CS 104 - Object-Oriented Programming

Lecture 4 - Java Control Structures, and Loops

Department of Computer Science, UET Peshawar

---

1. Switch Statements

A switch statement in Java is used when you have multiple conditions and you want to execute one of the many code blocks based on the condition's value. It's an alternative to using multiple `if-else` statements.

Syntax:

```
switch (variable) {
    case value1:
        // Code block for value1
        break;
    case value2:
        // Code block for value2
        break;
    // You can have any number of case statements
    default:
        // Code block for the default case
}
```

When to Use:

- Use `switch` when comparing a variable against a fixed set of values.
- Use `if-else` when handling complex conditions or ranges.

Example - Day of the Week Demo:

```
import java.util.Scanner;

public class DayOfWeek {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a number (1-7) to display the day of the
week: ");
        int day = scanner.nextInt();

        switch (day) {
            case 1:
                System.out.println("Monday");
                break;
            case 2:
```

```
                System.out.println("Tuesday");
                break;
            case 3:
                System.out.println("Wednesday");
                break;
            case 4:
                System.out.println("Thursday");
                break;
            case 5:
                System.out.println("Friday");
                break;
            case 6:
                System.out.println("Saturday");
                break;
            case 7:
                System.out.println("Sunday");
                break;
            default:
                System.out.println("Invalid day number");
        }
    }
}
```

---

2. Loops - `for` Loop

A `for` loop is used when you know how many times you want to iterate. It's composed of three parts: initialization, condition, and update.

Syntax:

```
for (initialization; condition; update) {
    // Code to be executed
}
```

Example - Capital and Interest Rate Calculation: Let's compute how capital grows with a fixed interest rate over a number of years using a `for` loop.

```
public class InterestCalculator {
    public static void main(String[] args) {
        double principal = 10000;  // Initial amount
        double rate = 0.05;        // Interest rate

        for (int year = 1; year <= 5; year++) {
            double interest = principal * rate;
            principal += interest;
            System.out.println("Year " + year + ": $" + principal);
        }
    }
}
```

Iteration Details:

- **First Iteration:** Initial capital = 10,000, interest added = 500 (5% of 10,000), new capital = 10,500.
- **Second Iteration:** Capital grows to 11,025, and so on.

