Handbook for Java

For High School Students

Prashanth Purushotham
ITPROACADEMY.CO.IN

About Me

Prashanth Purushotham

Head of IT Infrastructure & Automation
Support@ITProAcademy.co.in | **www.ITProAcademy.co.in**

With over **24 years of extensive experience in the IT industry**, I have dedicated my career to mastering and implementing innovative solutions across diverse domains, including **infrastructure, virtualization, converged systems, cloud technologies, cybersecurity**, and **automation**. As a **Microsoft Certified Trainer (MCT)** for the past **20 years**, I have been empowering professionals and students to harness cutting-edge technologies effectively.

Currently serving as the **Head of IT Infrastructure & Automation** for a UK-based IT company, I specialize in performance monitoring for both infrastructure and applications, business continuity planning (BCP), disaster recovery (DR), and solution design. My expertise spans a wide range of platforms, such as **Microsoft Windows, Linux, VMware, Hyper-V**, and **networking**. I excel in **migrating workloads to public, private, and hybrid cloud environments** while leveraging the latest advancements in **business process automation**.

I have obtained numerous certifications across multiple domains, including **Azure Solutions Architect Expert**, **Azure AI Engineer Associate**, and **Cybersecurity Architect Expert**, among others. These accolades reflect my commitment to continuous learning and staying at the forefront of technology trends. I have also been recognized for my contributions to the **Dell EMC Proven Professional Program**, **Oracle Cloud Foundations**, and **ISC² certifications**.

Through ITProAcademy, my vision is to bridge the gap between academic learning and industry demands by equipping engineering students with the knowledge and skills needed to excel in the dynamic world of IT. This guide is part of that mission, offering a comprehensive learning resource for **engineering students** as part of the **Connect Program** conducted with colleges. It is my endeavor to inspire the next generation of IT professionals and empower them to thrive in an ever-evolving technological landscape.

Contents

1. Introduction to Java

Objective

Understand the basics of Java and its significance in programming.

Topics to Cover

1. **What is Java?**
    - Java is a high-level, platform-independent, object-oriented programming language.
    - It is widely used for developing various applications, including mobile apps, web applications, and games.
2. **Why Learn Java?**
    - Java is one of the most popular programming languages globally.
    - It is robust, secure, and used extensively in the industry.
3. **Java Development Environment:**
    - **Java Development Kit (JDK):** Includes tools required for developing and running Java applications.
    - **Integrated Development Environment (IDE):** Software like **BlueJ**, **Eclipse**, or **IntelliJ IDEA** that simplifies writing and debugging Java code.

Activity

Setting Up Java

1. **Installing JDK and BlueJ:**
    - Download and install the Java Development Kit (JDK) from the official Oracle website or OpenJDK.
    - Install BlueJ IDE, designed for beginners to practice Java programming.
2. **Write and Run Your First Program:**
    - Open BlueJ and create a new project.
    - Add a new class and name it HelloWorld.
    - Write the following code inside the class:

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

```
        System.out.println("Hello, World!");
    }
}
```

3. **Running the Program:**
    - Compile the program by clicking the "Compile" button in BlueJ.
    - Run the program by right-clicking the class and selecting void main(String[] args).

Expected Output:

Hello, World!

Learning Outcomes:

- Understand how to set up the Java programming environment.
- Gain hands-on experience with writing and executing your first Java program.
- Learn the structure of a basic Java program, including the main() method and the System.out.println() statement.

---

2. Basic Syntax and Data Types

Objective

Learn the structure of Java programs, key syntax rules, and how to handle basic input/output.

Topics to Cover

1. **Java Keywords and Identifiers:**
    - **Keywords:** Reserved words in Java like public, class, void, etc.
    - **Identifiers:** Names used for variables, methods, or classes (e.g., myVariable, main).
2. **Variables and Data Types:**
    - **Variables:** Containers for storing data.
    - **Primitive Data Types:**
        - int - Stores integers (e.g., 5, 100).
        - double - Stores decimals (e.g., 5.5, 3.14).
        - char - Stores single characters (e.g., 'A', '1').
        - boolean - Stores true or false.
    - **Non-Primitive Data Types:**
        - String - Stores sequences of characters (e.g., "Hello").
3. **Input/Output in Java:**
    - **Output:** Use System.out.println() to display messages.
    - **Input:** Use Scanner for reading input from the user.

Activity

Writing a Program for Personal Information

1. **Objective:**
    - Create a program that takes a user's name and age as input and displays them.
2. **Program Code:**

    import java.util.Scanner;

```
public class PersonalInfo {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    // Taking name input
    System.out.print("Enter your name: ");
    String name = sc.nextLine();

    // Taking age input
    System.out.print("Enter your age: ");
    int age = sc.nextInt();

    // Displaying the inputs
    System.out.println("Name: " + name + ", Age: " + age);
  }
}
```

3. **Steps to Execute:**
   - Open your IDE (e.g., BlueJ).
   - Create a new class named PersonalInfo.
   - Copy and paste the code above.
   - Compile and run the program.

Expected Input and Output:

Example Input:

Enter your name: Aryan

Enter your age: 15

Example Output:

Name: Aryan, Age: 15

Key Learning Points

- Understand how to declare and use variables of different data types.
- Practice taking input using the Scanner class.
- Learn to display formatted output using concatenation (+ operator).

3. Control Statements

Objective

Understand how to control the flow of a Java program using conditional and looping statements.

Topics to Cover

1. **Conditional Statements:**
    - **if Statement:** Executes a block of code if a condition is true.

      ```
      if (condition) {
        // Code to execute if condition is true
      }
      ```

    - **if-else Statement:** Executes one block of code if a condition is true and another if it is false.

