

# CSC 125

## Object Oriented Programming

Ch05\_Loops

Dr. Fadi Alzhouri



# Loop Flow Control

- A loop is a programming construct that allows a set of instructions (or a block of code) to be executed repeatedly, either a specific number of times or until a certain condition is met.
- **Repetition:** Loops enable the same instructions to run multiple times without needing to write the code redundantly.
- **Importance:** Loops help reduce redundancy, enhance code efficiency, and manage repetitive tasks.
- Java provides three types of loop statements:
  - **While** Loop
  - **Do-While** Loop
  - **For** Loop

# While Loop

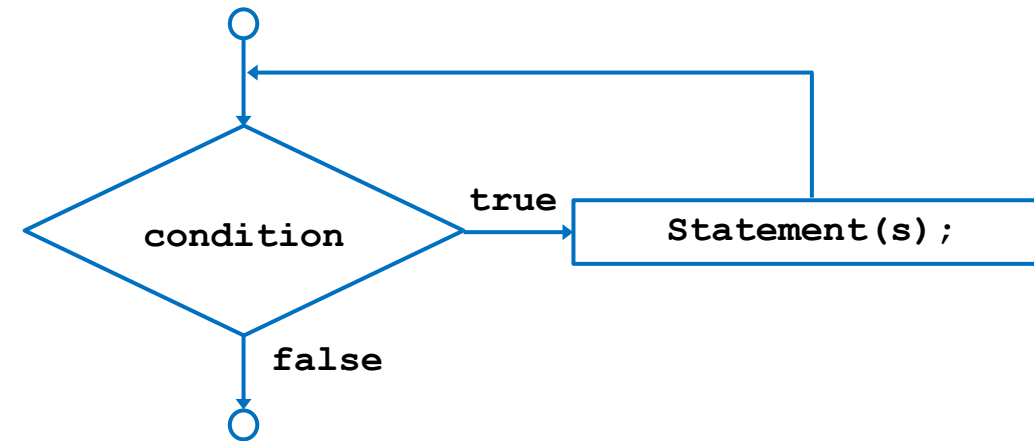
- Action repeated while some condition remains true
- A while loop repeated until the condition becomes false
- Syntax:

```
While (Boolean_Expression)  
    statement;
```

Single statement  
body

```
While (Boolean_Expression) {  
    statement;  
    statement;  
}
```

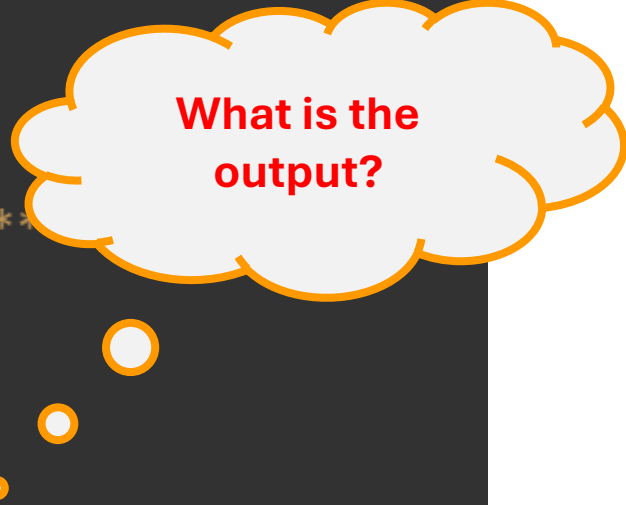
Multistatement body



- Note: You need to put braces if you have more than one statement in the while loop

# While loop (cont.)

```
/* *****  
Iteration statements  
Author: Dr. Fadi Alzhouri  
Example 17: While statement  
***** */  
public class Whileloop  
{  
    public static void main(String[] args) {  
  
        int i = 1;  
        while(i<=3){  
            System.out.println("i= " + i + " Welcome to Java");  
            ++i;  
        }  
        System.out.println("i= " + i );  
    }  
}
```



What is the output?

# While loop (cont.)

```
/**
 * Iteration statements
 * Author: Dr. Fadi Alzhouri
 * Example 17: While statement
 */
public class Whileloop
{
    public static void main(String[] args) {

        int i = 1;
        while(i<=3){
            System.out.println("i= " + i + " Welcome to Java");
            ++i;
        }
        System.out.println("i= " + i );
    }
}
```

At the end of the while loop, i is 4

```
i= 1 Welcome to Java
i= 2 Welcome to Java
i= 3 Welcome to Java
i= 4
```

# Iterations

- In general, we have two types of iterations:
  - Counter-Controlled Iterations
    - Usage: The number of iterations is **known**
    - The Loop repeated until the **counter** reaches a certain value
  - Sentinel-Controlled Iterations
    - Usage: The number of iterations is **unknown**
    - The Loop ends when the sentinel value is found
    - The **sentinel** value is chosen so it cannot be confused with regular input.

