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

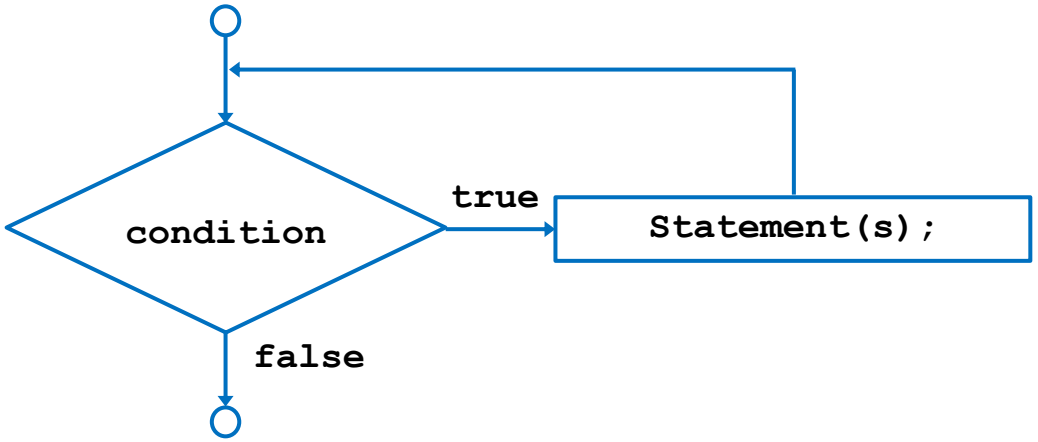
While Loop

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

```
While(Boolean_Expression)
        statement;

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

Single statement body

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.)

```java
/*****************************************************************
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.)

```java
/*****************************************************
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.)

```java
/*****************************************************************
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);
    }
}
```

The sentinel value

While: Sentinel-Controlled (cont.)

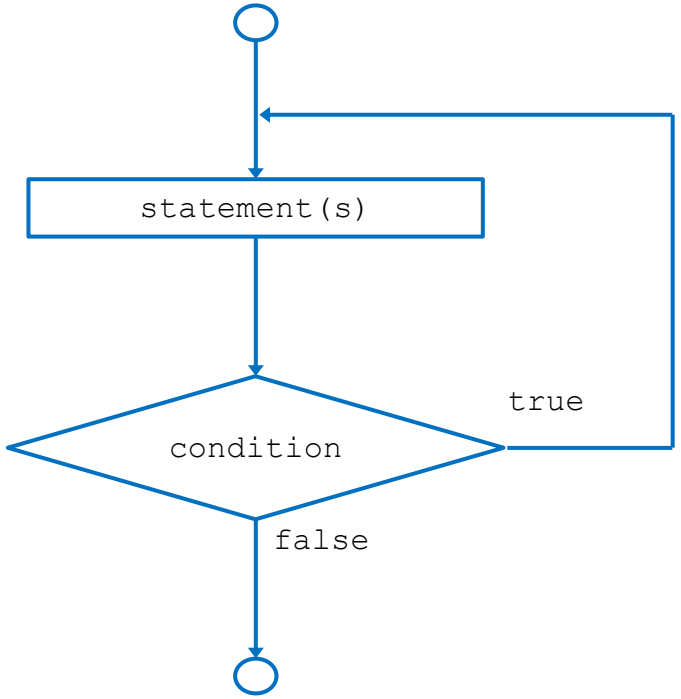
```
/*****************************************************************
Loop statements
Author: Dr. Fadi Alzhouri
Example 21: While statement(sentinal va
*****************************************
import java.util.Scanner;
public class WhileSen {
    public static void main(String[] args) {
        Scanner 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?

```java
/*****************************************************************
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.)

```java
/*****************************************************
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

