Structure

A comprehensive introduction to Java programming fundamentals

Introduction to Java & Project Setup

Why Java?

- **Statically Typed** - Type checking at compile-time catches errors early
- **Platform Independence** - Write once, run anywhere (WORA)
- **Industry Standard** - Used in enterprise applications, Android development
- **Educational Value** - Strong foundation for learning OOP concepts

Static Typing Benefits

- Early error detection
- Better performance
- Improved code readability

Project Structure in VS Code

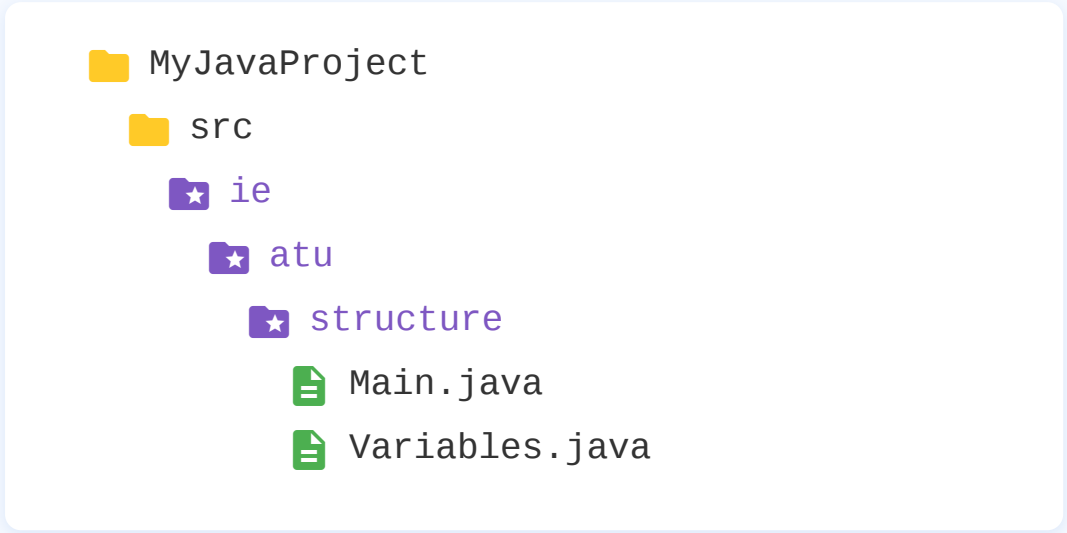
- **src folder** - Contains all source code files
- **Packages** - Organize related classes together
- **Classes** - Individual Java source files

Package Naming Convention

- Reverse domain name: `ie.atu.structure`
- All lowercase letters
- Reflects directory structure

Project Structure in VS Code

Directory Structure



- **src folder** - Contains all source code files
- **Packages** - Organize related classes together
- **Classes** - Individual Java source files

Package Naming Convention

Naming Rules

- All lowercase letters
- Reverse domain name format
- Reflects directory structure

Example: ie.atu.structure

Directory path: src/ie/atu/structure/

```
// Package declaration at top of file
package ie.atu.structure;

public class Main {
  // Class code here
}
```

Variables and Data Types

Data Types in Java

Primitive Types

- **int** - Whole numbers (32-bit)
- **double** - Decimal numbers (64-bit)
- **char** - Single characters (16-bit)
- **boolean** - true or false values

Reference Types

- **String** - Sequence of characters
- **Arrays** - Collections of same type
- **Objects** - Instances of classes

Memory Perspective

How Data is Stored in Memory

Primitive Types

int age	25
double gpa	3.75
char grade	'A'
boolean pass	true

Reference Types

String name	→ "John Doe"
int[] scores	→ [85, 92, 78]
Students	→ Student@1a2b3c

- **Primitives** - Store values directly in memory
- **References** - Store memory addresses that point to objects

Interactive Demo

Which data type would you use for:

