Java Programming Lab

1. Introduction to Java and Basic Input/Output

Concept: Java uses `System.out.println()` to output data to the console, and we can use the `Scanner` class to take input from users.

Explanation:

- `System.out.println()` prints text to the console.
- `Scanner` is a class that helps us read user input.

Example Code:

```
import java.util.Scanner;  // Import the Scanner class

public class HelloWorld {
    public static void main(String[] args) {
        // Output a welcome message
        System.out.println("Welcome to the Java Lab!");

        // Create a Scanner object
        Scanner scanner = new Scanner(System.in);

        // Get user input
        System.out.print("What's your name? ");  // Prompt the user
        String name = scanner.nextLine();  // Read the input
        System.out.println("Hello, " + name + "!");  // Greet the user
    }
}
```

Key Points:

- We use `import java.util.Scanner;` to access the Scanner class.
- `nextLine()` reads the entire line of input as a string.

2. Control Structures (if, else, else if)

Concept: Control structures allow the program to make decisions based on conditions.

Explanation:

- `if`, `else if`, and `else` are used to execute different blocks of code based on certain conditions.

Example Code:

```
import java.util.Scanner;

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

        System.out.print("How old are you? ");
        int age = scanner.nextInt();  // Read the integer input

        // Conditional statements
        if (age < 12) {
            System.out.println("You're a child!");
        } else if (age < 18) {
            System.out.println("You're a teenager!");
        } else {
            System.out.println("You're an adult!");
        }
    }
}
```

Key Points:

- `nextInt()` reads an integer from the user.
- The conditions check the age and print messages accordingly.

---

3. Operations and Logical Operators

Concept: We can perform arithmetic operations and combine conditions using logical operators.

Explanation:

- Arithmetic operators include `+`, `-`, `*`, and `/`.
- Logical operators like `&&` (AND) and `||` (OR) help combine multiple conditions.

Example Code:

```
public class Operations {
    public static void main(String[] args) {
        int x = 10;
        int y = 5;

        // Arithmetic operations
        int sum = x + y;  // Add x and y
        System.out.println("Sum: " + sum);

        // Logical operations
        if (x > 5 && y < 10) {
            System.out.println("Both conditions are true!");
        }
    }
}
```

Key Points:

- `sum` calculates the total of `x` and `y`.
- The `if` statement checks if both conditions are true using `&&`.

---

4. Arrays and Looping Through Arrays

Concept: Arrays are used to store multiple values in a single variable, and we can loop through them to access each value.

Explanation:

- An array is a collection of elements of the same type.
- We can use a `for` loop or `while` loop to iterate through the elements.

Example Code:

```
public class FruitArray {
    public static void main(String[] args) {
        String[] fruits = {"apple", "banana", "cherry"};  // Declare and initialize an array

        // Loop through the array using a for loop
        System.out.println("Fruits in the array:");
        for (String fruit : fruits) {
            System.out.println(fruit);  // Print each fruit
        }

        // Loop through the array using a while loop
        int index = 0;
        System.out.println("Fruits using while loop:");
        while (index < fruits.length) {
            System.out.println(fruits[index]);  // Print the fruit at the current index
            index++;
        }
    }
}
```

Key Points:

- `fruits` is an array that holds multiple fruit names.
- The `for` loop iterates through each fruit, while the `while` loop uses an index to access each element.

---

5. ArrayList and Looping Through ArrayLists

Concept: `ArrayList` is a part of the Java Collections Framework that allows dynamic arrays, meaning the size can change as we add or remove elements.

Explanation:

- `ArrayList` can hold objects, and we can easily add or remove elements.

Example Code:

```
 import java.util.ArrayList;

public class FruitList {
    public static void main(String[] args) {
        ArrayList<String> fruits = new ArrayList<>();  // Create an ArrayList
        fruits.add("apple");  // Add elements
        fruits.add("banana");
        fruits.add("cherry");

        // Loop through the ArrayList
        System.out.println("Fruits in the ArrayList:");
        for (String fruit : fruits) {
            System.out.println(fruit);
        }

        // Useful ArrayList methods
        fruits.add("orange");  // Add to the list
        System.out.println("After adding orange: " + fruits);

        fruits.remove("banana");  // Remove from the list
        System.out.println("After removing banana: " + fruits);

        System.out.println("Number of fruits: " + fruits.size());  // Size of the list
    }
}
```

Key Points:

- `ArrayList` automatically resizes itself as you add or remove elements.
- Methods like `add()`, `remove()`, and `size()` help manage the list.

