Sukkur Institute of Business Administration University
Department of Computer Science

Object Oriented Programming

BS – II (CS/SE/AI)
Spring 2025

Lab # 03: Basic Games Implementation

Instructor: Moona Solangi

Lab Report Rubrics (Add the points in each column, then add across the bottom row to find the total score)					**Total Marks**
S.No	**Criterion**	**0.5**	**0.25**	**0.125**	
1	Accuracy	☐ Desired output	☐ Minor mistakes	☐ Critical mistakes	
2	Timing	☐ Submitted within the given time	☐ 1 day late	☐ More than 1 day late	

Submission Profile

Name: Submission date (dd/mm/yy):
Enrollment ID: Receiving authority name and signature:
Comments:

____

Instructor Signature

Note: Submit this lab handout before the next lab with attached solved activities and exercises

Objectives

This lab will help students apply fundamental Java concepts by implementing interactive games. The following topics will be covered:

- ✅ Loops (for, while, do-while)
- ✅ Conditional Statements (if-else, switch)
- ✅ Arrays (int[], String[])
- ✅ Break and Continue Statements
- ✅ User Input Handling (Scanner)
- ✅ Basic Math Operations
- ✅ Console Output (System.out.println())

By completing this lab, students will:

- Develop logic-building skills using loops and conditionals.
- Understand how arrays store and manipulate data.
- Improve problem-solving abilities through interactive games.

#	Game Task	Concepts Used
1	• **Number Guessing Game** 🎯	• **Loops, If-Else, Scanner**
2	• **Simple Quiz Game** 📝	• **Arrays, Loops, If-Else**
3	• **Simple Dice Roll Game** 🎲	• **Loops, If-Else, Scanner**
4	• **Hangman** 🔠	• **Loops, Arrays, If-Else**
5	• **Shopping Cart Simulator** 🛒	• **Loops, Arrays, Scanner**
6	• **Speed Typing Test** ⌨	• **Timer, Loops, Scanner**
7	• **Even & Odd Counter** 🔢	• **Arrays, Loops, Modulus**
8	• **Find the Missing Number** ❓	• **Arrays, If-Else**

📝 Lab Tasks

Each task provides **a description, hints, and expected output**.

🔷 Task 1: Number Guessing Game 🎯

The program will ask the user to guess a **fixed number (50).** The user will keep guessing until they find the correct number.

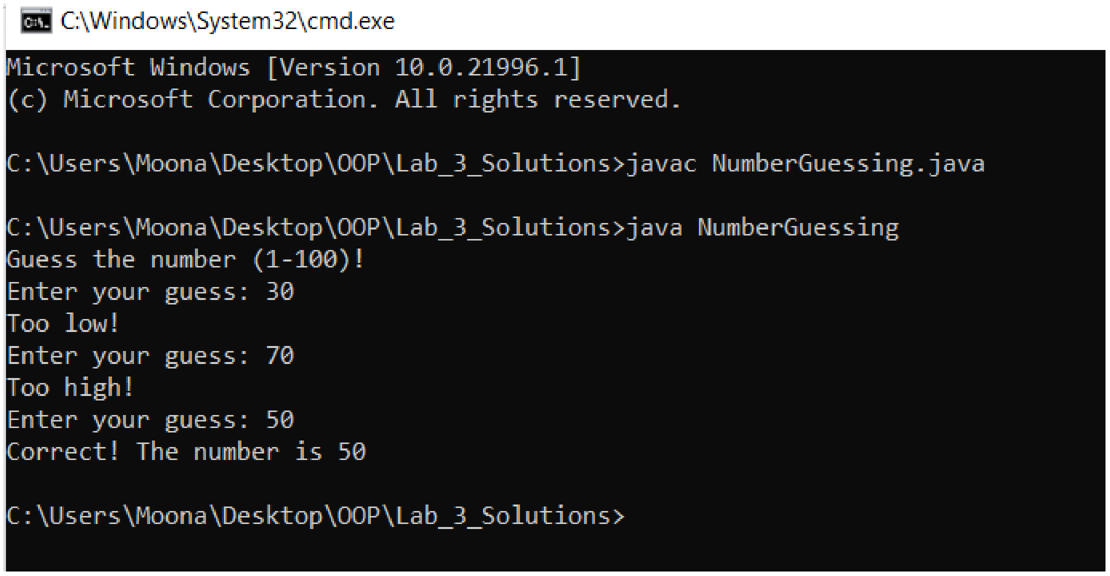
How It Works:

1. The program **stores a fixed number (50)** as the correct answer.
2. The user is asked to **guess a number.**
3. After each guess, the program gives a **hint:**
    - If the guess is **too high,** it prints **"Too high!"**
    - If the guess is **too low,** it prints **"Too low!"**
4. The user can **keep guessing** until they get the right number.
5. When the user guesses correctly, the program **congratulates them and ends the game.**

Hints to Solve the Task:

- Use a **while loop** so the user can guess multiple times.
- Use **if-else conditions** to check if the guess is higher or lower than the correct number.
- Use **break** to stop the loop when the user guesses the correct number.

Expected Output:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.21996.1]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>javac NumberGuessing.java

C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>java NumberGuessing
Guess the number (1-100)!
Enter your guess: 30
Too low!
Enter your guess: 70
Too high!
Enter your guess: 50
Correct! The number is 50

C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>
```

🔷 Task 2: Simple Quiz Game 📝

The program will present the user with **5 multiple-choice questions.** The user will answer each question, and the program will check if the **answer is correct or incorrect**. At the end, the program will display the **final score.**

How It Works:

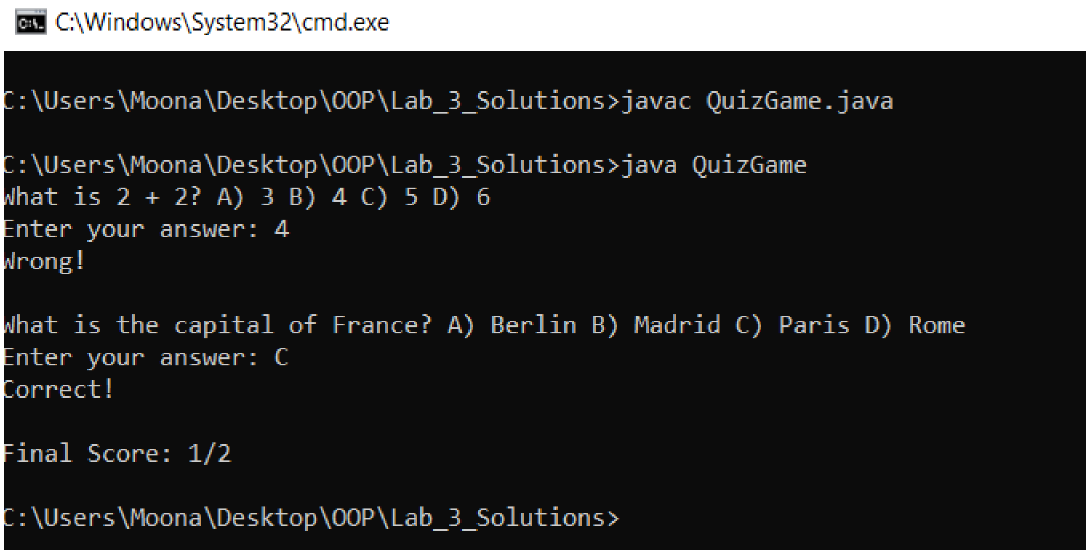
1. The program **stores 5 questions and their correct answers** in arrays.

2. The user is **shown one question at a time,** along with **four answer choices (A, B, C, D).**
3. The user **inputs their answer (A, B, C, or D).**
4. The program **checks the answer:**
    - If correct, it displays **"Correct!"** and increases the score.
    - If incorrect, it displays **"Wrong!".**
5. After **all 5 questions,** the program shows the **final score.**

Hints to Solve the Task:

- Use **arrays** to store questions and answers.
- Use a **for loop** to ask all 5 questions one by one.
- Use **if-else** conditions to check if the answer is correct or incorrect.

Expected Output:



```
C:\Windows\System32\cmd.exe

C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>javac QuizGame.java

C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>java QuizGame
What is 2 + 2? A) 3 B) 4 C) 5 D) 6
Enter your answer: 4
Wrong!