      ```
      if (condition) {
        // Code if true
      } else {
        // Code if false
      }
      ```

    - **switch Statement:** Chooses between multiple options based on a variable's value.

      ```
      switch (value) {
        case 1:
          // Code for case 1
          break;
        case 2:
          // Code for case 2
          break;
        default:
          // Code if no cases match
      ```

```
}
```

2. **Looping Statements:**
   - **for Loop:** Executes a block of code a fixed number of times.

     ```
     for (initialization; condition; increment/decrement) {
       // Code to execute
     }
     ```

   - **while Loop:** Repeats a block of code as long as a condition is true.

     ```
     while (condition) {
       // Code to execute
     }
     ```

   - **do-while Loop:** Similar to while, but executes the code at least once.

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

Activity

Display Even Numbers Using a Loop

1. **Objective:**
   - Write a program that prints all even numbers from 1 to 20 using a for loop.
2. **Program Code:**

```
public class EvenNumbers {
  public static void main(String[] args) {
  // Loop through numbers 1 to 20
  for (int i = 1; i <= 20; i++) {
    // Check if the number is even
    if (i % 2 == 0) {
      System.out.println(i);
    }
  }
```

```
    }
}
```

3. **Steps to Execute:**
    - Open your IDE (e.g., BlueJ).
    - Create a new class named EvenNumbers.
    - Copy and paste the program code.
    - Compile and run the program.

Expected Output:

2

4

6

8

10

12

14

16

18

20

Extensions and Practice:

1. Modify the program to display all odd numbers from 1 to 20.
2. Write a program to calculate the sum of numbers from 1 to 50 using a while loop.
3. Use a switch statement to display the name of the day based on a number input (1 for Monday, 2 for Tuesday, etc.).

---

4. Arrays

Objective

Understand the concept of arrays, how to use them in Java programs, and perform basic operations on arrays.

Topics to Cover

1. **Introduction to Arrays:**
    - An array is a collection of variables of the same type stored at contiguous memory locations.
    - Arrays allow you to store multiple values under a single name and access them using an index.
2. **Declaring and Initializing Arrays:**
    - **Declaration:**

      int[] arrayName; // Declares an array
    - **Initialization:**

      arrayName = new int[5]; // Allocates memory for 5 integers
    - **Combined Declaration and Initialization:**

      int[] arrayName = {value1, value2, value3}; // Initializes with values
3. **Accessing Array Elements:**
    - Use the index (starting from 0) to access or modify elements.

      arrayName[0] = 10; // Assigns value to the first element

      System.out.println(arrayName[0]); // Prints the first element
4. **Simple Operations on Arrays:**
    - **Sum of Elements:** Add all elements in the array.
    - **Average of Elements:** Divide the sum by the total number of elements.
    - **Finding Largest or Smallest Element:** Iterate through the array and compare each value.

Activity

Find the Largest Number in an Array

1. **Objective:**

- Write a program to find the largest number in an array.

2. **Program Code:**

```
public class LargestInArray {
  public static void main(String[] args) {
    // Initialize an array with values
    int[] numbers = {12, 45, 7, 34, 89};

    // Assume the first element is the largest
    int max = numbers[0];

    // Iterate through the array
    for (int num : numbers) {
      if (num > max) {  // Compare each number with max
        max = num;
      }
    }

    // Print the largest number
    System.out.println("Largest number is: " + max);
  }
}
```

3. **Steps to Execute:**
   - Open your IDE (e.g., BlueJ).
   - Create a new class named LargestInArray.
   - Copy and paste the program code.
   - Compile and run the program.

Expected Output:

Largest number is: 89

Extensions and Practice:

1. **Find the Smallest Number:** Modify the program to find the smallest number in the array.
2. **Sum and Average of Array Elements:** Write a program to calculate the sum and average of elements in an array.
3. **Sorting Arrays:** Learn how to sort an array using loops or the built-in Arrays.sort() method.

```
import java.util.Arrays;


public class SortArray {
  public static void main(String[] args) {
    int[] numbers = {12, 45, 7, 34, 89};
    Arrays.sort(numbers);
    System.out.println("Sorted Array: " + Arrays.toString(numbers));
  }
}
```

4. **Interactive Array Input:** Allow the user to input array elements via the console.

5. Functions and Methods

Objective

Learn to create and use methods in Java, understand return types, parameters, and the concept of static methods.

Topics to Cover

1. **What are Methods?**
    - Methods are blocks of code that perform a specific task.
    - They help avoid repetition and make programs modular and easier to debug.
2. **Creating and Calling Methods:**
    - **Syntax to Create a Method:**

      ```
      returnType methodName(parameters) {
        // Method body
        return value;  // Optional, depending on returnType
      }
      ```

    - **Calling a Method:**

      ```
      methodName(arguments);
      ```

3. **Return Types and Parameters:**
    - **Return Type:** Specifies the type of value the method returns. Use void if no value is returned.
    - **Parameters:** Variables passed to a method to provide input.

      ```
      public int addNumbers(int a, int b) {
        return a + b;
      }
      ```

4. **Static Methods:**
    - Methods declared as static belong to the class and can be called without creating an object.

      ```
      public static void greet() {
        System.out.println("Hello!");
      ```

```
}
```

Activity

Write a Program to Calculate the Factorial of a Number

1. **Objective:**
    - Create a method that calculates the factorial of a given number.
2. **Program Code:**

```
public class Factorial {
  // Method to calculate factorial
  public static int findFactorial(int n) {
    int result = 1; // Initialize result
    for (int i = 1; i <= n; i++) { // Loop through numbers from 1 to n
      result *= i; // Multiply each number with result
    }
    return result; // Return the final factorial value
  }

  public static void main(String[] args) {
    // Call the method and print the factorial of 5
    System.out.println("Factorial of 5: " + findFactorial(5));
  }
}
```

Steps to Execute:

1. Open your IDE (e.g., BlueJ).
2. Create a new class named Factorial.
3. Copy and paste the program code.
4. Compile and run the program.

