# Programming Fundamentals

## Looping Statements

Week 5| Iresh Bandara

## Learning Outcomes

- Covers part of LO1, LO2 & LO3 for Module

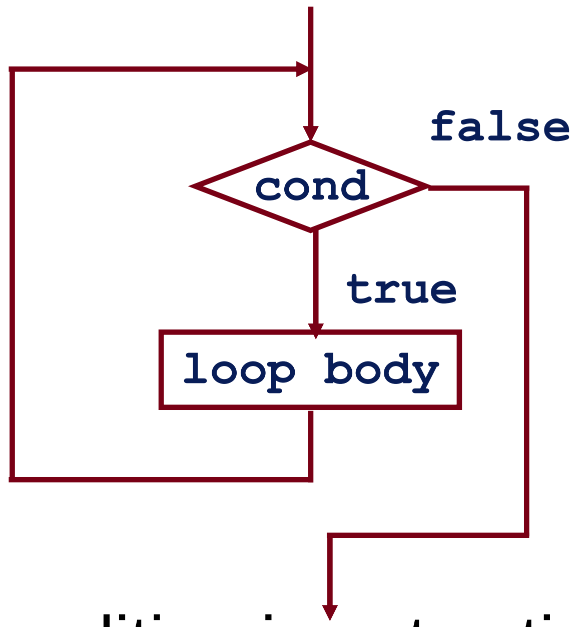
- On completion of this lecture, students are expected to be able to:
  - Recognize the appropriate places where loops are required.
  - Apply any looping statements to a given problem.
  - Use breaks and continues in java to control the program.

## Loops

- Two kinds of loops
  - Finite loop: The statements in the block may be executed any number of times, from zero to infinite number.
  - Infinite loop: A loop that continues forever.
- A loop consist of;
  - Body of the loop
  - Control statement

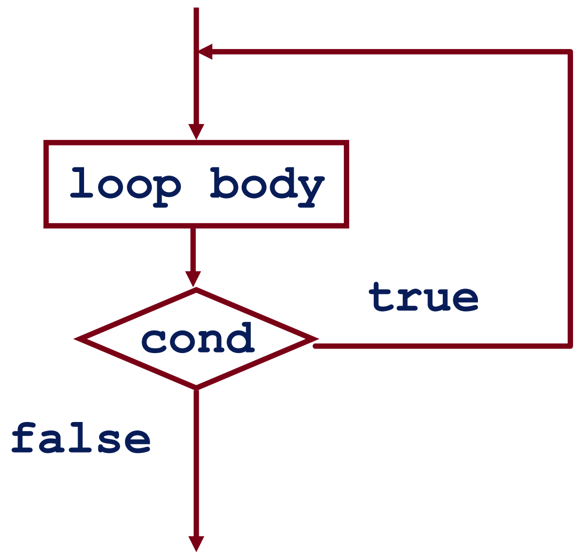
# Loop Control Structures

**Entry Control loop**



- If condition is not satisfied body of the loop is never executed.

**Exit Control loop**



- The body of the loop is executed unconditionally for the 1st time.

## Loop Control Structures

- The **while** loop
- The **do while** loop
- The **for** loop

## The `while` Loop

```
initialization;
while (condition)  {
      statement1;
      statement2;
      ...
      statementN;
}
```

**condition** is any logical expression, as in if

The body of the loop

- This is an **entry-controlled** loop

# The `while` Loop contd…

- **Initialization**: The variables tested in the condition must be initialized to some values. If the condition is false at the outset, the loop is never entered.
- **Testing**: The condition is tested before each iteration. If false, the program continues with the first statement after the loop.
- **Change**: At least one of the variables tested in the condition must change within the body of the loop.