`Student's age: int` `GPA: double`

`Name: String` `Course fee: double`

Variables and Data Types - Code Example

SimpleVariables.java

```java
// SimpleVariables.java - Demonstrating primitive and reference types
package ie.atu.structure;
public class SimpleVariables {
    public static void main(String[] args) {
        // Primitive types - store values directly
        int studentAge = 20;           // Whole number
        double studentGpa = 3.75;         // Decimal number
        char grade = 'A';               // Single character
        boolean isPassing = true;      // true or false
        // Reference types - store memory addresses that point to objects
        String studentName = "John Doe";  // String object
        // Output the values
        System.out.println("Student Name: " + studentName);
        System.out.println("Student Age: " + studentAge);
        System.out.println("Student GPA: " + studentGpa);
        System.out.println("Grade: " + grade);
        System.out.println("Passing Status: " + isPassing);
    }
}
```

Primitive Types

- **int** - Stores whole numbers directly
- **double** - Stores decimal values directly
- **char** - Stores single character using single quotes
- **boolean** - Stores true or false values

Reference Types

- **String** - Stores reference to a String object
- String values use double quotes
- Reference types can be null
- Memory allocated on the heap

Operators and Arithmetic

Arithmetic Operators

+ Addition	- Subtraction	* Multiplication
/ Division	% Modulo	

Division Types

Integer Division

`10 / 3`

= 3

(Truncates decimal)

Floating-Point

`10.0 / 3`

= 3.333...

(Preserves decimal)

Mini Quiz

What does **7 % 3** evaluate to?

1 2 3 4

Code Examples

```java
// Arithmetic operations in Java int a = 10; int b = 3; int sum = a + b; // 13 int diff = a - b; // 7 int product = a * b; // 30 int quotient = a / b; // 3 (integer division) int remainder = a % b; // 1 (modulo operation) // For floating-point division double result = (double)a / b; // 3.333... // Output System.out.println("Sum: " + sum); System.out.println("Difference: " + diff); System.out.println("Product: " + product); System.out.println("Quotient: " + quotient); System.out.println("Remainder: " + remainder); System.out.println("Result: " + result);
```

Integer Division
Truncates decimal part

Floating-Point
Preserves decimals

Modulo Operator
Returns remainder

Type Casting

Type Casting Types

Implicit (Widening)

- **Automatic** conversion
- **No data loss**

Explicit (Narrowing)

- Requires **cast operator** `(type)`
- **Potential data loss**

Conversion Examples

Widening

```
int i = 100;
double d = i;
```

↓

```
Result: 100.0
No data loss
```

Narrowing

```
double d = 100.75;
int i = (int)d;
```

↓

```
Result: 100
Decimal lost
```

Dangers of Explicit Casting

- **Data truncation** - Loss of decimal places
- **Overflow** - Value exceeds target type range

Code Examples

```
// Type casting examples double price = 19.99; int dollars =
(int)price; // Explicit narrowing // Output the results
System.out.println("Original price: " + price);
System.out.println("Dollars only: " + dollars); // Another
example with precision loss int largeNumber = 130; byte
smallNumber = (byte)largeNumber; // Potential overflow
System.out.println("Large number: " + largeNumber);
System.out.println("Small number: " + smallNumber); // -126!
```

Widening
Automatic • Safe

Narrowing
Manual • Risky

Precision
Loss • Overflow

Type Casting - Code Example

TypeCastingDemo.java

```java
// TypeCastingDemo.java - Demonstrating widening and narrowing type casting
package ie.atu.structure;
public class TypeCastingDemo {
    public static void main(String[] args) {
        // Widening Type Casting (Implicit)
        int intValue = 100;
        long longValue = intValue;          // int to long
        float floatValue = longValue;        // long to float
        double doubleValue = floatValue;      // float to double
        System.out.println("Widening Type Casting (Implicit):");
        System.out.println("int value: " + intValue);
        System.out.println("long value: " + longValue);
        System.out.println("float value: " + floatValue);
        System.out.println("double value: " + doubleValue);
        System.out.println();
        // Narrowing Type Casting (Explicit)
        double price = 19.99;
        float priceFloat = (float)price;        // double to float
        long priceLong = (long)priceFloat;      // float to long
        int priceInt = (int)priceLong;          // long to int
        System.out.println("Narrowing Type Casting (Explicit):");
        System.out.println("Original price: " + price);
        System.out.println("price as float: " + priceFloat);
        System.out.println("price as long: " + priceLong);
        System.out.println("price as int: " + priceInt + " (data loss!)");
        System.out.println();
        // Example of overflow when narrowing
        int largeNumber = 130;
        byte smallNumber = (byte)largeNumber;  // Potential overflow
        System.out.println("Overflow Example:");
        System.out.println("Large number: " + largeNumber);
        System.out.println("Small number: " + smallNumber + " (overflow!)");
    }
}
```