# While: Counter-Controlled

- Example: Write a program to find the total cost of purchases in your shopping cart. Assume there are 4 items in the cart.



# While: Counter-Controlled (cont.)

- Algorithm:
  - Define a new Scanner to read inputs.
  - Declare and initialize any required variables
    - Set total to zero
    - Set item counter to 1
    - Declare item price
  - While the counter is less than or equal 4
    - Prompt the user to Input the price of the next item
    - Add the price to the total
    - Increment item counter
  - Print the total cost



# While: Counter-Controlled (cont.)

```
/**
 * Loop statements
 * Author: Dr. Fadi Alzhouri
 * Example 20: While statement(counter)
 */
import java.util.Scanner;
public class ShoppingCart {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        double totalCost = 0;
        double itemPrice = 0;
        int itemCount = 1;

        while (itemCount <=4) {
            System.out.print("Enter the price of item_ "+ itemCount + ": ");
            itemPrice = keyboard.nextDouble();

            totalCost += itemPrice;
            itemCount++;

        }
        System.out.println("Total cost of purchases: $" + totalCost);
    }
}
```

Counter

Update the counter

# While: Counter-Controlled (cont.)

```
/******
```

```
Loop statements
```

```
Author: Dr. Fadi Alzhouri
```

```
Example 20: While statement(counter)
```

```
*****
```

```
import java.util.Scanner;
```

```
public class ShoppingCart {
```

```
    public static void main(String[] args) {
```

```
        Scanner keyboard = new Scanner(System.in);
```

```
        double totalCost = 0;
```

```
        double itemPrice = 0;
```

```
        int itemCount = 1;
```

```
        while (itemCount <=4) {
```

```
            System.out.print("Enter the price of item_ "+ itemCount + ": ");
```

```
            itemPrice = keyboard.nextDouble();
```

```
            totalCost += itemPrice;
```

```
            itemCount++;
```

```
        }
```

```
        System.out.println("Total cost of purchases: $" + totalCost);
```

```
    }
```

```
}
```

```
Enter the price of item_ 1: 2.500
Enter the price of item_ 2: 3.990
Enter the price of item_ 3: 1.200
Enter the price of item_ 4: 7.150
Total cost of purchases: $14.84
```

It is outside the  
While loop

# While: Sentinel-Controlled

- Example: Rewrite the previous program to find the total cost of purchases in your shopping cart. Assume you don't know the number of items in advance.

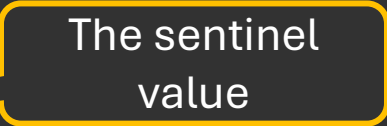


# While: Sentinel-Controlled (cont.)

```
/**
 * Loop statements
 * Author: Dr. Fadi Alzhouri
 * Example 21: While statement(sentinal value)
 */
```

```
import java.util.Scanner;
public class WhileSen {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        double totalCost = 0;
        double itemPrice = 0;

        while (itemPrice > -1) {
            totalCost += itemPrice;
            System.out.print("Enter the price of the item or -1 to pay: ");
            itemPrice = keyboard.nextDouble();
        }
        System.out.println("Total cost of purchases: $" + totalCost);
    }
}
```



# While: Sentinel-Controlled (cont.)

```
/**
 * *****
 */
```

*Loop statements*

*Author: Dr. Fadi Alzhouri*

*Example 21: While statement(sentinal va*

*\*\*\*\*\**

```
import java.util.Scanner;
```

```
public class WhileSen {
```

```
    public static void main(String[] args) {
```

```
        Scanner keyboard = new Scanner(System.in);
```

```
        double totalCost = 0;
```

```
        double itemPrice = 0;
```

```
        while (itemPrice > -1) {
```

```
            totalCost += itemPrice;
```

```
            System.out.print("Enter the price of the item or -1 to pay: ");
```

```
            itemPrice = keyboard.nextDouble();
```

```
        }
```

```
        System.out.println("Total cost of purchases: $" + totalCost);
```

```
    }
```