Expected Output:

Factorial of 5: 120

Extensions and Practice:

1. **Interactive Input:** Modify the program to allow the user to input a number.

```
import java.util.Scanner;


public class Factorial {
  public static int findFactorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
      result *= i;
    }
    return result;
  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a number: ");
    int num = sc.nextInt();
    System.out.println("Factorial of " + num + ": " + findFactorial(num));
  }
}
```

2. **Overloading Methods:** Create multiple methods with the same name but different parameters. Example:

```
public static int findFactorial(int n) { ... }
public static long findFactorial(long n) { ... }
```

3. **Recursive Factorial Calculation:** Introduce recursion to solve the factorial problem.

```
public static int findFactorial(int n) {
  if (n == 0 || n == 1) {
    return 1;
  } else {
    return n * findFactorial(n - 1);
  }
}
```

6. Object-Oriented Programming (OOP) Basics

Objective

Learn the foundational concepts of Object-Oriented Programming (OOP) in Java, including classes, objects, constructors, and encapsulation.

Topics to Cover

1. **Classes and Objects:**
    - A **class** is a blueprint for creating objects. It defines properties (attributes) and methods (behaviors).
    - An **object** is an instance of a class.

      ClassName objectName = new ClassName();
2. **Constructors:**
    - Special methods used to initialize objects.
    - The **constructor name** must match the class name.
    - No return type.

      ```
      public class Student {
        public Student() {
          // Constructor logic
        }
      }
      ```
3. **Access Modifiers:**
    - Define the visibility of class members:
        - **public:** Accessible from anywhere.
        - **private:** Accessible only within the class.
        - **protected and default:** Introduced later.
4. **Encapsulation:**
    - Wrapping data (variables) and code (methods) into a single unit (class).
    - Achieved by:
        - Declaring class variables as private.

- Using public methods (getters and setters) to access and update them.

Activity

Create a Student Class

Objective:
Write a program to define a class Student with properties such as name, roll number, and grade. Use a constructor to initialize these properties and display the details using a method.

Code Example:

```
// Define the Student class
public class Student {
  // Properties of the class
  String name;
  int rollNo;
  char grade;

  // Constructor to initialize the Student object
  public Student(String name, int rollNo, char grade) {
    this.name = name;
    this.rollNo = rollNo;
    this.grade = grade;
  }

  // Method to display student details
  public void displayDetails() {
    System.out.println("Name: " + name);
    System.out.println("Roll No: " + rollNo);
    System.out.println("Grade: " + grade);
  }
```

```
  // Main method to test the Student class
  public static void main(String[] args) {
    // Create a Student object and initialize it
    Student student = new Student("Arya", 101, 'A');
    // Call the method to display student details
    student.displayDetails();
  }
}
```

Steps to Execute:

1. Open your IDE (e.g., BlueJ or IntelliJ IDEA).
2. Create a new class named Student.
3. Copy and paste the program code into the editor.
4. Compile and run the program.

Expected Output:

Name: Arya

Roll No: 101

Grade: A

Extensions and Practice:

1. **Encapsulation Enhancement:** Modify the program to use private access modifiers and provide getters and setters.

```
public class Student {
  private String name;
  private int rollNo;
  private char grade;

  public Student(String name, int rollNo, char grade) {
```

```
        this.name = name;
        this.rollNo = rollNo;
        this.grade = grade;
    }

    public String getName() {
        return name;
    }

    public int getRollNo() {
        return rollNo;
    }

    public char getGrade() {
        return grade;
    }

    public void displayDetails() {
        System.out.println("Name: " + getName());
        System.out.println("Roll No: " + getRollNo());
        System.out.println("Grade: " + getGrade());
    }

    public static void main(String[] args) {
        Student student = new Student("Arya", 101, 'A');
        student.displayDetails();
    }
}
```

2. **Adding More Functionality:**
    - Add methods like updateGrade(char newGrade) to update the student's grade.
3. **Practice Task:**
    - Create a Library class with properties like bookName, authorName, and bookID. Add methods to issue and return books.

---

7. Strings in Java

Objective

Learn how to work with strings in Java and understand key methods for string manipulation.

Topics to Cover

1. **String Class Basics:**
    - In Java, strings are objects of the String class.
    - Strings are immutable, meaning once created, their content cannot be changed.
2. **Common String Methods:**
    - length(): Returns the number of characters in the string.

      String str = "Hello";

      System.out.println(str.length()); // Output: 5
    - substring(startIndex, endIndex): Extracts a portion of the string.

      String str = "Welcome";

      System.out.println(str.substring(0, 4)); // Output: Welc
    - charAt(index): Returns the character at the specified index.

      String str = "Java";

      System.out.println(str.charAt(1)); // Output: a
    - toLowerCase() and toUpperCase(): Converts the string to lower or upper case.

      String str = "Hello";

      System.out.println(str.toLowerCase()); // Output: hello

      System.out.println(str.toUpperCase()); // Output: HELLO
3. **String Comparison:**
    - equals(): Checks if two strings have the same value.

