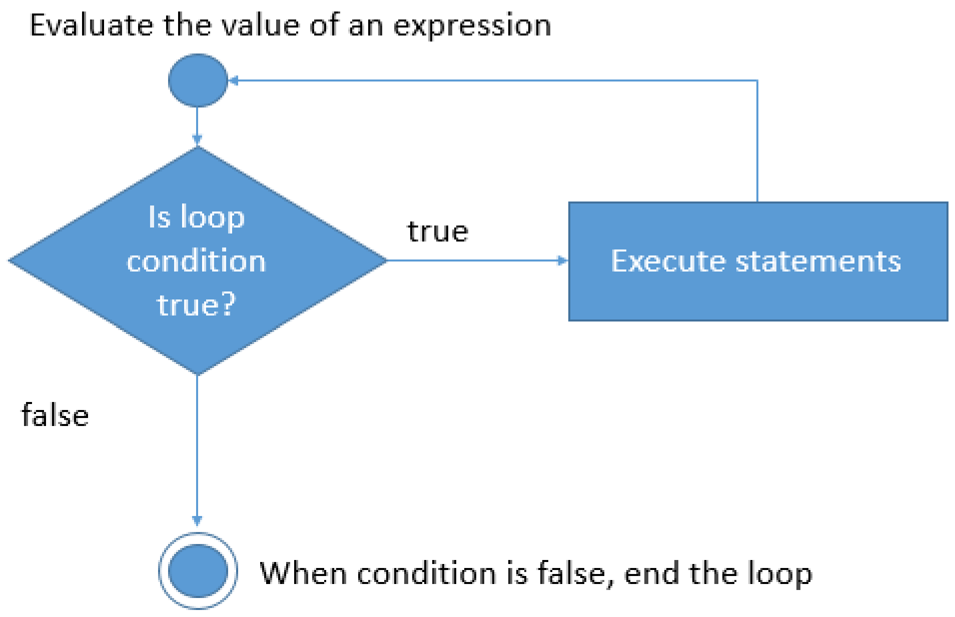
Consider another scenario. Here, the girl should keep juggling until she misses a catch. It is not known at the beginning of juggling when she will miss a catch.

Similarly, in programming, when you want to repeatedly execute the statements as long as a condition is met, you can use the iteration control structure called **while** loop. When the condition becomes false, the while loop terminates and control goes to the statement written after the while loop. The while loop is used when the number of iterations are not known. In case of while loop, the condition is tested before entering the while loop block and hence it is known as an **entry-controlled loop**.

Syntax:

```
while (<condition>) {
  <statements>;
}
```

Flowchart


Evaluate the value of an expression
Is loop condition true?
true
Execute statements
false
When condition is false, end the loop

As discussed before in pseudo-code, let us assume that an order should be placed for multiple food items and that the number of food items to be ordered is not known at the beginning. If the user wants to order an additional food item, the process of accepting the food item, quantity and calculation of the total amount is repeated whenever the order is placed.

In the below code, wantToAddFoodItem is initialized to 'Y'. The condition of while loop evaluates to true, so, the statements inside the loop get executed, i.e., the total cost of the order is calculated. The customer is then asked if he/she wants to add one more food item. If the customer provides the input as 'Y', then the condition evaluates to true and the statements inside the loop are executed again. If the input is 'N', the flow of control goes out of the loop.

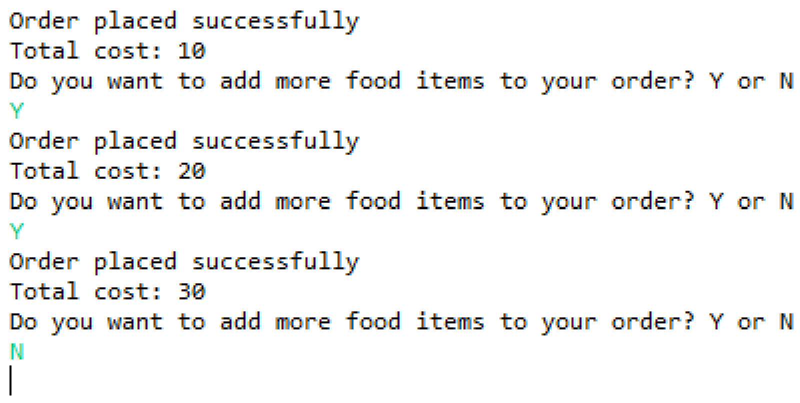
Execute the below code in Eclipse and observe the output by providing the input. We have imported Scanner class to take the input from the customer. You will learn about import statement later in the course.