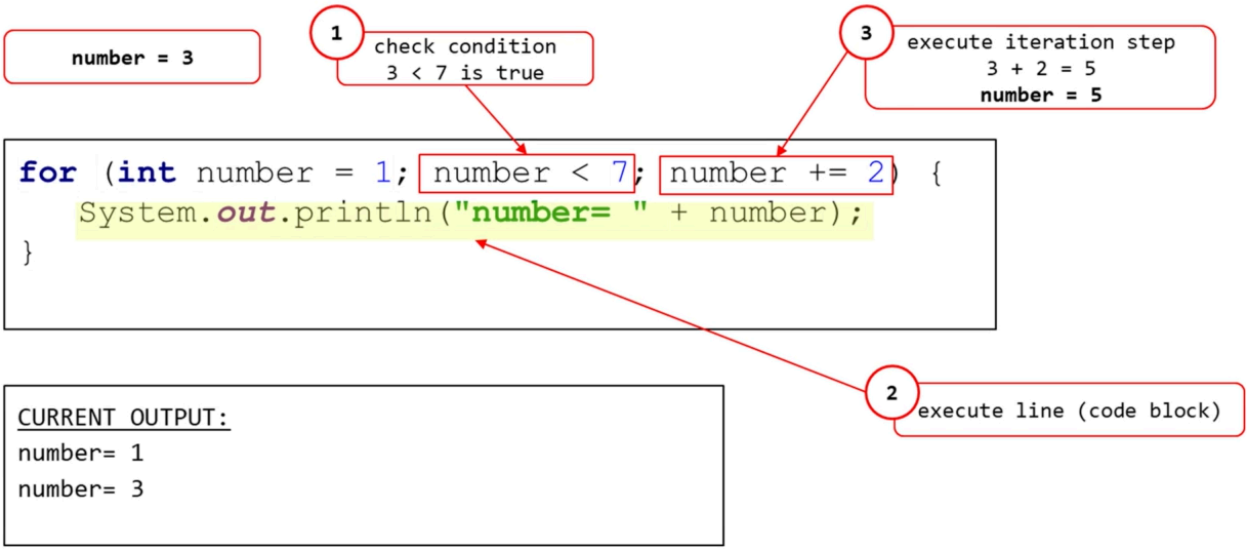

Figure 1

- **2nd iteration of the for loop**

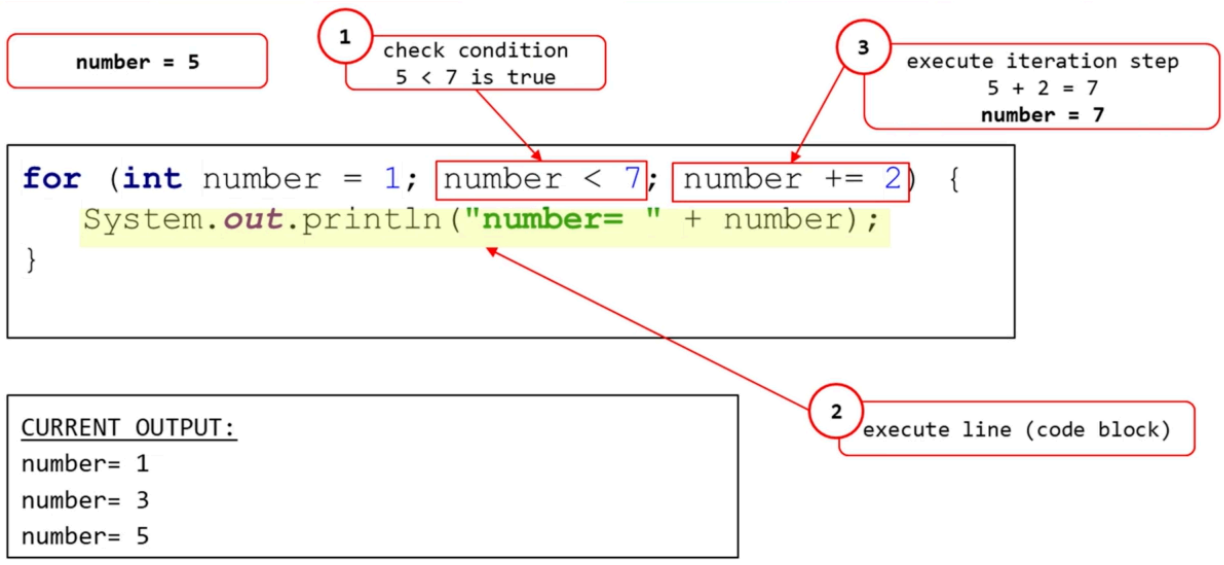

Figure 2

- **3rd iteration of the for loop**

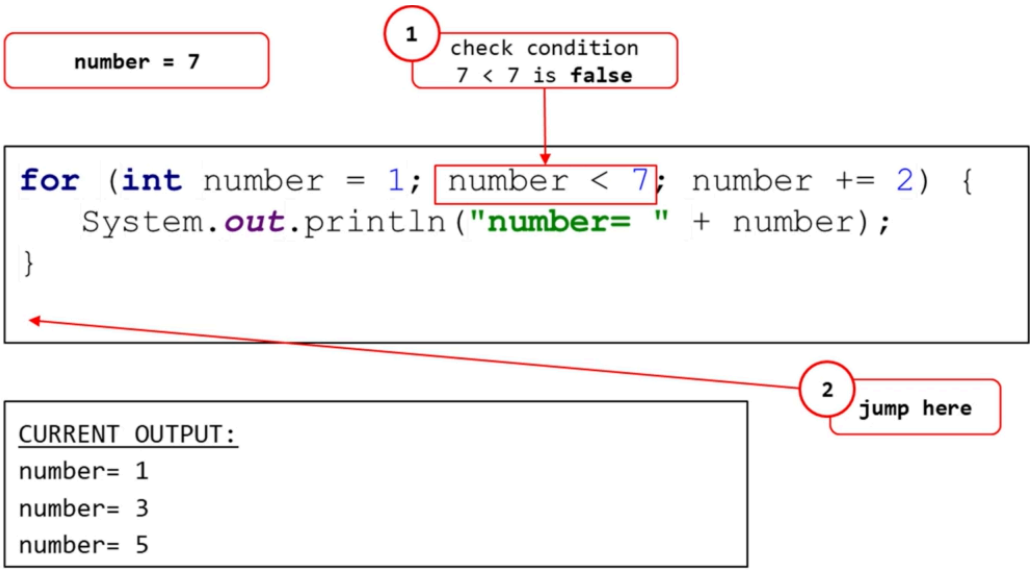

Figure 3

---

3. `while` and `do-while` Loops

The `while` loop is useful when the number of iterations is unknown, and you continue looping based on a condition.

`while` Loop Syntax:

```
while (condition) {
    // Code to be executed
}
```

`do-while` Loop Syntax:

Unlike `while`, a `do-while` loop guarantees at least one execution before checking the condition.

```
do {
    // Code to be executed
} while (condition);
```

Example - Sum of Digits Problem:

```
import java.util.Scanner;

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

```
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a number: ");
        int number = scanner.nextInt();
        int sum = 0;

        while (number > 0) {
            sum += number % 10;  // Add the last digit to sum
            number /= 10;        // Remove the last digit
        }

        System.out.println("Sum of digits: " + sum);
    }
}
```

---

4. User Input - `Scanner` Class

The `Scanner` class in Java is used to get user input. It provides methods like `nextInt()`, `nextLine()`, `hasNextInt()` to handle different types of input.

Example - Taking Name and Year of Birth:

```
import java.util.Scanner;

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

        System.out.println("Enter your name: ");
        String name = scanner.nextLine();

        System.out.println("Enter your year of birth: ");
        if (scanner.hasNextInt()) {
            int yearOfBirth = scanner.nextInt();
            System.out.println("Hello " + name + ", you are " + (2024 -
yearOfBirth) + " years old.");
        } else {
            System.out.println("Invalid year of birth.");
        }
    }
}
```

3. While and do While Loop