---

6. HashMap and Looping Through HashMaps

Concept: `HashMap` is a data structure that stores key-value pairs.

Explanation:

- Each key is unique, and it maps to a specific value, allowing for efficient data retrieval.

Example Code:

```java
import java.util.HashMap;

public class StudentMap {
    public static void main(String[] args) {
        HashMap<String, String> student = new HashMap<>();  // Create a HashMap
        student.put("name", "Alice");  // Add key-value pairs
        student.put("age", "12");
        student.put("grade", "7th");

        // Accessing values
        System.out.println("Student Name: " + student.get("name"));
        System.out.println("Student Age: " + student.get("age"));

        // Looping through the HashMap
        System.out.println("Student Information:");
        for (String key : student.keySet()) {
            System.out.println(key + ": " + student.get(key));  // Print each key-value pair
        }
    }
}
```

Key Points:

- `put()` adds key-value pairs, and `get()` retrieves values by their keys.
- `keySet()` returns all keys in the map for iteration.

---

7. File Handling

Concept: We can read from and write to files using Java's I/O classes.

Explanation:

- Writing to a file saves data permanently, while reading allows us to access stored data.

Example Code:

```java
 import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class FileHandling {
    public static void main(String[] args) {
        // Writing to a file
        try {
            FileWriter writer = new FileWriter("data.txt");  // Create a FileWriter
            writer.write("Hello, world!\n");  // Write text to the file
            writer.write("Name: Alice\n");
            writer.write("Age: 12\n");
            writer.close();  // Close the writer
        } catch (IOException e) {
            System.out.println("An error occurred while writing to the file.");
            e.printStackTrace();
        }

        // Reading from a file
        try {
            File file = new File("data.txt");  // Create a File object
            Scanner reader = new Scanner(file);  // Create a Scanner to read the file
            System.out.println("File Content:");
            while (reader.hasNextLine()) {
                String line = reader.nextLine();  // Read each line
                System.out.println(line);  // Print the line
            }
            reader.close();  // Close the reader
        } catch (IOException e) {
            System.out.println("An error occurred while reading the file.");
            e.printStackTrace();
        }
    }
}
```

Key Points:

- `FileWriter` is used for writing data to a file, while `Scanner` reads from a file.
- Always handle exceptions to avoid crashes.

---

8. Simple Object-Oriented Programming (OOP)

Concept: Classes are blueprints for objects, allowing us to encapsulate data and methods.

Explanation:

- Objects are instances of classes, and they can have properties (attributes) and behaviors (methods).

Example Code:

```java
 // Dog class with attributes and methods
class Dog {
    String name;

    // Constructor to initialize the dog's name
    Dog(String name) {
        this.name = name;  // Set the name



    }

    void bark() {
        System.out.println(name + " says woof!");  // Dog barks
    }
}

public class OOPExample {
    public static void main(String[] args) {
        // Creating an object of Dog
        Dog myDog = new Dog("Buddy");  // Pass the name to the constructor
        myDog.bark();  // Call the bark method
    }
}

// Car class as another example
class Car {
    String make;
    String model;

    // Constructor to initialize make and model
    Car(String make, String model) {
        this.make = make;
        this.model = model;
    }

    void drive() {
        System.out.println("The " + make + " " + model + " is driving!");  // Driving message
    }
}

// Creating a Car object in a separate main method or class
Car myCar = new Car("Toyota", "Corolla");
myCar.drive();  // Call the drive method
```

Key Points:

- Classes can have constructors to initialize objects.
- Methods define the behaviors that objects can perform.

Conclusion

This detailed Java programming lab introduces key concepts in a clear and engaging way. Each section builds on the previous ones, providing a comprehensive understanding of Java programming. By working through these examples, they will gain a solid foundation in coding!