Please note that code containing Scanner class has to be executed in Eclipse only.

```
//Importing the Scanner class
import java.util.Scanner;
public class Customer {
   public static void main(String[] args) {
         // Create a Scanner object
         Scanner sc = new Scanner(System.in);
```

```
        int totalCost = 0;
        char wantToAddFoodItem = 'Y';
        int unitPrice = 10;
        int quantity = 1;
        while (wantToAddFoodItem == 'Y') {
            totalCost = totalCost + (quantity * unitPrice);
            System.out.println("Order placed successfully");
            System.out.println("Total cost: " + totalCost);
            System.out.println("Do you want to add more food items to your order? Y or N");
            String input = sc.next(); // Accepting input from the customer
        // Extracting first character from the input string
            wantToAddFoodItem = input.charAt(0);
        }
    }
}
```

O/P



```
Order placed successfully
Total cost: 10
Do you want to add more food items to your order? Y or N
Y
Order placed successfully
Total cost: 20
Do you want to add more food items to your order? Y or N
Y
Order placed successfully
Total cost: 30
Do you want to add more food items to your order? Y or N
N
```

The next loop that you will see is **do-while**.

When the loop has to be executed at least once before the condition is checked, do-while loop is used. After the first execution, the loop then gets repeated as long as the condition is true. In case of do-while loop, the condition is tested after executing the code block. Hence, it is called an **exit-controlled loop**.

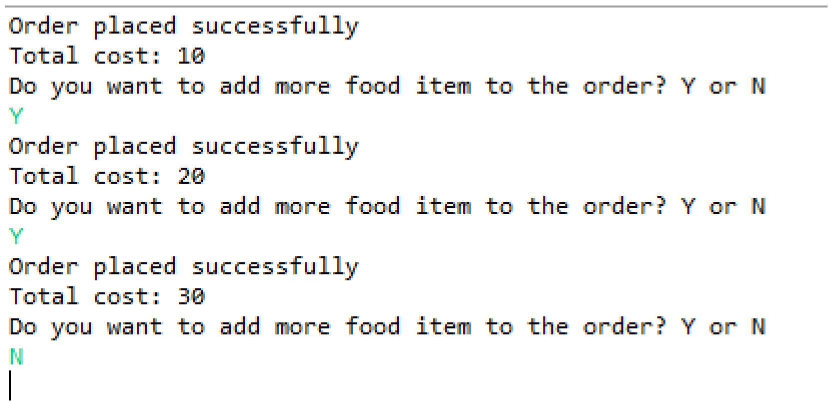
Syntax:

```
do {
  <statements>;
} while (<condition>);
```

Observe the below code. The variable wantToAddFoodItem is initialized to 'N'. In this case, first the code block is executed and then the condition is tested. Hence, the cost is calculated first and then the condition is evaluated. When the condition is false, the loop terminates.

```
//Importing the Scanner class
import java.util.Scanner;
public class Customer {
   public static void main(String[] args) {
         Scanner sc = new Scanner(System.in); // Creating a Scanner object
         int totalCost = 0;
         char wantToAddFoodItem = 'N';
         int unitPrice = 10;
         int quantity = 1;
         do {
               totalCost = totalCost + (quantity * unitPrice);
               System.out.println("Order placed successfully!");
               System.out.println("Total cost: " + totalCost);
               System.out.println("Do you want to add more food items to the order? Y or N");
               String input = sc.next(); // Accepting input from the customer
               // Extracting first character from the input string
               wantToAddFoodItem = input.charAt(0);
         } while (wantToAddFoodItem == 'Y');
         System.out.println("Thank you for ordering the food! It will reach you shortly...");
   }
}
```

O/P



```
Order placed successfully
Total cost: 10
Do you want to add more food item to the order? Y or N
Y
Order placed successfully
Total cost: 20
Do you want to add more food item to the order? Y or N
Y
Order placed successfully
Total cost: 30
Do you want to add more food item to the order? Y or N
N
```

For loop:
Consider 800 metres running event on track. If the track is 400 metres, how many laps the participants should take to complete the race? The participants will start the race from a start line and should take two laps to finish the race. The two laps can be taken as two iterations. In

this case, the number of iterations are known before starting the race. When the number of iterations are known, for loop can be used.

The 'for' loop is used when the number of iterations are known.

