while, do-while, and exception handling for robust loops

teacher: Майер С.Ф.
sfmayer@sfedu.ru

While Loops: Repetition with Dynamic Conditions

Use **while** when the number of iterations is unknown

Syntax of the `while` Loop

```
while (condition) {
        // loop body
}
```

Core rule:

- Condition checked **before** each iteration.
- If `false` initially → body never executes.

Critical requirement:

➢ update the condition variable **inside** the loop body to avoid infinite loops.

Counter-Controlled while Loops

Example 1

When to use: exact number of iterations known in advance.

Task:
Write a program that uses a **while** loop to print numbers **from 1 to 5**, each on a new line. Then print "Done!" after the loop.

Expected Output:

```
1
2
3
4
5
Done!
```

Counter-Controlled while Loops

Example 1

```
public class CounterControlled {
    public static void main(String[] args)
{
        int counter = 1;

        while (counter <= 5) {
            System.out.println(counter);
            counter++;
        }

        System.out.println("Done!");
    }
}
```

Expected Output:

1
2
3
4
5
Done!

Counter-Controlled while Loops

Example 2

Task:

Write a program that processes payroll for exactly **4 employees**. For each employee, the program asks for the number of hours worked and their hourly rate in rubles. It then calculates and displays their salary (hours × rate). After processing all 4 employees, the program prints a completion message.

Use a counter-controlled while loop.

Expected Output:

```
Enter hours worked: 40
Enter hourly rate in rubles: 500
Salary is 20000.0 rub.

Enter hours worked: 35
Enter hourly rate in rubles: 450
Salary is 15750.0 rub.

Enter hours worked: 42
Enter hourly rate in rubles: 520
Salary is 21840.0 rub.

Enter hours worked: 38
Enter hourly rate in rubles: 480
Salary is 18240.0 rub.

All employees processed.
```

Example 2

```
import java.util.Scanner;
public class EmployeePayroll {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int count_emp = 0;

        while (count_emp < 4) {
            System.out.print("Enter hours worked: ");
            double hours = input.nextDouble();

            System.out.print("Enter hourly rate in rubles: ");
            double rate = input.nextDouble();

            double pay = hours * rate;
            System.out.println("Salary is " + pay + " rub.\n");

            count_emp ++;
}
        System.out.println("All employees processed.");
        input.close();
    }
}
```

Expected Output:

```
Enter hours worked: 40
Enter hourly rate in rubles: 500
Salary is 20000.0 rub.

Enter hours worked: 35
Enter hourly rate in rubles: 450
Salary is 15750.0 rub.

Enter hours worked: 42
Enter hourly rate in rubles: 520
Salary is 21840.0 rub.

Enter hours worked: 38
Enter hourly rate in rubles: 480
Salary is 18240.0 rub.

All employees processed.
```

Loops with General Condition (Condition-Controlled)

Example 1

Concept: used when the exact number of repetitions is **unknown** before execution.

Task:
Write a program that continuously asks the user to enter integers. The program should multiply each entered number with the current product and display the running total after each multiplication. The loop should continue **while the product is less than 10,000**. Once the product reaches or exceeds 10,000, the loop stops and displays the final product.

Expected Output:

```
Current product: 1
Enter an integer: 10
Current product: 10
Enter an integer: 20
Current product: 200
Enter an integer: 5
Current product: 1000
Enter an integer: 15
Final product: 15000
```

Loops with General Condition (Condition-Controlled)

```
import java.util.Scanner;
public class ComputeProduct {
    public static void main(String[] args) {
        int n, product = 1;
        Scanner input = new Scanner(System.in);

        while (product < 10000) {
            System.out.println("Current product: " + product);
            System.out.print("Enter an integer: ");
            n = input.nextInt();
            product *= n;
        }
        System.out.println("Final product: " + product);
        input.close();
    }
}
```

Expected Output:

```
Current product: 1
Enter an integer: 10
Current product: 10
Enter an integer: 20
Current product: 200
Enter an integer: 5
Current product: 1000
Enter an integer: 15
Final product: 15000
```

Condition-Controlled loop. Example 2

Program logic: keep asking until correct answer (`number1 + number2 = ?`).

Task:
Write a Java program that generates two random integers between 0 and 9. It then asks the user to enter their **sum**. If the user's answer is incorrect, the program should keep asking the same question until the correct answer is provided. Once the user answers correctly, the program prints a success message.

Expected Output:

```
What is 7 + 3? 9
Wrong answer. Try again.
What is 7 + 3? 11
Wrong answer. Try again.
What is 7 + 3? 10
You got it!
```

Condition-Controlled loop. Example 2

Program logic: keep asking until correct answer (`number1 + number2 = ?`).