      String str1 = "Java";

      String str2 = "Java";

```
System.out.println(str1.equals(str2)); // Output: true
```

- compareTo(): Compares two strings lexicographically.

```
String str1 = "Apple";

String str2 = "Banana";

System.out.println(str1.compareTo(str2)); // Output: -1 (Apple < Banana)
```

Activity

Check for Palindrome

Objective:
Write a program to determine whether a given string is a palindrome (reads the same forward and backward).

Code Example:

```
public class Palindrome {
  public static void main(String[] args) {
    String str = "madam"; // Input string
    String reverse = new StringBuilder(str).reverse().toString(); // Reverse the string

    // Check if the original string and reversed string are the same
    if (str.equals(reverse)) {
      System.out.println(str + " is a palindrome.");
    } else {
      System.out.println(str + " is not a palindrome.");
    }
  }
}
```

Steps to Execute:

1. Open your Java IDE (e.g., BlueJ, IntelliJ IDEA).
2. Create a new class named Palindrome.
3. Copy and paste the program code into the editor.

4. Compile and run the program.

Expected Output:

If the input is "madam":

madam is a palindrome.

For input like "hello":

hello is not a palindrome.

Extensions and Practice:

1. **Modify the Program:**
   - Accept user input for the string using Scanner.

```
import java.util.Scanner;


public class Palindrome {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a string: ");
    String str = sc.nextLine();
    String reverse = new StringBuilder(str).reverse().toString();

    if (str.equals(reverse)) {
      System.out.println(str + " is a palindrome.");
    } else {
      System.out.println(str + " is not a palindrome.");
    }
  }
}
```

2. **Practice Task:**

- Write a program to count the number of vowels in a given string.

3. **Challenge Task:**
   - Write a program to remove all spaces from a string.

Tips for Students:

- Pay attention to case sensitivity when comparing strings (use equalsIgnoreCase() for case-insensitive comparison).
- Experiment with other String methods such as replace(), indexOf(), and trim() to understand their utility.

---

8. Basic File Handling

Objective

Understand how to work with files in Java for reading and writing text data.

Topics to Cover

1. **What is File Handling?**
   File handling in Java allows programs to create, read, and write to files, enabling data persistence.
2. **Important Classes for File Handling:**
   - File: Represents file and directory pathnames.
   - FileWriter: Used to write character data to a file.
   - FileReader: Used to read character data from a file.
   - Scanner: Reads text data from files or input streams.
3. **Basic Methods to Know:**
   - Creating a file.
   - Writing to a file using FileWriter.
   - Reading from a file using Scanner.

Activity

Write to a File

Objective:
Create a file and write a message into it.

Code Example: Writing to a File

```
import java.io.FileWriter;
import java.io.IOException;


public class FileWriteExample {
  public static void main(String[] args) {
    try {
      // Create a FileWriter object to write to a file named "example.txt"
```

```
        FileWriter writer = new FileWriter("example.txt");

        // Write a message to the file
        writer.write("Hello, this is a Java file!");

        // Close the writer to save the file
        writer.close();

        System.out.println("Message written to file successfully.");
    } catch (IOException e) {
        System.out.println("An error occurred.");
    }
  }
}
```

Steps to Execute:

1. Open your Java IDE (e.g., BlueJ, IntelliJ IDEA).
2. Create a new class named FileWriteExample.
3. Copy and paste the code into the editor.
4. Run the program.

Expected Outcome:

- A file named **example.txt** will be created in the program's directory.
- The file will contain the message:

Hello, this is a Java file!

Extension Activity: Reading from a File

Objective:
Read and display the contents of the created file.

Code Example: Reading from a File

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class FileReadExample {
  public static void main(String[] args) {
    try {
      // Create a File object to represent the file
      File file = new File("example.txt");

      // Create a Scanner object to read from the file
      Scanner reader = new Scanner(file);

      System.out.println("Contents of the file:");

      // Read and display the contents line by line
      while (reader.hasNextLine()) {
        System.out.println(reader.nextLine());
      }

      reader.close(); // Close the scanner
    } catch (FileNotFoundException e) {
      System.out.println("File not found.");
    }
  }
}
```

Steps for the Reading Program:

1. Run the **writing program** first to create the file.
2. Create another class named FileReadExample for the reading program.
3. Copy and paste the code into the editor and run it.

Expected Outcome:

- The program will display:

Contents of the file:

Hello, this is a Java file!

Practice Tasks:

1. **Modify the Writing Program:**
    - Write multiple lines into the file using \n for line breaks.

      writer.write("Line 1: Java is fun!\n");

      writer.write("Line 2: File handling is useful.\n");
2. **Challenge: Write a Menu-Driven Program:**
    - Add options for creating, reading, and appending data to a file.
3. **Explore Advanced Features:**
    - Use BufferedReader and BufferedWriter for efficient file operations.

Tips for Students:

- Always close file streams using close() to avoid data loss.
- Use try-catch blocks to handle exceptions like IOException or FileNotFoundException.
- Ensure the file path is correct if the file is in a specific directory.

---

9. Java Best Practices

Objective

Learn and implement best practices to write clean, readable, and efficient Java code.

Topics to Cover

1. **Proper Indentation and Code Readability**
    - Consistent indentation is essential for making your code readable and easier to understand.
    - Use 4 spaces or a tab for each level of indentation.
    - Group related lines of code together using whitespace (blank lines) for separation.
    - Example of good indentation:

```
public class Example {
  public static void main(String[] args) {
    int sum = 0;
    for (int i = 0; i < 10; i++) {
      sum += i;
    }
    System.out.println("Sum: " + sum);
  }
}
```

2. **Meaningful Variable Names**
    - Always use descriptive and meaningful names for variables, methods, and classes.
    - Avoid single-letter variable names (except for loop counters), and use names that indicate the purpose of the variable.
    - Example of good variable names:

```
int totalMarks = 95;
double averageScore = 89.5;
```

    - Bad example:

```
int x = 95; // What does x represent?
```

3. **Adding Comments for Clarity**
   - Comments help other programmers (and yourself) understand the code's purpose, especially when you return to it later.
   - Use comments to explain complex or important logic, and use them sparingly to avoid clutter.
   - Java supports two types of comments:
     - **Single-line comment:** // This is a comment
     - **Multi-line comment:**
       /* This is a multi-line comment */
     - **Documentation comment:**
       /** This comment is for generating documentation */
   - Example:

```
// Loop through numbers 1 to 10 and print each number
for (int i = 1; i <= 10; i++) {
  System.out.println(i);
}
```

   - Documentation comments can also be used to describe methods:

```
/**
 * This method calculates the area of a rectangle.
 * @param length The length of the rectangle.
 * @param width The width of the rectangle.
 * @return The area of the rectangle.
 */
public double calculateArea(double length, double width) {
  return length * width;
}
```

4. **Avoiding Redundancy in Code**

- Don't repeat the same code in multiple places. Instead, create methods to handle common tasks and avoid code duplication.
- Use loops and methods to perform repetitive tasks.
- Example of avoiding redundancy by creating a method:

```
public class Calculator {
  // Method to add two numbers
  public static int add(int a, int b) {
    return a + b;
  }

  public static void main(String[] args) {
    // Calling the method
    int sum = add(5, 7);
    System.out.println("Sum: " + sum);
  }
}
```

- Instead of repeating the addition code, we reuse the add() method to perform the operation.

Best Practices Summary:

- **Consistency:** Ensure consistent indentation and formatting throughout your code to improve readability.
- **Descriptive Names:** Always name variables and methods according to what they represent or do.
- **Code Comments:** Add comments to explain the purpose of code sections, but avoid over-commenting simple or self-explanatory code.
- **Refactor Redundant Code:** Avoid writing the same logic repeatedly; use methods and loops to handle similar tasks.

Activity: Refactor and Comment a Program

1. Write a simple program that calculates the sum of the squares of numbers from 1 to 5.
2. Refactor the program by using a loop to calculate the sum of squares.
3. Add comments to explain the logic in your program.

Example:

```
public class SumOfSquares {
  public static void main(String[] args) {
    int sum = 0;

    // Loop through numbers 1 to 5
    for (int i = 1; i <= 5; i++) {
      sum += i * i; // Add the square of i to sum
    }

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

Practice Tasks:

1. **Indentation Challenge:**
   Given a poorly indented program, fix the indentation and make it more readable.
2. **Code Refactoring:**
   Take a repetitive block of code and refactor it into a method to reduce redundancy.
3. **Add Comments:**
   Write a program and add helpful comments explaining the logic of different sections of the code.

Tips for Students:

- Use an IDE like IntelliJ IDEA or Eclipse, which automatically formats your code for better readability.
- Practice writing code in a clean and consistent manner to develop habits that will help you in larger projects.
- Always test your code after making changes and adding comments to ensure everything works correctly.

---

10. Project: Mini Calculator

Objective:

Combine all the concepts learned so far to build a simple calculator application that performs basic arithmetic operations.

Project Guidelines

In this project, we will build a simple calculator that can:

- Perform addition, subtraction, multiplication, and division.
- Accept inputs using the Scanner class.
- Use a switch statement to perform the operations.
- Handle invalid inputs gracefully, including checking for division by zero.

Steps to Create the Mini Calculator

1. **Get Input from the User:**
    - Use the Scanner class to take two numbers and the desired operation from the user.
2. **Use a Switch Statement:**
    - A switch statement will allow you to choose between the four operations: addition (+), subtraction (-), multiplication (*), and division (/).
3. **Handle Invalid Input:**
    - Ensure that if the user enters a wrong operation or tries to divide by zero, the program will print an appropriate error message.

Example Program:

```
import java.util.Scanner;


public class Calculator {
  public static void main(String[] args) {
    // Create a Scanner object to take user input
    Scanner sc = new Scanner(System.in);


    // Take input for the first number
```

```
System.out.print("Enter first number: ");
double num1 = sc.nextDouble();

// Take input for the second number
System.out.print("Enter second number: ");
double num2 = sc.nextDouble();

// Take input for the operation
System.out.print("Enter operation (+, -, *, /): ");
char op = sc.next().charAt(0);

// Perform the operation based on user input
switch (op) {
  case '+':
    System.out.println("Result: " + (num1 + num2));  // Add numbers
    break;
  case '-':
    System.out.println("Result: " + (num1 - num2));  // Subtract numbers
    break;
  case '*':
    System.out.println("Result: " + (num1 * num2));  // Multiply numbers
    break;
  case '/':
    // Check if the denominator is zero to avoid division by zero
    if (num2 != 0) {
      System.out.println("Result: " + (num1 / num2));  // Divide numbers
    } else {
      System.out.println("Cannot divide by zero.");  // Handle division by zero
```

```
            }
            break;
        default:
            // Handle invalid operator input
            System.out.println("Invalid operation.");
    }

    // Close the scanner
    sc.close();
  }
}
```

Explanation of the Program

- **Scanner class:**
  We use the Scanner class to accept inputs from the user for the numbers and the operation to perform.
- **Switch statement:**
  Based on the operation input (+, -, *, /), the switch statement selects the corresponding case and performs the desired arithmetic operation.
- **Division by Zero:**
  Before performing division, we check if the second number (num2) is zero. If it is, we display an error message instead of performing the division.
- **Invalid Input Handling:**
  If the user enters an operator that is not +, -, *, or /, the program will output "Invalid operation."