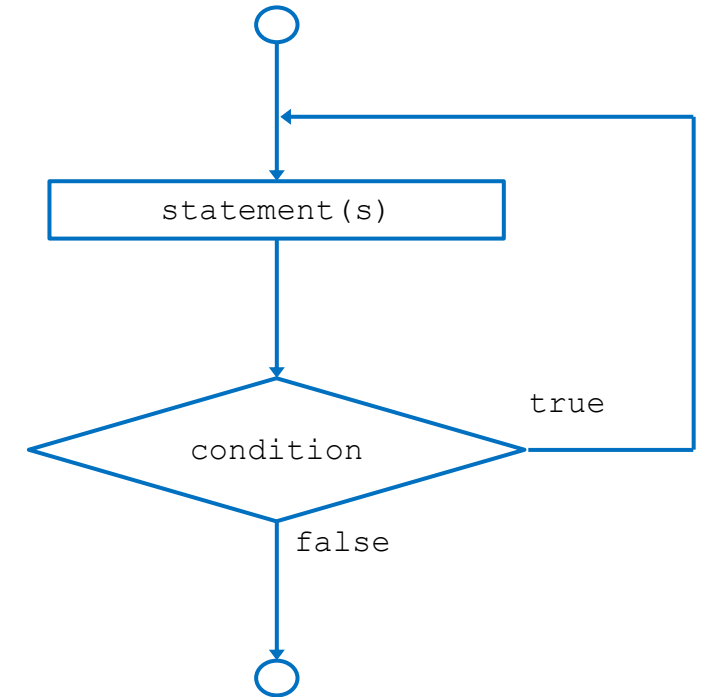
```
}
```

```
Enter the price of the item or -1 to pay: 2.500
Enter the price of the item or -1 to pay: 3.990
Enter the price of the item or -1 to pay: 1.200
Enter the price of the item or -1 to pay: -1
Total cost of purchases: $7.69
```

# Do-While Loop

- Similar to the **While** statement but
  - Makes loop continuation condition at the **end**, not the beginning
  - Loop body executes at **least once**
- Syntax:

```
do {  
    statement(s) ;  
}while (boolean_condition) ;
```



# Do-While Loop (cont.)

- What is the output of the following code?

```
/******  
Loop statements  
Author: Dr. Fadi Alzhouri  
Example 22: Do-While statement  
*****/  
import java.util.Scanner;  
public class DoWhile {  
    public static void main(String[] args) {  
        int i =1;  
        do{  
            System.out.println("Line: " + i);  
            i++;  
        }while(i <= 4);  
  
        System.out.println("i = "+ i);  
    }  
}
```

# Do-While Loop (cont.)

```
/**
 * Loop statements
 * Author: Dr. Fadi Alzhouri
 * Example 22: Do-While statement
 */
import java.util.Scanner;
public class Dowhile {
    public static void main(String[] args) {
        int i =1;
        do{
            System.out.println("Line: " + i);
            i++;
        }while(i <= 4);

        System.out.println("i = " + i);
    }
}
```

```
Line: 1
Line: 2
Line: 3
Line: 4
i = 5
```



# For loop

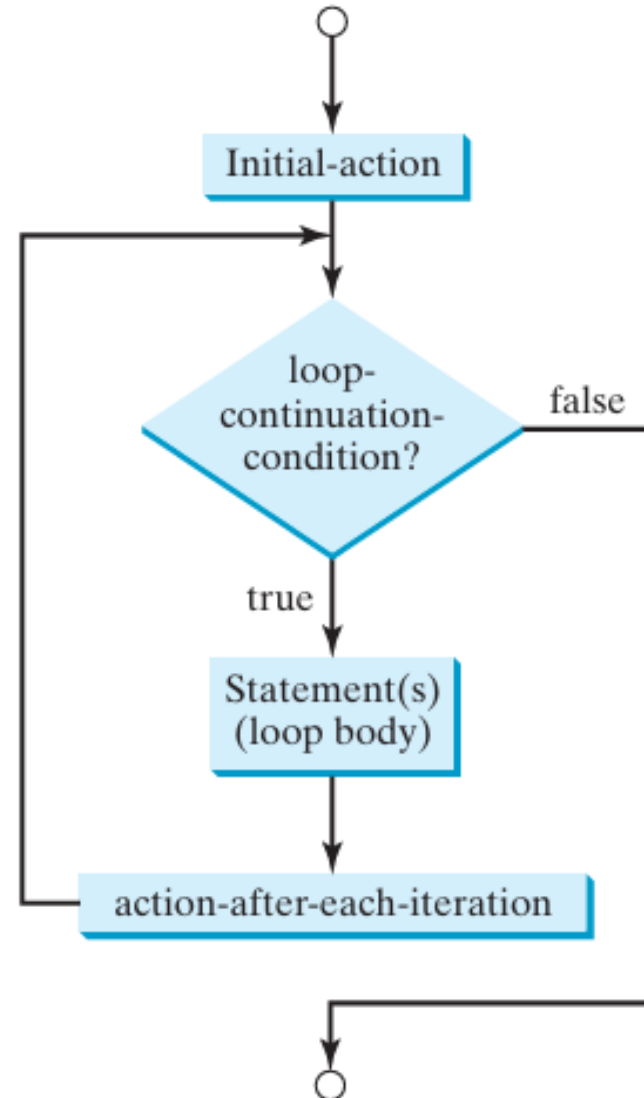
- It is used when the number of iterations is known in advance.
- Syntax:

```
for (statement 1; statement 2; statement 3) {  
    statement(s); // code block to be executed  
}
```