The **do-while loop** is similar to the `while` loop, but it **executes the code block at least once** because the condition is checked **after** each iteration. This loop is useful when the code inside the loop should run at least once, even if the condition is false.

Syntax:

```
do {
```

```
    // code to be executed
} while (condition);
```

Example: Do-While Loop:

```
int count = 5;
do {
    System.out.println("Count: " + count);
    count--;
} while (count > 0);
```

When to Use While and Do-While Loops

- **While loop**: Use when the condition needs to be checked **before** the first iteration. It's ideal when the number of iterations is not predetermined but can vary based on a condition that might be known dynamically during program execution.
- **Do-While loop**: Use when you want the loop to execute **at least once**, regardless of the condition's initial value. This is useful when you need user input before checking the condition or when a specific action must occur once.

Problem: Digit Sum Using the While Loop

One common problem to solve using loops is the **digit sum problem**, where the goal is to find the sum of all digits in a given number. This problem is well-suited for a `while` loop because we don't know in advance how many digits the number contains, but we continue summing digits until none are left.

Steps to Solve:

1. Initialize a variable to store the sum of digits.
2. Use the `while` loop to extract each digit by repeatedly dividing the number by 10.
3. Add each extracted digit to the sum.
4. Continue the loop until the number becomes zero.

Example: Sum of Digits:

```
import java.util.Scanner;

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

        System.out.print("Enter a number: ");
        int number = scanner.nextInt();
        int sum = 0;
```

```
        // Use while loop to calculate the sum of digits
        while (number > 0) {
            int digit = number % 10;  // Extract the last digit
            sum += digit;  // Add it to the sum
            number /= 10;  // Remove the last digit from the number
        }