## Try this…

- Write a program that uses a **while** loop to print the sum of integers from 1 to 10.

```
sum = 0
i = 1
while i <= 10:
    sum += i
    i += 1
print("The sum of integers from 1 to 10 is:", sum)
```

```
public class Main {
    public static void main(String[] args) {
        int sum = 0;
        int i = 1;
        while (i <= 10) {
            sum += i;
            i++;
        }
        System.out.println("The sum of integers from 1 to 10 is: " + sum);
    }
}
```

## The do-while Loop

```
initialization;
do  {
    statement1;
    statement2;
    ...
    statementN;
} while ( condition );
```

The code runs through the body of the loop at least once

if condition is false, the next iteration is not executed

- This is an **exit-controlled** loop

## Try this…

- Write a program that uses **do-while** loop to print the sum of squares of integers from 1 to 10.

```
sum = 0
i = 1
while True:
    sum += i**2
    i += 1
    if i > 10:
        break
print("The sum of squares of integers from 1 to 10 is:", sum)
```

```
public class Main {
    public static void main(String[] args) {
        int sum = 0;
        int i = 1;
        do {
            sum += i * i;
            i++;
        } while (i <= 10);
        System.out.println("The sum of squares of integers from 1 to 10 is: " + sum);
    }
}
```

## The for Loop

- **for** is a shorthand that combines in one statement initialization, testing, and change:

```
for(initialization;condition;change )
{
      statement1;
      statement2;
      ...
      statementN;
   }
```

- This is an **entry-controlled** loop.

## Try this…

1. Write a program that uses **for** loop to generate even numbers that are less than 10.

i.e.

```
0
2
4
6
8
```

```
public class Number{
    public static void main(String[] args){
        for(int i = 0; i<10; i+=2){
            System.out.println(i);
        }
    }
}
```

## Summary of Loop Control Structures

**for**

```
for
   (n=1;n<=10;++n
   )
{
…………….. .
…………….. .
}
```

**while**

```
n = 1;
while (n<=10)
{
……………….. .
……………….. .
n = n+1;
}
```

**do while**

```
n = 1;
do
{
…………….
…………….
n = n+1;
}
while (n<=10);
```

## Nested Loops

- A **nested loop** is a looping construct that appears as one of the statements within the body of another loop construct.
- When using such a form;
  - Do need to take care that the conditions used to terminate each loop
  - do not interact in a destructive manner!

## Try this…

- Write a program to print a multiplication table using **for** loops.

```
import java.util.Scanner;

public class MultiplicationTable {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of rows: ");
        int rows = sc.nextInt();

        System.out.print("Enter the number of columns: ");
        int cols = sc.nextInt();

        for (int i = 1; i <= rows; i++) {
            for (int j = 1; j <= cols; j++) {
                System.out.print(i * j + "\t");
            }
            System.out.println();
        }
    }
}
```