What is the capital of France? A) Berlin B) Madrid C) Paris D) Rome
Enter your answer: C
Correct!

Final Score: 1/2

C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>
```

🔷 Task 3: Simple Dice Roll Game 🎲 (User Chooses a Number)

The program simulates a **dice game** where the user chooses a number, and the computer has a **fixed number (e.g., 3).** The one with the **higher number wins.**

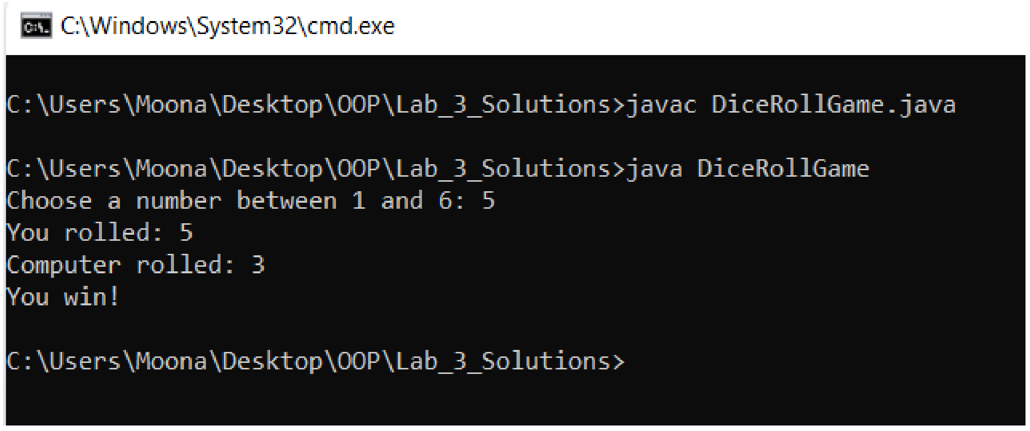
How It Works:

1. The user **chooses a number between 1 and 6** (like rolling a dice).
2. The computer **always picks the fixed number 3.**
3. The program **compares both numbers:**
    - If the user's number is **higher, they win.**
    - If the user's number is **lower,** the **computer wins.**
    - If both numbers are **equal**, **it's a tie.**
4. The user can **play multiple rounds** until they decide to stop.

Hints to Solve the Task:

- Use **if-else** conditions to compare the numbers and decide the winner.
- Use a **while loop** if the user wants to play multiple rounds.

Expected Output:


C:\Windows\System32\cmd.exe

```
C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>javac DiceRollGame.java

C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>java DiceRollGame
Choose a number between 1 and 6: 5
You rolled: 5
Computer rolled: 3
You win!