        System.out.println("Sum of digits: " + sum);

        scanner.close();
    }
}
```

Explanation:

- The loop continues to extract digits from the number until the number becomes zero.
- `number % 10` gives the last digit of the number, and `number /= 10` removes the last digit, preparing for the next iteration.

Best Practices for Using Loops

1. **Ensure the condition eventually becomes false** to avoid infinite loops. Always make sure the loop variable is updated correctly within the loop.
2. **Minimize code inside the loop**: The code should only contain what's necessary for each iteration, making the loop efficient.
3. **Use meaningful conditions**: Ensure the condition you're checking reflects the loop's purpose. Clear conditions make the loop easier to understand.
4. **Limit use of do-while**: It can sometimes lead to confusion if used unnecessarily. Ensure it's the best choice when opting for it.

4. User Input

User input is an essential part of interactive programs where the user provides data that the program processes. In Java, user input can be handled using the **Scanner class**, which provides methods for reading input from the console.

Scanner Class

The `Scanner` class in Java is part of the `java.util` package. It is used to read various types of input like integers, strings, and other primitive data types.

How to Use the Scanner Class:

1. Import the Scanner class using:

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

2. Create an instance of Scanner:

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

nextInt() and nextLine()

- **nextInt()**: This method is used to read an integer input from the user. It reads the **next token of the input as an int**.

  Example:

```
System.out.print("Enter your age: ");
int age = scanner.nextInt();
System.out.println("You are " + age + " years old.");
```

- **nextLine()**: This method reads a **line of input** as a String, which includes spaces until the user hits enter.

  Example:

```
System.out.print("Enter your full name: ");
String name = scanner.nextLine();
System.out.println("Hello, " + name);
```

Input Order: Getting Name and Year of Birth

When combining the use of `nextInt()` and `nextLine()`, be cautious of how the Scanner processes input. `nextInt()` does not consume the newline character `\n`, so if you call `nextLine()` after `nextInt()`, it might skip the input.

Example: Common Mistake:

```
System.out.print("Enter your year of birth: ");
int yearOfBirth = scanner.nextInt();  // Reads the int

System.out.print("Enter your name: ");
String name = scanner.nextLine();  // This may be skipped due to leftover
newline
```

Solution: Use `nextLine()` to clear the input buffer after `nextInt()`:

```
System.out.print("Enter your year of birth: ");
```

```
int yearOfBirth = scanner.nextInt();
scanner.nextLine();  // Consumes the leftover newline

System.out.print("Enter your name: ");
String name = scanner.nextLine();
```

Validation: Checking for Integer Input with `hasNextInt()`

To avoid errors when the user enters an invalid data type (e.g., entering text instead of a number), we can use the `hasNextInt()` method to **check if the input is an integer**.

- **hasNextInt()**: This method returns a **boolean**. If the next input is an integer, it returns `true`; otherwise, it returns `false`.

Example:

```
System.out.print("Enter your year of birth: ");
boolean hasInt = scanner.hasNextInt();
if (hasInt) {
    int yearOfBirth = scanner.nextInt();
    System.out.println("Your year of birth is: " + yearOfBirth);
} else {
    System.out.println("Invalid input. Please enter a number.");
}
```

Practical Example: Checking Year of Birth

It's a good programming practice to validate input data to ensure it meets certain criteria. For example, we may want to check if the year of birth falls within a **valid range** (e.g., no one is born in the future).

Example: Checking the Year of Birth:

```
import java.util.Scanner;

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

        System.out.print("Enter your year of birth: ");
        boolean hasInt = scanner.hasNextInt();

        if (hasInt) {
            int yearOfBirth = scanner.nextInt();
            int currentYear = 2024;
            if (yearOfBirth > 1900 && yearOfBirth <= currentYear) {
                int age = currentYear - yearOfBirth;
                System.out.println("You are " + age + " years old.");
```

```
            } else {
                System.out.println("Invalid year of birth.");
            }
        } else {
            System.out.println("Invalid input. Please enter a valid number.");
        }

        scanner.close();
    }
}
```

Best Practices for User Input

1. **Always validate user input**: Ensure that the input type matches the expected data type. Use methods like `hasNextInt()` for type validation.
2. **Clear the input buffer** after reading primitive types (like using `nextLine()` after `nextInt()` to consume the newline).
3. **Provide meaningful error messages**: If the input is invalid, guide the user to correct their input.
4. **Ensure logical validation**: For example, when asking for a year of birth, validate that the year is realistic (not in the future or too far in the past).