Syntax:

```
for (<initialization>; <condition>; <increment/decrement>) {
    <statements>;
}
```

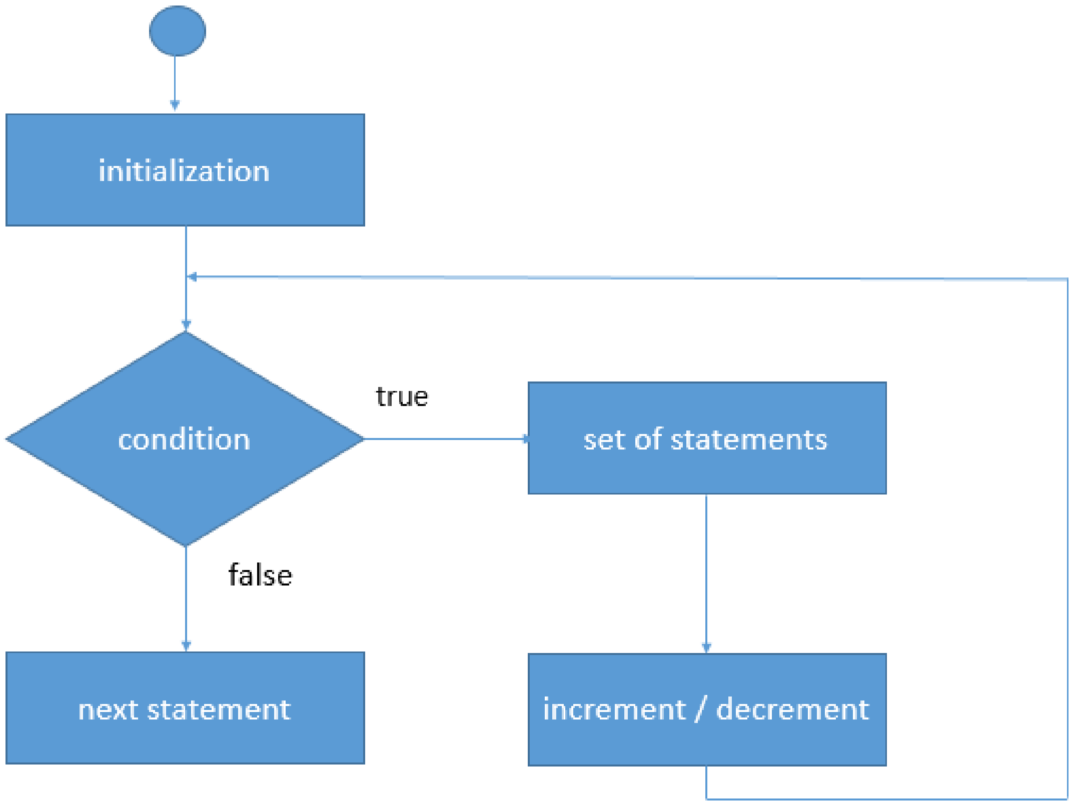
- **Initialization**: It is used for initializing the variables used for checking the condition. It is executed only once and gets executed when the loop starts.

- **Condition**: It is used for checking the condition to decide whether the loop should be terminated or executed. If the condition is true, the body of the loop is executed, else the loop terminates.

- **Increment / Decrement**: It increments or decrements the value of the variable used for checking the condition after every iteration of the loop.

All the three parts of for loop are optional. For instance, the for loop can also be written as *for(;;)* where none of the parts are provided. This for loop will result in infinite loop.

Flowchart:

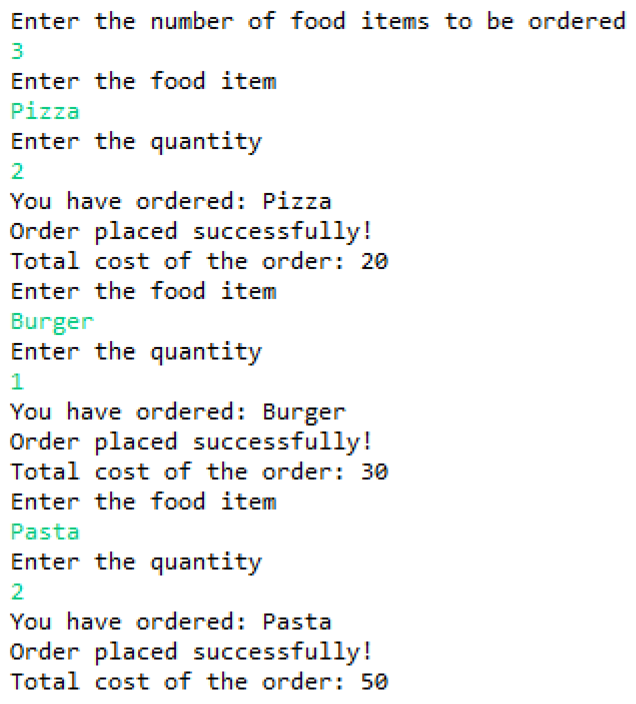


Let us consider that the customer wants to place order for multiple food items and the number of food items to be ordered is already decided. In the code given below, we are assuming that the customer wants to order 3 food items.

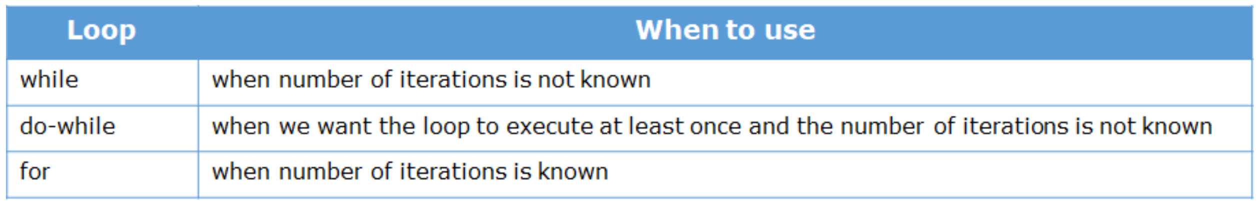
```
//Importing the Scanner class
import java.util.Scanner;
public class Customer {
   public static void main(String[] args) {
          // Create a Scanner object
          Scanner sc = new Scanner(System.in);
          int totalCost = 0;
          int unitPrice = 10;
          System.out.println("Enter the number of food items to be ordered");
          int noFoodItemsToBeOrdered = sc.nextInt();
          for (int counter = 1; counter <= noFoodItemsToBeOrdered; counter++) {
                 System.out.println("Enter the food item");
                 String foodItem = sc.next();
                 System.out.println("Enter the quantity");
                 int quantity = sc.nextInt();
                 System.out.println("You have ordered: " + foodItem);
                 totalCost += unitPrice * quantity;
```

```
                System.out.println("Order placed successfully!");
                System.out.println("Total cost of the order: " + totalCost);
            }
        }
    }
```

O/P



```
Enter the number of food items to be ordered
3
Enter the food item
Pizza
Enter the quantity
2
You have ordered: Pizza
Order placed successfully!
Total cost of the order: 20
Enter the food item
Burger
Enter the quantity
1
You have ordered: Burger
Order placed successfully!
Total cost of the order: 30
Enter the food item
Pasta
Enter the quantity
2
You have ordered: Pasta
Order placed successfully!
Total cost of the order: 50
```



Loop	When to use
while	when number of iterations is not known
do-while	when we want the loop to execute at least once and the number of iterations is not known
for	when number of iterations is known

A nested loop is a loop within another loop, an inner loop within the body of an outer one.

Let us print the following pattern:

1

1 2

1 2 3

1 2 3 4

```
1 2 3 4 5

1 2 3 4 5 6

1 2 3 4 5 6 7

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8 9

1 2 3 4 5 6 7 8 9 10
```

We can create the above pattern using the nested loop.

```
class Numbers {
   public static void main(String[] args) {
         int rows = 10;
         for (int i = 1; i <= rows; ++i) {
                  for (int j = 1; j <= i; ++j) {
                 // print displays the text without adding a new line
                           System.out.print(j + " ");
                  }
                  System.out.println(""); // println displays the text along with a new line
         }
   }
}
```

break statement is used to terminate a loop. After terminating the loop, the next statement following the loop gets executed. In case of break statement written in nested loops, the inner most loop gets terminated and the flow of control continues with the statements of outer loop.