↑ Widening (Implicit)

- Automatic conversion
- No data loss
- Smaller to larger type

↓ Narrowing (Explicit)

- Requires cast operator **(type)**
- Potential data loss
- Larger to smaller type

⚠ Potential Issues with Narrowing

- **Data truncation** - Decimal places lost when converting to integer types
- **Overflow** - Values outside target type range wrap around (e.g., 130 → -126 for byte)

User Input with Scanner

Getting User Input

Importing Packages

- **java.util.Scanner** - Contains the Scanner class
- Import statement: **import java.util.Scanner;**
- Place at the **top of your Java file**

Creating a Scanner Object

- **Scanner scanner = new Scanner(System.in);**
- **System.in** - Standard input stream (keyboard)

Important: Closing Scanner

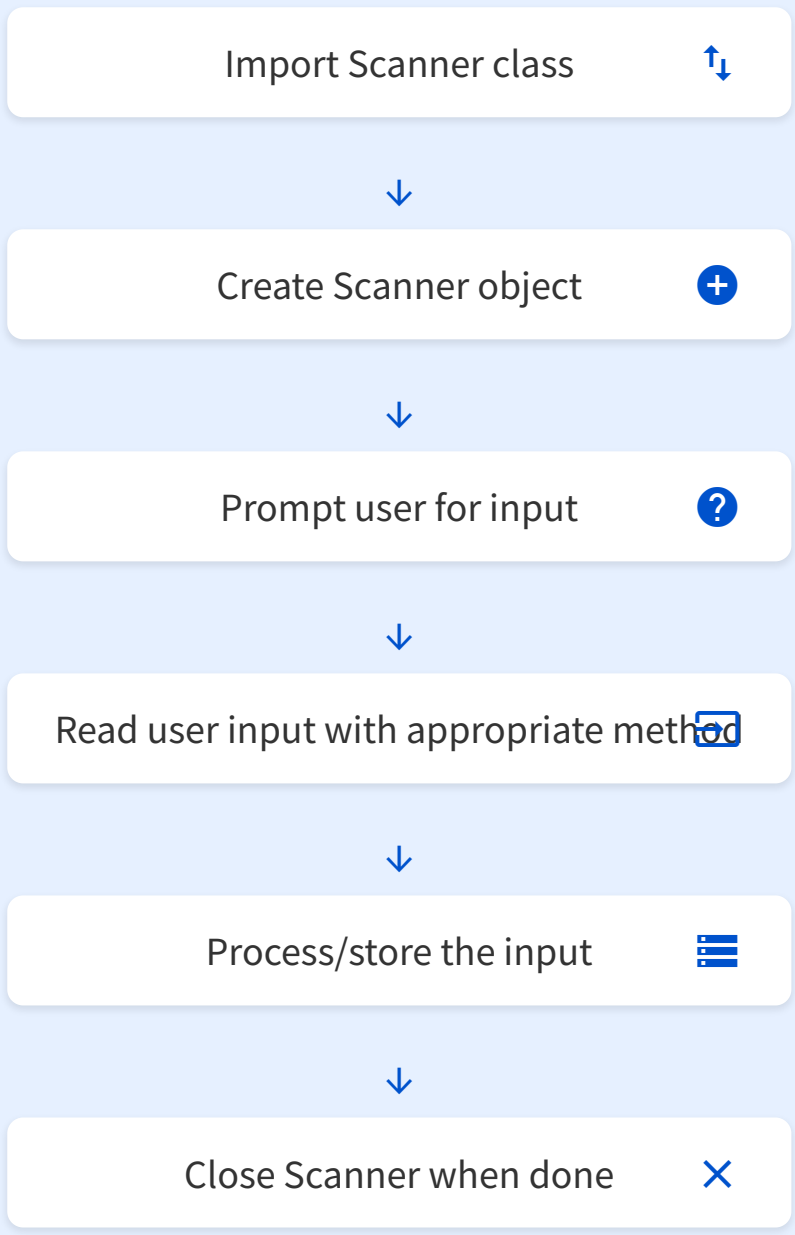
- **scanner.close();** - Releases system resources
- Prevents **resource leaks**
- Call when you're **done with input**