C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>
```

🔷 Task 4: Hangman (Word Guessing Game) 🔠

The program will simulate a **simple Hangman game** where the user tries to guess a **fixed word (apple)** letter by letter.
The user has a **limited number of attempts (lives)** to guess the correct letters before losing the game.

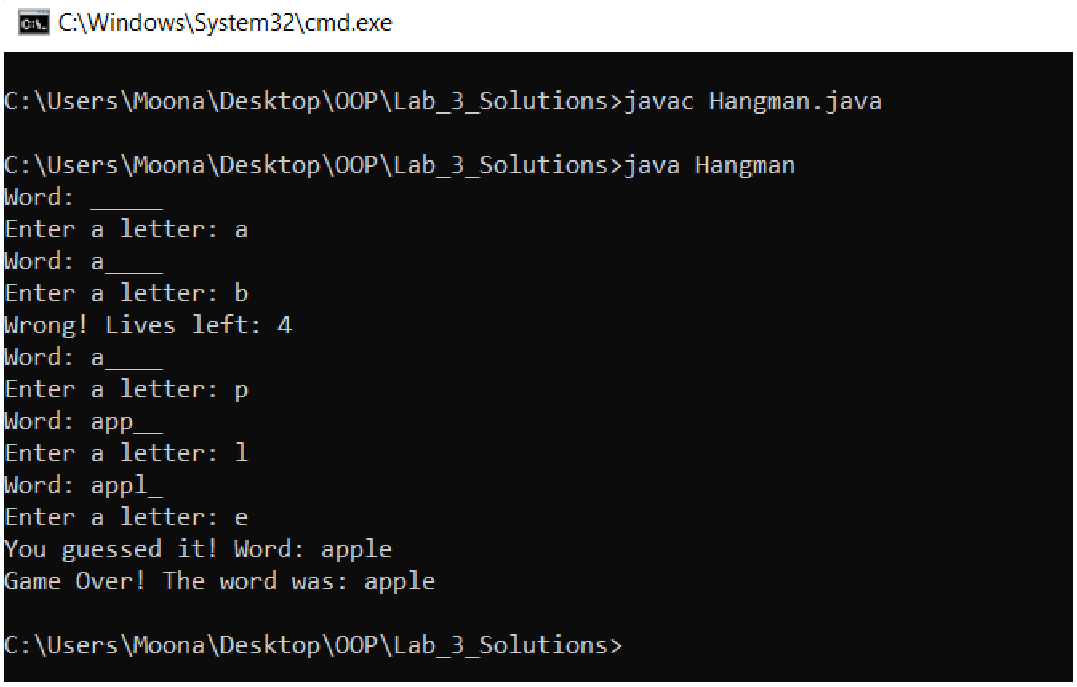
How It Works:

1. The program stores a **fixed word (apple).**
2. The word is displayed as **underscores (_ _ _ _)** to represent **hidden letters.**
3. The user **guesses one letter at a time.**
4. If the guessed **letter is in the word**, it is **revealed** in its correct position.
5. If the guessed **letter is not in the word**, the user **loses a life.**
6. The game continues until:
    - The user **correctly guesses all the letters (win).**
    - The user **runs out of lives (lose).**

Hints to Solve the Task:

- Use a **char array** to store the guessed letters.
- Use a **while loop** to allow multiple guesses.
- Use **if-else** conditions to check if the guessed letter is in the word.
- Reduce the **lives** for incorrect guesses.

Expected Output:



```
C:\Windows\System32\cmd.exe

C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>javac Hangman.java

C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>java Hangman
Word: _____
Enter a letter: a
Word: a____
Enter a letter: b
Wrong! Lives left: 4
Word: a____
Enter a letter: p
Word: app__
Enter a letter: l
Word: appl_
Enter a letter: e
You guessed it! Word: apple
Game Over! The word was: apple