Code snippet :

```
int number1 = (int)(Math.random() * 10);
int number2 = (int)(Math.random() * 10);
System.out.print("What is " + number1 + " + " + nu
int answer = input.nextInt();
while (number1 + number2 != answer) {
    System.out.println("Wrong answer. Try again.");
    System.out.print("What is " + number1 + " + " + number2 + "? ");
    answer = input.nextInt();
}
System.out.println("You got it!");
```

Expected Output:

What is 7 + 3? 9
Wrong answer. Try again.
What is 7 + 3? 11
Wrong answer. Try again.
What is 7 + 3? 10
You got it!

Sentinel-Controlled Loops

Concept: exit when a special sentinel value is read.

Rules for sentinel:

- Must be unique (not appear in valid data).
- Clearly communicated to user.

Example 1

Task:
Write a Java program that asks the user to enter numbers and calculates their **sum**. The program should stop when the user enters **0** (the sentinel value). Then print the total sum.

Expected Output:

```
Enter a number (0 to stop): 3
Enter a number (0 to stop): 5
Enter a number (0 to stop): 2
Enter a number (0 to stop): 0
Sum of all numbers: 10
```

Solution

```java
import java.util.Scanner;

public class SentinelLoop {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int number;
        int sum = 0;

        System.out.print("Enter a number (0 to stop): ");
        number = scanner.nextInt();

        while (number != 0) {
            sum += number;
            System.out.print("Enter a number (0 to stop): ");
            number = scanner.nextInt();
        }

        System.out.println("Sum of all numbers: " + sum);
        scanner.close();
    }
}
```

Expected Output:

```
Enter a number (0 to stop): 3
Enter a number (0 to stop): 5
Enter a number (0 to stop): 2
Enter a number (0 to stop): 0
Sum of all numbers: 10
```

do-while Loop

The do-while Loop

Key difference from `while`**:** body executes **at least once** before condition check.

Syntax:

```
do {
        // loop body
} while (condition);
```

Best use case: input validation (user must enter data at least once).

do-while guarantees at least one execution.

Example 1

Task:

Create a program that asks the user to input integers until they enter 0. After entering 0, the program should display the sum of all entered numbers (0 is not included in the sum). Use a do-while loop.

Expected Output:

Enter a number (0 to exit): 5
Enter a number (0 to exit): 3
Enter a number (0 to exit): -2
Enter a number (0 to exit): 0
Sum of entered numbers: 6.

```java
import java.util.Scanner;

public class SumNumbers {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int sum = 0;
        int input;

        do {
            System.out.print("Введите число (0 для выхода): ");
            input = scanner.nextInt();
            if (input != 0) {
                sum += input;
            }
        } while (input != 0);

        System.out.println("Сумма введённых чисел: " + sum);
        scanner.close();
    }
}
```

Expected Output:

Enter a number (0 to exit): 5
Enter a number (0 to exit): 3
Enter a number (0 to exit): -2
Enter a number (0 to exit): 0
Sum of entered numbers: 6.

Exception Handling in Loops

Exception Handling in Loops

Why use exceptions?

- Prevent program crashes on invalid input.
- Handle errors gracefully.

Key components:

- `try` block: code that might throw an exception.
- `catch` block: handles specific exceptions.

Common exception: `InputMismatchException` (invalid data type).

Example 1

Task:

Write a program that asks user to enter an integer and checks if the number is even. If the user enters text (non-integer), the program should catch the exception and output a message.

```
Enter an integer: hello
Invalid input. Please enter a number.
Enter an integer: 4
The number is even: true.
```

Example 1 – common solution

```
import java.util.Scanner;


public class Test {


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


        int s = scanner.nextInt();
        System.out.println("number is even: " + (s % 2 == 0));


        scanner.close();
    }
}
```

Exception Handling — Example 1

Single-attempt input with recovery:

```
import java.util.InputMismatchException;
import java.util.Scanner;
public class Test {
   public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        try {
            int s = scanner.nextInt();
            System.out.println("number is even: " + (s%2 == 0));
        } catch (InputMismatchException e) {
            System.out.println("Wrong input. Enter a valid integer.");
        }
        scanner.close();
    }
}
```

1.Valid input (6):

`int s = scanner.nextInt(6)` → 6.

Output: 123 OK.

2.Invalid input ("abc"):

`int s = scanner.nextInt("abc")` throws `InputMismatchException`

Catch block executes →

Output: `Wrong input. Please enter a valid integer.`

Example 1

Task:

Write a program that asks user to enter an integer and checks if the number is even. If the user enters text (non-integer), the program should catch the exception **and keep asking until a valid integer is entered.**

```
Enter an integer: hello
Invalid input. Please enter a number.
Enter an integer: 4
The number is even: true.
```

Exception Handling — Example 2

Robust input loop with try-catch**:**

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

class Test {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        boolean flag = false;
        int s = 0;

        do {
            try {
                System.out.println("Enter an integer:");
                s = scanner.nextInt();
                System.out.println("Number is even: " + (s % 2 == 0));
                flag = true;
            } catch (InputMismatchException e) {
                System.out.println("Wrong input. Enter a valid integer.");
                scanner.nextLine(); // clear the buffer from invalid input
            }
        } while (!flag);

        scanner.close();
    }
}
```