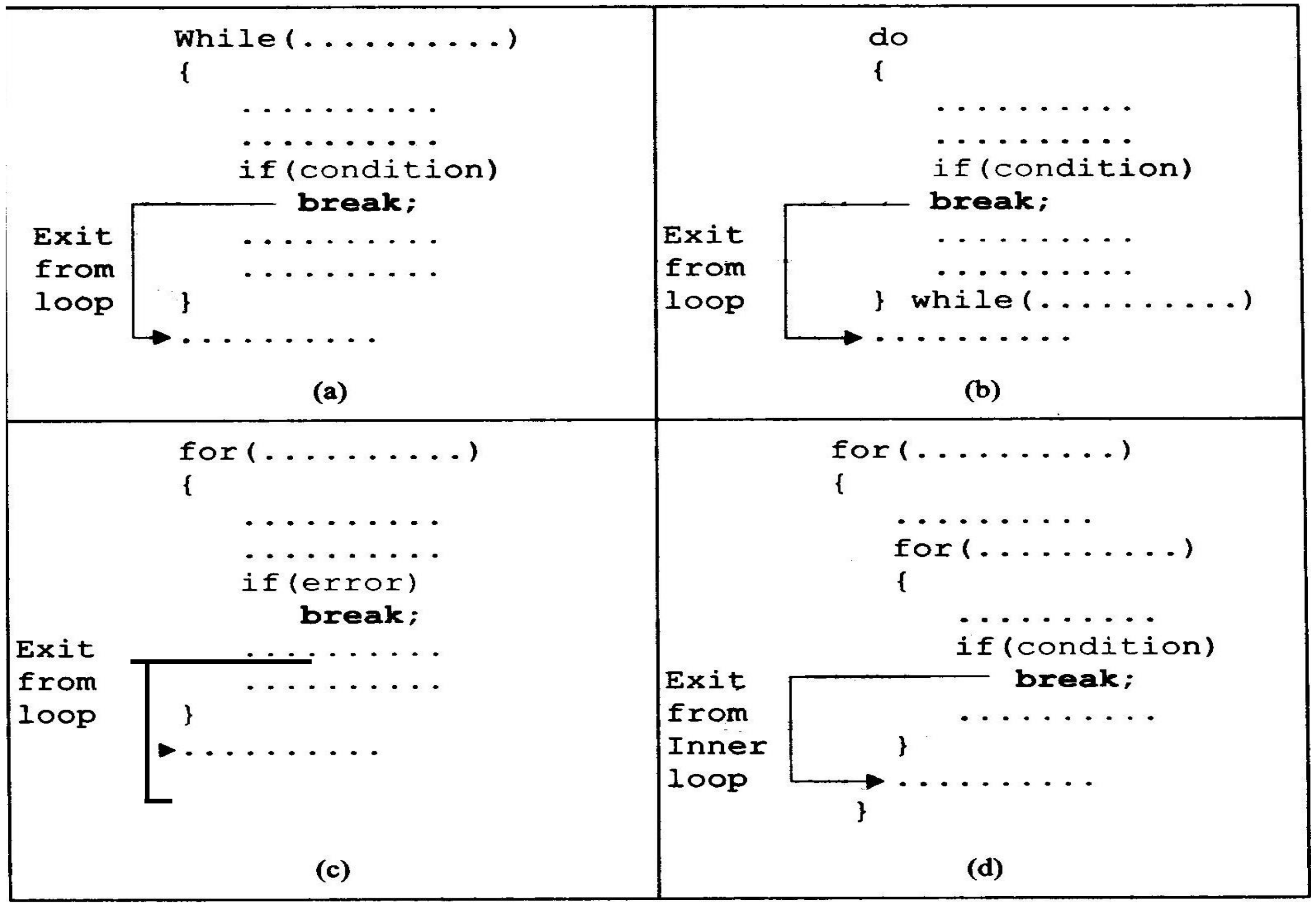
## Jumps in Loops

- **break** – To jump out of a loop
- **continue** – To skip a part of a loop

## `break` Statement

- Used to achieve an early exit from a loop.
- Used within **`while`**, **`do while`**, **`for`** and **`switch`**.
- **`break`** results an immediate **exit from the loop containing it** (nearest loop).
- **`break`** will exit only a single loop.

# break Statement



```
While(..........)
{
    ..........
    ..........
    if(condition)
        break;
    ..........
    ..........
}
..........
```

(a)

```
do
{
    ..........
    ..........
    if(condition)
    break;
    ..........
    ..........
} while(..........)
..........
```

(b)

```
for(..........)
{
    ..........
    ..........
    if(error)
        break;
    ..........
    ..........
}
..........
```

(c)