C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>
```

🔷 Task 5: Shopping Cart Simulator 🛒

The program simulates a **shopping cart** where the user can **add items** to their cart from a **menu**. The program calculates the **total price** and displays it when the user **checks out**.

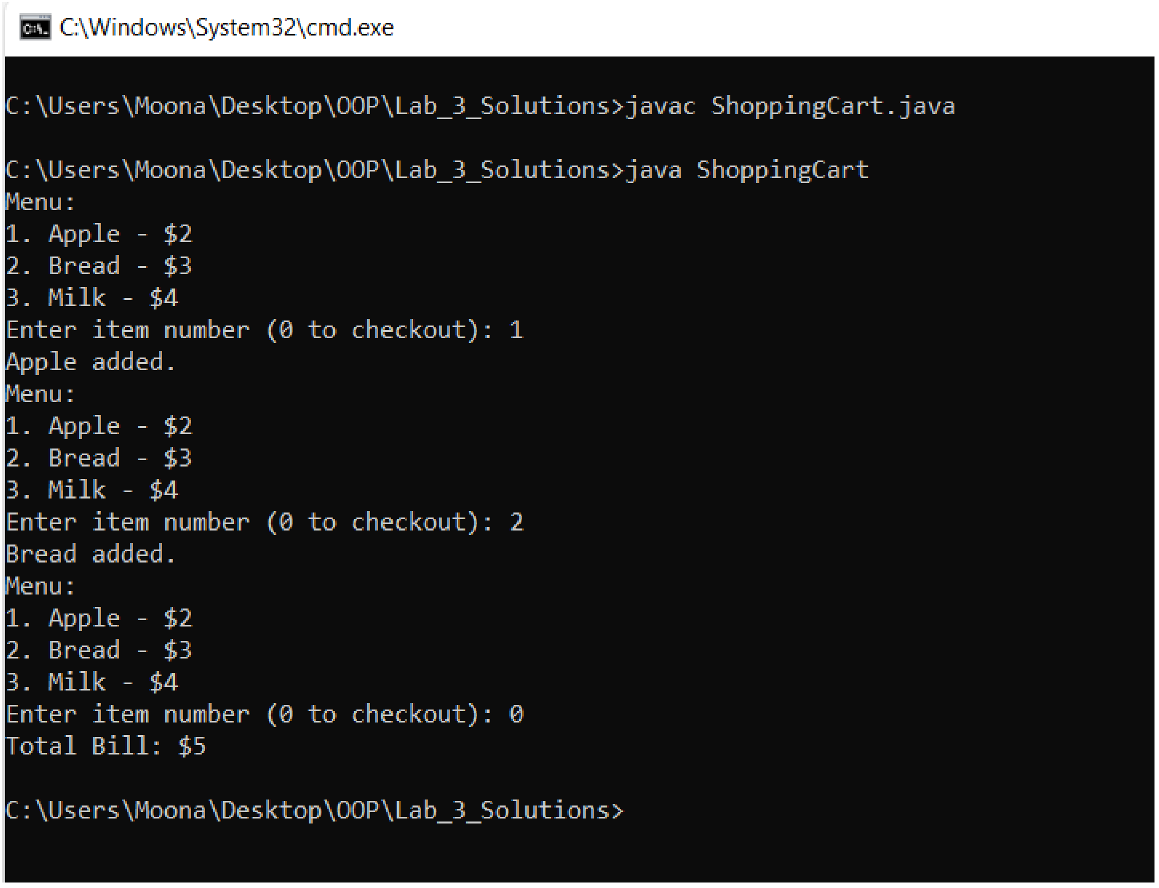
How It Works:

1. The program **displays a menu** of items with their **prices**.
2. The user selects an **item number** to **add it to the cart**.
3. The program keeps track of the **total price**.
4. The user can **continue selecting items** or **enter "0" to checkout**.
5. Once the user **checks out**, the program **displays the total bill** and ends.

Hints to Solve the Task:

- Use **arrays** to store **item names and prices**.
- Use a **while loop** to allow **multiple item selections**.
- Use an **if-else condition** to **handle user choices** and calculate the **total bill**.

Expected Output:



C:\Windows\System32\cmd.exe

```
C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>javac ShoppingCart.java

C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>java ShoppingCart
Menu:
1. Apple - $2
2. Bread - $3
3. Milk - $4
Enter item number (0 to checkout): 1
Apple added.
Menu:
1. Apple - $2
2. Bread - $3
3. Milk - $4
Enter item number (0 to checkout): 2
Bread added.
Menu:
1. Apple - $2
2. Bread - $3
3. Milk - $4
Enter item number (0 to checkout): 0
Total Bill: $5