Scanner Methods

Common Input Methods

`nextInt()` Reads integer	`nextDouble()` Reads double
`nextLine()` Reads entire line	`next()` Reads single word
`nextBoolean()` Reads boolean	`nextLong()` Reads long

Input/Output Flowchart



User Input with Scanner - Code Example

UserInputExample.java

```java
// UserInputExample.java - Reading user's name and age
package ie.atu.structure;
// Import the Scanner class
import java.util.Scanner;
public class UserInputExample {
    public static void main(String[] args) {
        // Create a Scanner object
        Scanner scanner = new Scanner(System.in);

        // Prompt user for name
        System.out.print("Enter your name: ");

        // Read user input
        String name = scanner.nextLine();

        // Prompt user for age
        System.out.print("Enter your age: ");

        // Read user input
        int age = scanner.nextInt();

        // Print greeting
        System.out.println();
        System.out.println("Hello, " + name + "!");
        System.out.println("You are " + age + " years old.");

        // Close the scanner to prevent resource leaks
        scanner.close();
    }
}
```

Key Points

- **Import Scanner** from java.util package
- **Create Scanner object** with System.in
- **nextLine()** reads entire line of text
- **nextInt()** reads integer value

Code Structure

- **Prompt user** before reading input
- Use appropriate **method** for data type
- **Concatenate** strings with + operator
- **Always close** Scanner when done

Sample Output

```
Enter your name: John
Enter your age: 25

Hello, John!
You are 25 years old.
```

Best Practices

- ★ Close Scanner to **prevent resource leaks**
- ★ Use **nextLine()** after **nextInt()** to clear input buffer

Mini Programs: Math & Logic (TemperatureConverter)

Temperature Conversion Program

Σ Celsius to Fahrenheit: (C × 9/5) + 32

Σ Fahrenheit to Celsius: (F - 32) × 5/9

```java
// TemperatureConverter.java - Convert between Celsius and Fahrenheit
package ie.atu.structure;
import java.util.Scanner;
public class TemperatureConverter {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Temperature Converter");
        System.out.println("1. Celsius to Fahrenheit");
        System.out.println("2. Fahrenheit to Celsius");
        System.out.print("Enter your choice (1 or 2): ");

        int choice = scanner.nextInt();

        if (choice == 1) {
            // Celsius to Fahrenheit conversion
            System.out.print("Enter temperature in Celsius: ");
            double celsius = scanner.nextDouble();

            // Apply conversion formula
            double fahrenheit = (celsius * 9/5) + 32;

            System.out.println(celsius + "°C = " + fahrenheit + "°F");
        }
        else if (choice == 2) {
            // Fahrenheit to Celsius conversion
            System.out.print("Enter temperature in Fahrenheit: ");
            double fahrenheit = scanner.nextDouble();

            // Apply conversion formula
            double celsius = (fahrenheit - 32) * 5/9;

            System.out.println(fahrenheit + "°F = " + celsius + "°C");
        }
        else {
            System.out.println("Invalid choice!");
        }

        scanner.close();
    }
}
```

Program Features

- **User choice** between C → F or F → C
- **if-else** for decision making
- **Mathematical formulas** for conversion

Key Concepts

- **Scanner** for user input
- **Double** for decimal precision
- **Arithmetic operators** for calculations

Sample Output

```
Temperature Converter
1. Celsius to Fahrenheit
2. Fahrenheit to Celsius
Enter your choice (1 or 2): 1
Enter temperature in Celsius: 25
25.0°C = 77.0°F
```