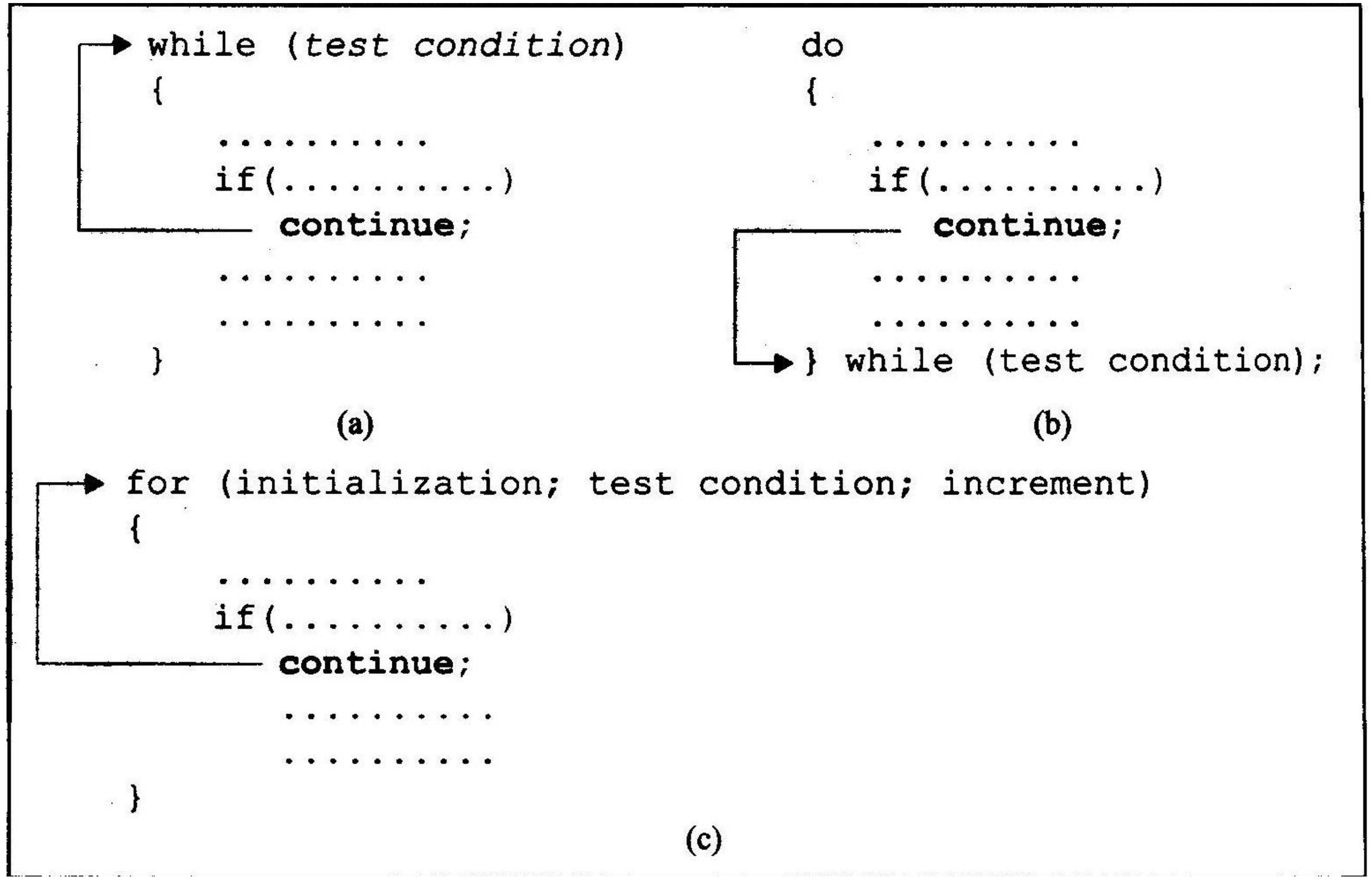
```
for(..........)
{
    ..........
    for(..........)
    {
        ..........
        if(condition)
            break;
        ..........
    }
    ..........
}
```

(d)

## `continue` Statement

- Restarts the current loop.
- Causes the loop to be **continues with the next iteration** after skipping any statements in between.
- Used within **`while`**, **`do while`** and **`for`**.

# continue Statement



```
while (test condition)
{
    ..........
    if(..........)
      continue;
    ..........
    ..........
}
```

(a)

```
do
{
    ..........
    if(..........)
      continue;
    ..........
    ..........
} while (test condition);
```

(b)

```
for (initialization; test condition; increment)
{
    ..........
    if(..........)
      continue;
      ..........
      ..........
}
```

(c)

## Labeled Loops

- To jump outside a loop that is outside the current one,

  **OR**

- To continue a loop that is outside the current one.

# Example

```
outer: for (int m=1; m<11; m++)   {
             System.out.println();
            for (int n=1; n<11; n++)
            {
            System.out.print (" " + m*n) ;
            if ( n == m )
                 continue outer;
            }
      }
```

- The `continue` statement terminates the inner loop **when n = m** and continues with the next iteration of the outer loop (after counting m).

## Try this

- It is required to write a program that reads employee information iteratively. (Employee name, emp_no, job title )
- Additionally users are allowed to decide the termination point.
- This means that the program prompts user for entering employee information.
- Soon after the input process the input values are displayed for the user.
- Then ask the user whether he wants to continue or exit. For example, you may ask users to enter ‘Yes’ to continue and ‘No’ to terminate.

## Summery

- Two kinds of loops Finite loops, Infinite loops
- The while loop, the do while loop and the for loop are three Loop Control Structures
- A nested loop is a looping construct that appears as one of the statements within the body of another loop construct
- Two jumps in the loops, break – To jump out of a loop, continue – To skip a part of a loop

# Thank you