C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>
```

🔷 Task 6: Speed Typing Test ⌨

The program will **test the user's typing speed** by displaying a **fixed sentence** that the user must **type exactly**. The program will then calculate and display **the time taken** to type the sentence.

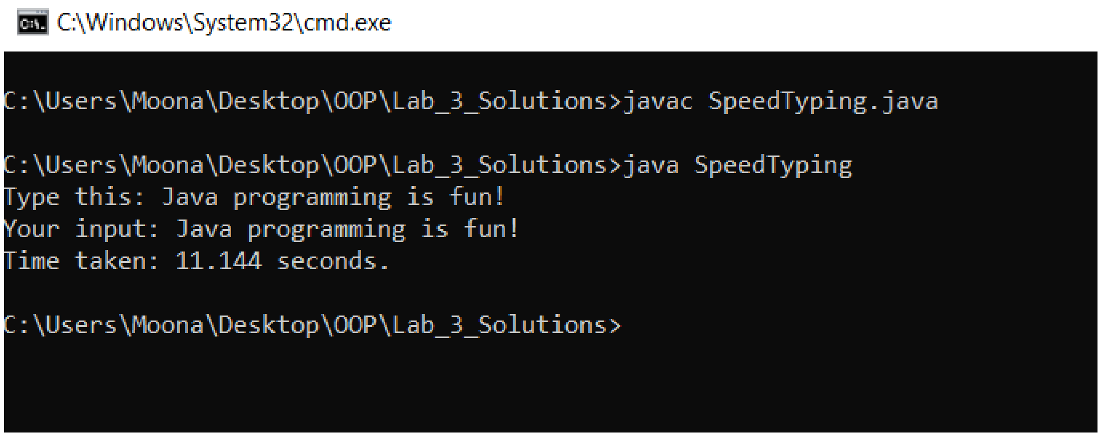
How It Works:

1. The program **displays a fixed sentence** that the user needs to type.
2. The user **types the sentence** exactly as shown.
3. The program **records the start time** before the user starts typing.
4. Once the user **presses Enter**, the program **records the end time**.
5. The program **calculates and displays the total time taken**.

Hints to Solve the Task:

- Use System.currentTimeMillis() to **record the start and end time**.
- Use Scanner.nextLine() to **take user input** for the sentence.
- Use **basic subtraction** to find the total time taken.
  (timeTaken = (endTime - startTime) / 1000.0)

Expected Output:



C:\Windows\System32\cmd.exe

```
C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>javac SpeedTyping.java

C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>java SpeedTyping
Type this: Java programming is fun!
Your input: Java programming is fun!
Time taken: 11.144 seconds.

C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>
```

🔷 Task 7: Even & Odd Counter Game 🔢

The program will take **5 numbers** as input from the user and count how many of them are **even** and how many are **odd**.

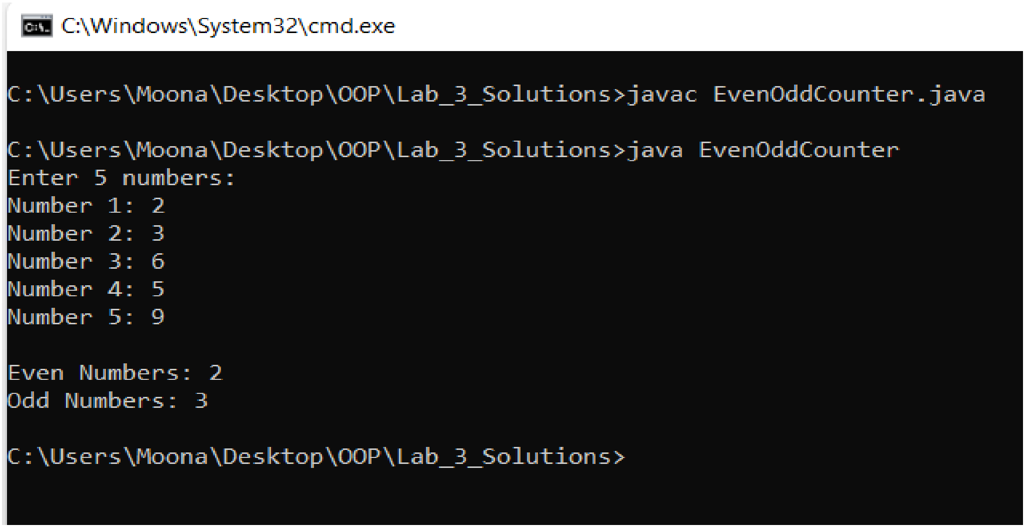
How It Works:

1. The user **enters 5 numbers**.
2. The program **stores these numbers in an array**.
3. It **checks each number** using the modulus operator (%):
    - If the number **divides evenly by 2 (num % 2 == 0)**, it is **even**.
    - Otherwise, it is **odd**.
4. The program **counts and displays** the number of **even** and **odd** numbers.

Hints to Solve the Task:

- Use an **array** to store the **5 numbers**.
- Use a **for loop** to **iterate through the array** and check each number.
- Use the **modulus operator (%)** to check if a number is **even** or **odd**.

Expected Output:



C:\Windows\System32\cmd.exe