Mini Programs: Math & Logic (MinFinder & AverageCalculator)

MinFinder.java

```java
// MinFinder.java - Find smallest number
package ie.atu.structure;
import java.util.Scanner;
public class MinFinder {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter first number: ");
        int num1 = scanner.nextInt();

        System.out.print("Enter second number: ");
        int num2 = scanner.nextInt();

        // Using Math.min() to find smaller number
        int min = Math.min(num1, num2);

        System.out.println("The smaller number is: " + min);

        // Using Math.max() to find larger number
        int max = Math.max(num1, num2);

        System.out.println("The larger number is: " + max);

        scanner.close();
    }
}
```

AverageCalculator.java

```java
// AverageCalculator.java - Calculate average
package ie.atu.structure;
import java.util.Scanner;
public class AverageCalculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("How many numbers? ");
        int count = scanner.nextInt();

        double sum = 0;

        for (int i = 0; i < count; i++) {
            System.out.print("Enter number " + (i + 1) + ": ");
            double num = scanner.nextDouble();

            // Summation of all numbers
            sum += num;
        }

        // Division for average calculation
        double average = sum / count;

        System.out.println("Sum: " + sum);
        System.out.println("Average: " + average);

        scanner.close();
    }
}
```

Real-World Applications

- **Education** - Calculating grade averages
- **Data Analysis** - Finding min/max values in datasets
- **Sports** - Determining highest/lowest scores

Key Concepts

- **Math.min()** - Returns smaller of two values
- **Math.max()** - Returns larger of two values
- **Summation** - Accumulating values with += operator
- **Division** - Calculating average with / operator

Iteration
Using for loops to process multiple values

Data Types
Using double for decimal precision in calculations

User Input
Reading multiple values with Scanner

Control Structures (Do-While Loop)

Do-While Loop

Key Characteristics

- **Ensures at least one execution** of the loop body
- Condition is checked **after** the loop body executes
- Syntax: **do { ... } while (condition);**

Flow of Execution

Execute loop body

↓

Check condition

↓

If true, repeat from step 1

↓

If false, exit loop

While vs Do-While

While Loop	Do-While Loop
Condition checked before execution	Condition checked after execution
May execute zero times	Executes at least once

Code Example

```java
// PasswordValidation.java - Do-While Loop Example
package ie.atu.structure;
import java.util.Scanner;
public class PasswordValidation {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        String password;
        String correctPassword = "Java123";

        do {
            // This code will execute at least once
            System.out.print("Enter password: ");
            password = scanner.nextLine();

            if (!password.equals(correctPassword)) {
                System.out.println("Incorrect password. Try again.");
            }
        } while (!password.equals(correctPassword));  // Condition checked after execution

        System.out.println("Access granted!");
        scanner.close();
    }
}
```

Use Cases

- **Password validation** - Must prompt at least once
- **Menu systems** - Display menu before checking choice
- **Input validation** - Ensure valid user input

Control Structures (Do-While Loop)

Key Characteristics

Guaranteed Execution
Runs at least once

Post-Test Loop
Condition checked after execution

Syntax
`do { ... } while (condition);`

Flow of Execution

Execute loop body

↓

Check condition

↓

If true
Repeat

If false
Exit

While vs Do-While

While Loop
Condition checked before execution
May execute zero times

Do-While Loop
Condition checked after execution
Executes at least once

Code Example

```java
// PasswordValidation.java - Do-While Loop Example
package ie.atu.structure;
import java.util.Scanner;
public class PasswordValidation {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        String password;
        String correctPassword = "Java123";

        do {
            // This code will execute at least once
            System.out.print("Enter password: ");
            password = scanner.nextLine();

            if (!password.equals(correctPassword)) {
                System.out.println("Incorrect password. Try again."
            }
        } while (!password.equals(correctPassword));  // Condition

        System.out.println("Access granted!");
        scanner.close();
    }
}
```

Use Cases

Password Validation
Must prompt at least once

Menu Systems
Display menu before checking choice

Input Validation
Ensure valid user input

Guaranteed Execution
Runs at least once

User Input
Ideal for validation

Simplicity
Cleaner code structure