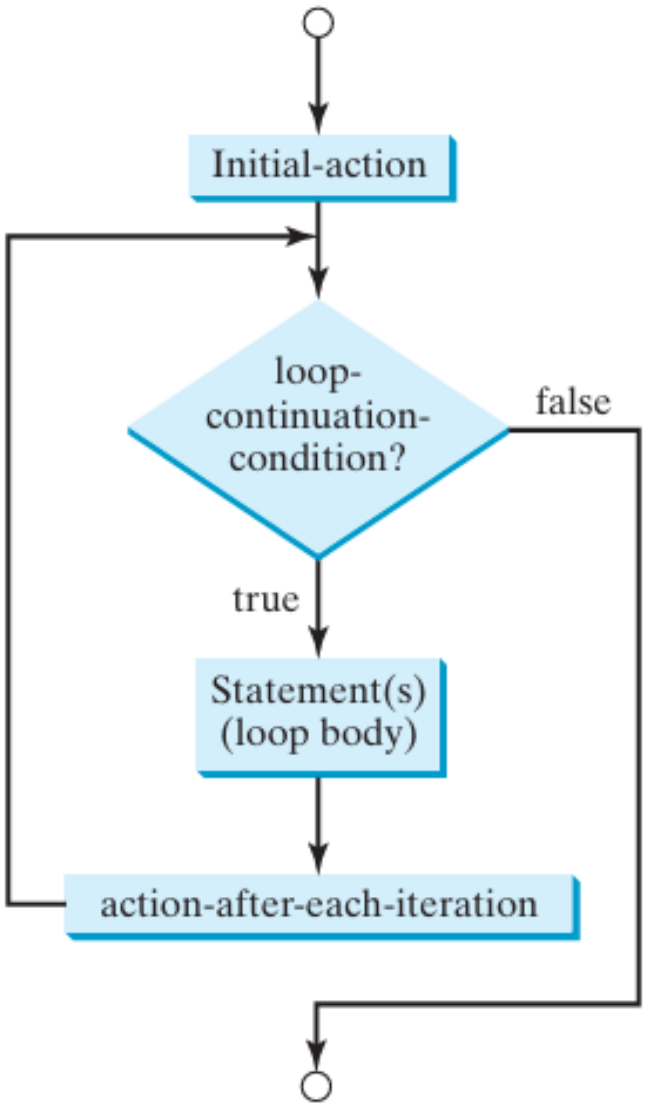
- It is used when the number of iterations is known in advance.
- Syntax:

```
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?

```java
/*****************************************************
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.)

```java
/*****************************************************************
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.)

```java
/*****************************************************************
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.

```java
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:

```java
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
```

Exercise (cont.)

- Example 2: Rewrite the code to arrange the output as shown

```
i\j        1        2        3        4        5        6        7        8        9
1 |         1        2        3        4        5        6        7        8        9

2 |         2        4        6        8       10       12       14       16       18

3 |         3        6        9       12       15       18       21       24       27

4 |         4        8       12       16       20       24       28       32       36

5 |         5       10       15       20       25       30       35       40       45

6 |         6       12       18       24       30       36       42       48       54

7 |         7       14       21       28       35       42       49       56       63

8 |         8       16       24       32       40       48       56       64       72

9 |         9       18       27       36       45       54       63       72       81
```

Exercise (cont.)

- Example 3: Guess the output?

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 5; j++)
        System.out.print("*");
        System.out.println();
}
```

Exercise (cont.)

- Example 3: Guess the output?

```java
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 5; j++)
        System.out.print("*");
        System.out.println();
}
```

It is outside the inner For loop

```
*****
*****
*****
*****
*****
```

Exercise (cont.)

- Example 4: Write a program to create the following output.

```
*
**
***
****
*****
```

For vs. While

- Any for loop can be written using a while loop and vice versa.
- Convert the following for loop into While:

```
for (int i = 1; i <= 5; i++)
        System.out.print(" *\n");
```

For vs. While (cont.)

- Any for loop can be written using a while loop and vice versa.
- Convert the following for loop into While:

```
for (int i = 1; i <= 5; i++)
        System.out.print(" *\n");
```

Statement 1

Statement 2

```
int i=1;
while(i<=5){
    System.out.print(" *\n");
    i++;
}
```

Statement 3

Nested loops vs. Performance

- How many times is the inner loop executed in total?

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        System.out.print("*"); // inner loop code
    }
}
```

Nested loops vs. Performance

- How many times is the inner loop executed in total?
- If the outer loop runs n times and the inner loop runs m times, the total number of iterations is n * m.
- More iterations lead to longer execution times.
- For multiple nested loops, performance can degrade significantly

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        System.out.print("*"); // inner loop code
    }
}
```

Unconditional Branching

- Unconditional branching allows control to jump to different parts of the code without any condition.
- Helps manage the execution flow in loops, improving code flexibility and clarity.
- The break and continue keywords provide additional controls in a loop.

Unconditional Branching: break Statement

- The break statement terminates the nearest enclosing loop or switch statement.
- Used to exit a loop or switch body prematurely.
- Example:

```java
for (int i = 1; i <= 10; i++) {
    if (i == 5){
        break; // Exit the loop when i equals 5
    }
    System.out.print(i + " , ");
}
```

```
1 , 2 , 3 , 4 ,
```

Unconditional Branching: continue Statement

- The continue statement skips the current iteration of the nearest enclosing loop and proceeds to the next iteration.
- Used to bypass certain conditions without exiting the loop entirely.
- Example:

```
for (int i = 1; i <= 10; i++) {
    if (i == 5){
        continue; // skip 5
    }
    System.out.print(i + " , ");
}
```

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

References

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