- **Statement 1** is executed (one time) before the execution of the loop body. It is called an **initialization statement**.
- **Statement 2** defines the **condition** for executing the loop body.
- **Statement 3** is executed (every time) after the loop body has been executed. Statement after each **iteration**.

# For loop (cont.)

- Flowchart



# For loop (cont.)

- What is the output of the following code?

```

/*****
Loop statements
Author: Dr. Fadi Alzhouri
Example 23: for statement
*****/

import java.util.Scanner;
public class Forloop {
    public static void main(String[] args) {
        int i;
        for(i=1 ; i<=4 ; i++)
            System.out.println("Line: " + i);

        System.out.println("i = " + i);
    }
}

```

# For loop (cont.)

```
/******  
Loop statements  
Author: Dr. Fadi Alzhouri  
Example 23: for statement  
*****/  
import java.util.Scanner;  
public class Forloop {  
    public static void main(String[] args) {  
        int i;  
        for(i=1 ; i<=4 ; i++)  
            System.out.println("Line: " + i);  
  
        System.out.println("i = " + i);  
    }  
}
```

the body of  
for loop

Outside the loop

```
Line: 1  
Line: 2  
Line: 3  
Line: 4  
i = 5
```

# For loop (cont.)

```

/*****
Loop statements
Author: Dr. Fadi Alzhouri
Example 23: for statement
*****/

import java.util.Scanner;
public class Forloop {
    public static void main(String[] args) {
        int i;
        for(i=1 ; i<=4 ; i++){
            System.out.println("Line: " + i);
        }
        System.out.println("i = " + i);
    }
}

```

It is better to use  
braces (curly brackets)

```

Line: 1
Line: 2
Line: 3
Line: 4
i = 5

```

# Nested Loops

- A nested loop is a loop inside another loop.
- The inner loop runs completely for each iteration of the outer loop.
- Useful for working with multi-dimensional data structures, such as 2D arrays.

```
for (initialization; condition; inc/dec) {  
    for (initialization; condition; inc/dec) {  
        // Inner Loop code  
    }  
    // Outer Loop code  
}
```

# Exercise

- Write a program to create a multiplication table of numbers from 1 to 4.

## Exercise (cont.)

- Write a program to create a multiplication table of numbers from 1 to 4.

```
for (int i = 1; i <= 4; i++) {  
    System.out.println("Multiplication Table for " + i + ":");  
  
    for (int j = 1; j <= 4; j++) {  
        System.out.println(i + " x " + j + " = " + (i * j));  
    }  
    System.out.println(); // Print a blank line for better formatting  
}
```



# Exercise (cont.)

- Output:

```
for (int i = 1; i <= 4; i++) {  
    System.out.println("Multiplication Table for " + i + ":");  
  
    for (int j = 1; j <= 4; j++) {  
        System.out.println(i + " x " + j + " = " + (i * j));  
    }  
    System.out.println(); // Print a blank line for better formatting  
}
```

Multiplication Table for 1:

1 x 1 = 1

1 x 2 = 2

1 x 3 = 3

1 x 4 = 4

Multiplication Table for 2:

2 x 1 = 2

2 x 2 = 4

2 x 3 = 6

2 x 4 = 8

Multiplication Table for 3:

3 x 1 = 3

3 x 2 = 6

3 x 3 = 9

3 x 4 = 12

Multiplication Table for 4:

4 x 1 = 4

4 x 2 = 8

4 x 3 = 12

4 x 4 = 16

# References

- **Introduction to Java Programming, Brief Version, Global Edition, 11th edition**, Published by Pearson (June 21, 2018) © 2018