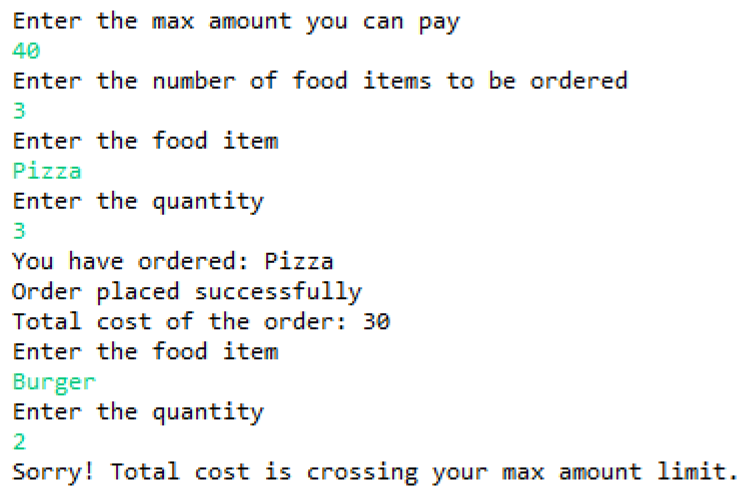
break statement is also used to terminate the execution of a switch case, as already discussed.

Consider the code given below. The number of food items to be ordered is accepted as an input from the customer. Let us assume that the customer has only $40 with him. So, the loop should be terminated when the total cost of food items goes beyond $40. In this scenario, break statement can be used to terminate the loop.

```
//Importing the Scanner class
import java.util.Scanner;
public class Customer {
   public static void main(String[] args) {
         // Create a Scanner object
```

```
        Scanner sc = new Scanner(System.in);
        int totalCost = 0;
        int unitPrice = 10;
        System.out.println("Enter the max amount you can pay");
        int maxAmountCustomerCanPay = sc.nextInt();
        System.out.println("Enter the number of food items to be ordered");
        int noFoodItemsToBeOrdered = sc.nextInt();
        for (int counter = 1; counter <= noFoodItemsToBeOrdered; counter++) {
            System.out.println("Enter the food item");
            String foodItem = sc.next();
            System.out.println("Enter the quantity");
            int quantity = sc.nextInt();
            totalCost += unitPrice * quantity;
            if (totalCost > maxAmountCustomerCanPay) {
                System.out.println("Sorry! Total cost is crossing your max amount limit.");
                break;
            }
            System.out.println("You have ordered: " + foodItem);
            System.out.println("Order placed successfully");
            System.out.println("Total cost of the order: " + totalCost);
        }
    }
}
```

O/P



```
Enter the max amount you can pay
40
Enter the number of food items to be ordered
3
Enter the food item
Pizza
Enter the quantity
3
You have ordered: Pizza
Order placed successfully
Total cost of the order: 30
Enter the food item
Burger
Enter the quantity
2
Sorry! Total cost is crossing your max amount limit.
```

continue statement is used to skip the current iteration of a loop and continue with the next iteration. In case of while and do-while loops, continue statement skips the remaining code of the loop and passes the control to check the loop condition. Whereas in case of for loop, the control goes to the increment section and then the condition is checked.

In the below code, when the value of counter is 3, continue statement is encountered which skips the current iteration and continues with the next iteration.

```
//Importing the Scanner class
import java.util.Scanner;
public class Customer {
   public static void main(String[] args) {
         // Create a Scanner object
         Scanner sc = new Scanner(System.in);
         int totalCost = 0;
         int unitPrice = 10;
         System.out.println("Enter the number of food items to be ordered");
         int noFoodItemsToBeOrdered = sc.nextInt();
         for (int counter = 1; counter <= noFoodItemsToBeOrdered; counter++) {
                  if (counter == 3)
                           continue;
                  System.out.println("Enter the food item");
                  String foodItem = sc.next();
                  System.out.println("Enter the quantity");
                  int quantity = sc.nextInt();
                  System.out.println("You have ordered: " + foodItem);
                  System.out.println("You have successfully placed the order number: "+ counter);
                  totalCost += unitPrice * quantity;
                  System.out.println("Total cost of the order: " + totalCost);
         }
   }
}
```

O/P

```
Enter the number of food items to be ordered
5
Enter the food item
Pizza
Enter the quantity
2
You have ordered: Pizza
You have successfully placed the order number: 1
Total cost of the order: 20
Enter the food item
Burger
Enter the quantity
2
You have ordered: Burger
You have successfully placed the order number: 2
Total cost of the order: 40
Enter the food item
Pasta
Enter the quantity
1
You have ordered: Pasta
You have successfully placed the order number: 4
Total cost of the order: 50
Enter the food item
```

break	continue
terminates a loop or a switch statement	skips the remaining statements of the loop for the current iteration
can be used with loops and switch	can be used only within loops