Activity

1. **Run the Program:**
   - Compile and run the provided program. Test different inputs, such as valid and invalid operations, and try dividing by zero.
2. **Modify the Program:**
   - Extend the calculator to handle more operations (e.g., modulus %, exponentiation).

- Add input validation to check if the user enters a valid number (i.e., avoid errors when non-numeric values are input).

Practice Tasks:

1. **Add More Features:**
   - Implement a loop to keep the calculator running until the user chooses to exit.
   - Implement a function to calculate the square root of a number using Math.sqrt().
2. **Error Handling:**
   - Add error handling for cases where the user inputs something other than a number or operator.
3. **Additional Functionality:**
   - Allow the user to perform multiple operations in one calculation (e.g., 2 + 3 * 5).

Tips for Students:

- **Test your code:**
  After implementing new features or making changes, always test your program with different inputs to ensure it behaves as expected.
- **Use meaningful variable names:**
  Name your variables, like num1, num2, and op, in a way that makes it clear what they represent in the program.
- **Comment your code:**
  Comment each part of the program to explain what it does, especially if the logic is complex.

Additional Resources

1. **Online Learning Platforms:**
    - **Codecademy**: Offers interactive Java lessons to help you understand basic and advanced concepts.
    - **W3Schools Java Tutorial**: Provides simple explanations and examples on Java basics, methods, arrays, and more.
    - **GeeksforGeeks Java Programming**: A comprehensive collection of tutorials and examples on Java topics, including advanced concepts.
2. **Books:**
    - **Head First Java** by Kathy Sierra and Bert Bates: A beginner-friendly book with a hands-on approach to learning Java.
    - **Java: The Complete Reference** by Herbert Schildt: A more detailed guide suitable for understanding Java in-depth.
3. **Online Java Practice:**
    - **HackerRank** Java Practice: A platform offering coding challenges to enhance problem-solving skills.
    - **LeetCode** Java Practice: A great place to practice coding problems in Java and improve your algorithmic skills.

Worksheets

Worksheet 1: Basic Syntax and Data Types

1. **Question 1:**
   Write a Java program to input two numbers and display their sum, difference, and product.
2. **Question 2:**
   Create a program that asks the user to input their age and displays whether they are a minor (age < 18) or an adult (age >= 18).
3. **Question 3:**
   Write a program to calculate the average of three numbers entered by the user.

Worksheet 2: Control Statements

1. **Question 1:**
   Write a program to print all prime numbers from 1 to 100.

2. **Question 2:**
   Create a program that takes a number as input and displays whether the number is even or odd.

3. **Question 3:**
   Write a program using a switch statement that asks the user to choose a day (1-7) and prints the corresponding weekday name.

Worksheet 3: Arrays and Strings

1. **Question 1:**
   Write a program that sorts an array of 10 integers in ascending order.

2. **Question 2:**
   Write a program that finds the smallest number in an array of 5 integers.

3. **Question 3:**
   Write a program to count the number of vowels in a string.

Worksheet 4: Object-Oriented Programming (OOP)

1. **Question 1:**
   Create a Book class with the properties title, author, and price. Add a method displayBookInfo() to display the book details.

2. **Question 2:**
   Create a Student class with the properties name, rollNo, and marks. Write a method to calculate the grade based on marks.

3. **Question 3:**
   Create a class called Rectangle with the properties length and width. Write methods to calculate area and perimeter.

Worksheet 5: File Handling

1. **Question 1:**
   Write a program to read the contents of a text file and print it to the console.

2. **Question 2:**
   Create a program to append a user's name and favorite color into a text file.

Practice Projects

1. **Tic-Tac-Toe Game:**
   - Build a simple Tic-Tac-Toe game using Java. Use a 2D array to represent the game board and implement methods to check for a winner.

2. **Student Database System:**

- Create a program that stores student details (name, roll number, marks) in an array of objects. Implement methods to search for a student by roll number and display their information.

3. **Bank Account System:**
   - Design a simple bank account system with classes for Account and methods like deposit(), withdraw(), and checkBalance(). Implement basic transaction logic.

Java Debugging Challenges

1. **Bug 1:**
   Given the following code, find and fix the error that prevents it from running properly:

```
public class DebuggingExample {
  public static void main(String[] args) {
    int[] numbers = new int[5];
    numbers[6] = 10; // Error: Index out of bounds
  }
}
```

2. **Bug 2:**
   Correct the following program that calculates the factorial of a number but doesn't handle negative numbers properly:

```
public class Factorial {
  public static void main(String[] args) {
    int n = -5;
    int fact = 1;
    for (int i = 1; i <= n; i++) {
      fact *= i;
    }
    System.out.println("Factorial of " + n + ": " + fact);
  }
}
```

Additional Activities

1. **Algorithm Practice:**
    - Write an algorithm for sorting an array using bubble sort or insertion sort.
2. **Code Optimization:**
    - Given a program that calculates the Fibonacci sequence, optimize it to improve performance (e.g., avoid redundant calculations).
3. **Unit Testing:**
    - Write unit tests for a method that calculates the power of a number (e.g., power(base, exponent)).

These worksheets, resources, and activities will help students practice Java concepts and develop their coding skills. If you need any specific solutions or additional exercises, feel free to ask!