```
C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>javac EvenOddCounter.java

C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>java EvenOddCounter
Enter 5 numbers:
Number 1: 2
Number 2: 3
Number 3: 6
Number 4: 5
Number 5: 9

Even Numbers: 2
Odd Numbers: 3

C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>
```

🔷 Task 8: Find the Missing Number Game 🔢❓

The program stores a **sequence of numbers with one missing number**. The user must guess the **missing number** to complete the sequence.

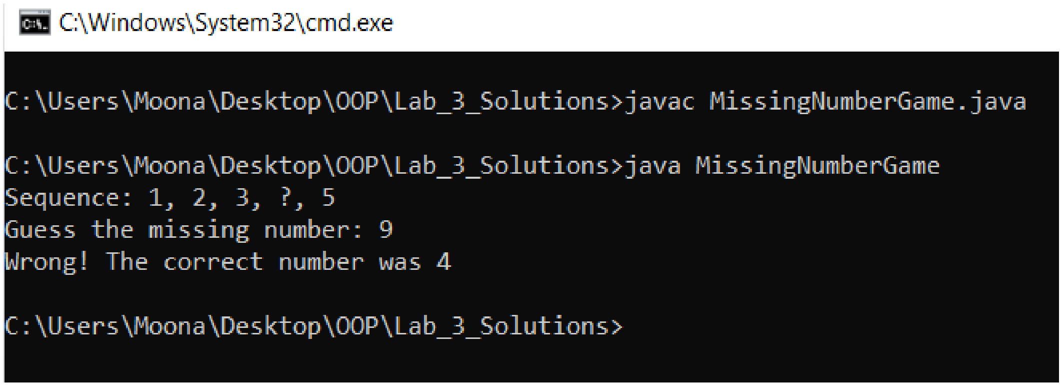
How It Works:

1. The program **displays a sequence of numbers**, but **one number is missing**.
2. The user is asked to **guess the missing number**.
3. The program **checks if the guessed number is correct**.
    - If the guess is **correct**, it prints **"Correct! The missing number was X."**
    - If the guess is **incorrect**, it prints **"Wrong! The correct number was X."**

Hints to Solve the Task:

- Use an **array** to store the sequence with **one missing value**.
- Use a **loop** to check **where the missing number is**.
- Use an **if-else condition** to verify the user's guess.

Expected Output:



```
C:\Windows\System32\cmd.exe

C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>javac MissingNumberGame.java

C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>java MissingNumberGame
Sequence: 1, 2, 3, ?, 5
Guess the missing number: 9
Wrong! The correct number was 4

C:\Users\Moona\Desktop\OOP\Lab_3_Solutions>
```

Wish You Best of Luck

😊