Sample Java Programs

Here are some sample Java programs that align with the 10th Standard ICSE syllabus, which can help students better understand key programming concepts:

1. Program to Check Whether a Number is Prime

Concepts Covered: Looping, conditional statements.

```
import java.util.Scanner;


public class PrimeNumber {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a number: ");
    int num = sc.nextInt();

    boolean isPrime = true;

    for (int i = 2; i <= num / 2; i++) {
      if (num % i == 0) {
        isPrime = false;
        break;
      }
    }

    if (isPrime && num > 1) {
      System.out.println(num + " is a prime number.");
    } else {
      System.out.println(num + " is not a prime number.");
    }
  }
```

```
}
```

2. Program to Find the Factorial of a Number

Concepts Covered: Looping, methods.

```
import java.util.Scanner;

public class Factorial {
  public static int calculateFactorial(int n) {
    int factorial = 1;
    for (int i = 1; i <= n; i++) {
      factorial *= i;
    }
    return factorial;
  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a number: ");
    int num = sc.nextInt();

    System.out.println("Factorial of " + num + " is: " + calculateFactorial(num));
  }
}
```

3. Program to Reverse a String

Concepts Covered: String manipulation.

```
import java.util.Scanner;

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

```
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a string: ");
    String str = sc.nextLine();

    String reversed = new StringBuilder(str).reverse().toString();

    System.out.println("Reversed String: " + reversed);
  }
}
```

4. Program to Check Whether a String is Palindrome

Concepts Covered: String manipulation, conditional statements.

```
import java.util.Scanner;

public class Palindrome {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a string: ");
    String str = sc.nextLine();

    String reversed = new StringBuilder(str).reverse().toString();

    if (str.equals(reversed)) {
      System.out.println(str + " is a palindrome.");
    } else {
      System.out.println(str + " is not a palindrome.");
    }
  }
}
```

5. Program to Find the Largest Element in an Array

Concepts Covered: Arrays, loops.

```
import java.util.Scanner;

public class LargestInArray {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter the size of the array: ");
      int n = sc.nextInt();

      int[] arr = new int[n];

      System.out.println("Enter " + n + " elements:");
      for (int i = 0; i < n; i++) {
         arr[i] = sc.nextInt();
      }

      int largest = arr[0];
      for (int i = 1; i < n; i++) {
         if (arr[i] > largest) {
            largest = arr[i];
         }
      }

      System.out.println("The largest number in the array is: " + largest);
   }
}
```

6. Program to Display Fibonacci Series

Concepts Covered: Loops, recursion.

```
import java.util.Scanner;

public class FibonacciSeries {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter the number of terms: ");
    int n = sc.nextInt();

    int a = 0, b = 1;

    System.out.print("Fibonacci Series: ");
    for (int i = 1; i <= n; i++) {
      System.out.print(a + " ");
      int nextTerm = a + b;
      a = b;
      b = nextTerm;
    }
  }
}
```

7. Program to Sort an Array in Ascending Order

Concepts Covered: Arrays, sorting.

```
import java.util.Scanner;

public class SortArray {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
```

```
        System.out.print("Enter the size of the array: ");
        int n = sc.nextInt();

        int[] arr = new int[n];

        System.out.println("Enter " + n + " elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        // Sorting the array in ascending order
        for (int i = 0; i < n-1; i++) {
            for (int j = 0; j < n-i-1; j++) {
                if (arr[j] > arr[j+1]) {
                    // Swap arr[j] and arr[j+1]
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
            }
        }

        System.out.println("Sorted array in ascending order:");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

8. Program to Calculate the Sum of Digits of a Number

Concepts Covered: Loops, number manipulation.

```
import java.util.Scanner;

public class SumOfDigits {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();

        int sum = 0;
        while (num != 0) {
            sum += num % 10;  // Extract the last digit
            num /= 10;        // Remove the last digit
        }

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

9. Program to Implement a Simple Calculator

Concepts Covered: Switch-case, arithmetic operations.

```
import java.util.Scanner;

public class Calculator {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter first number: ");
        double num1 = sc.nextDouble();
```

```
System.out.print("Enter second number: ");
double num2 = sc.nextDouble();

System.out.print("Enter operator (+, -, *, /): ");
char operator = sc.next().charAt(0);

double result;
switch (operator) {
  case '+':
    result = num1 + num2;
    break;
  case '-':
    result = num1 - num2;
    break;
  case '*':
    result = num1 * num2;
    break;
  case '/':
    if (num2 != 0) {
      result = num1 / num2;
    } else {
      System.out.println("Cannot divide by zero.");
      return;
    }
    break;
  default:
    System.out.println("Invalid operator.");
    return;
```

```
    }

    System.out.println("Result: " + result);
  }
}
```

10. Program to Count Vowels in a String

Concepts Covered: String manipulation, loops.

```
import java.util.Scanner;

public class CountVowels {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a string: ");
    String str = sc.nextLine();

    int vowelCount = 0;
    str = str.toLowerCase();

    for (int i = 0; i < str.length(); i++) {
      char ch = str.charAt(i);
      if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
        vowelCount++;
      }
    }

    System.out.println("Number of vowels: " + vowelCount);
  }
}
```

Conclusion:

These sample programs cover a variety of core concepts that are essential for the 10th Standard ICSE Java syllabus, including control structures, arrays, string manipulation, and object-oriented programming. You can modify and extend these programs to explore additional functionalities or integrate them into larger projects as part of your studies.

Questions with Answers

Here are **10 questions with answers** based on the content of the **Java Learning Materials for High School Students** document:

---

1. What is Java, and why is it significant?

Answer:
Java is a high-level, platform-independent, object-oriented programming language. It is significant because it is robust, secure, and widely used for applications like mobile apps, web applications, and games.

---

2. What is the purpose of the main() method in Java?

Answer:
The main() method is the entry point of a Java program. It is where the program begins execution. Example:

```
public static void main(String[] args) {
    // Code to execute
}
```

---

3. What is an array in Java?

Answer:
An array is a collection of elements of the same type stored in contiguous memory locations. It allows you to store and access multiple values using an index. Example:

```
int[] numbers = {10, 20, 30};
System.out.println(numbers[0]); // Output: 10
```

---

4. What is the difference between if and switch statements?

Answer:

- if statements are used for conditional branching with complex conditions.
- switch statements are used for choosing between multiple fixed options based on a single variable.

5. What is a constructor in Java?

Answer:
A constructor is a special method used to initialize objects. It has the same name as the class and no return type. Example:

```
public class Student {
  public Student() {
    // Initialization code
  }
}
```

---

6. What are primitive data types in Java?

Answer:
Primitive data types in Java include:

- int: For integers (e.g., 5, 100)
- double: For decimal numbers (e.g., 5.5)
- char: For single characters (e.g., 'A')
- boolean: For true/false values

---

7. Write a program to check if a number is prime.

Answer:

```
import java.util.Scanner;
public class PrimeNumber {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a number: ");
    int num = sc.nextInt();
    boolean isPrime = num > 1;
    for (int i = 2; i <= num / 2; i++) {
```

```
        if (num % i == 0) {
            isPrime = false;
            break;
        }
    }
    System.out.println(num + " is " + (isPrime ? "a prime number." : "not a prime number."));
  }
}
```

---

8. What is the purpose of the Scanner class in Java?

Answer:
The Scanner class is used to read input from the user. Example:

```
import java.util.Scanner;
Scanner sc = new Scanner(System.in);
System.out.print("Enter a number: ");
int num = sc.nextInt();
```

---

9. Explain the concept of encapsulation in Java.

Answer:
Encapsulation is the process of wrapping data (variables) and methods into a single unit (class) and restricting access to them using access modifiers like private. It promotes data hiding and security.

---

10. How can you create a simple calculator in Java?

Answer:

```
import java.util.Scanner;
public class Calculator {
  public static void main(String[] args) {
```

```
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter first number: ");
        double num1 = sc.nextDouble();
        System.out.print("Enter second number: ");
        double num2 = sc.nextDouble();
        System.out.print("Enter operation (+, -, *, /): ");
        char op = sc.next().charAt(0);
        switch (op) {
            case '+': System.out.println("Result: " + (num1 + num2)); break;
            case '-': System.out.println("Result: " + (num1 - num2)); break;
            case '*': System.out.println("Result: " + (num1 * num2)); break;
            case '/': System.out.println(num2 != 0 ? "Result: " + (num1 / num2) : "Cannot divide
by zero."); break;
            default: System.out.println("Invalid operation.");
        }
    }
}
```

About ITProAcademy.co.in

Empowering Learners in the Digital Age

ITProAcademy.com is a premier online platform dedicated to equipping individuals with the technical knowledge and skills required to thrive in the ever-evolving world of IT and cloud technologies. Established with a mission to bridge the gap between aspirations and achievements, ITProAcademy offers cutting-edge training programs tailored for both students and professionals.

With a strong focus on certification-based learning, ITProAcademy.com specializes in areas such as Microsoft Azure, AWS, Google Cloud, Artificial Intelligence, AIOps, Automation, Security, and more. The platform provides a comprehensive suite of services, including certification exam preparation, deep-dive webinars, and real-world scenario-based training.

Key Highlights of ITProAcademy.co.in:

- **Programming Languages Made Fun and Easy for Kids & Students:** Our interactive courses in **HTML, C++, Java, and Python** make learning to code fun and engaging. We inspire young learners to explore their creativity, develop problem-solving skills, and build a strong coding foundation, all while enjoying the journey. Join us to turn curiosity into coding skills and imagination into innovation!
- **Expert-Led Training:** Courses are designed and delivered by seasoned industry professionals with hands-on experience in cloud technologies, infrastructure automation, and cybersecurity.
- **Certifications for Growth:** Dedicated programs for Microsoft certifications, cloud computing, AI, and other emerging technologies to help learners stay ahead in their careers.
- **Innovative Learning Approach:** ITProAcademy integrates theoretical concepts with practical demonstrations, ensuring learners gain both knowledge and applicable skills.
- **Custom Learning Tracks:** Whether you are a student looking to begin your IT journey or a professional seeking to upgrade skills, ITProAcademy offers customized learning tracks to meet individual needs.

Our Commitment at ITProAcademy.co.in, we believe in empowering learners to excel in the digital landscape. Our aim is to make advanced IT education accessible, affordable, and impactful for individuals and businesses alike.

Join us at ITProAcademy.com and take the next step towards mastering IT technologies and achieving your career goals!

Disclaimer

This document, Handbook for **Java** for High School Students, is exclusively designed for **high school students** as part of the Programming Languages Made Fun and Easy for Kids & Students initiative. The content is intended solely for **learning and practice purposes** and aims to make programming engaging and accessible for young learners.

All material presented in this handbook is of a general nature and has been referenced from multiple sources, including the **Internet** and **ChatGPT**. While every effort has been made to ensure the accuracy and relevance of the content, it should be used as a supplementary guide to enhance students' understanding of Scratch programming.

The authors and contributors to this handbook disclaim any responsibility for errors, omissions, or the outcomes of any actions taken based on the information provided. This material should be used under the guidance of educators or guardians to support the students' learning journey.

Contact Us

Website Link (ITProAcademy.co.in): https://itproacademy.co.in/

Contact Email Address: support@itproacademy.co.in

Services & Programs Link: https://itproacademy.co.in/services-%26-programs

Contact Us Link: https://itproacademy.co.in/contact-us