Chapter 2:
Elementary Programming

2.1. Introduction

Programming Fundamentals

This chapter serves as a gentle introduction to the fundamental concepts of programming. It covers the essential building blocks of a Java program, including data types, variables, operators, and control structures.

2.2. Writing a Simple Program

Problem-Solving and Algorithms

- Problem-solving involves designing a sequence of steps to solve a problem. This sequence is known as an algorithm.
- An algorithm is a set of well-defined steps to solve a problem. It is independent of any programming language.

Problem Area of a Circle

- **Step 1**: Read in the circle’s radius (input).
  - Prompt the user to enter the radius of the circle.
  - Use a Scanner object to read the radius as a double.

- **Step 2**: Compute the area using the formula:

$$\text{area} = \text{radius} \times \text{radius} \times \pi$$

  - Declare a variable `radius` of type `double` to store the radius.
  - Declare a variable `area` of type `double` to store the calculated area.
  - Use the formula to calculate the area: `area = radius * radius * Math.PI;`

- **Step 3**: Display the result (output).
  - Print the calculated area to the console using `System.out.println`.

Variables and Data Types

- Variables are used to store data. For instance, `radius` and `area` are variables to store the radius and area of the circle.
- Declare variables with their data types (e.g., `double radius; double area;`).
- Data types are essential to let the compiler know what kind of data will be stored in the variables.

Reading Input and Displaying Output

- Prompt the user to enter the radius.
- Assign a value to `radius` (initially it can be a fixed value, but later prompt the user for input).
- Compute the area and display it using `System.out.println`.

Reading Input and Displaying Output

Example:

```
public class ComputeArea {
    public static void main(String[] args) {
        // Declare variables
        double radius;
        double area;

        // Assign a value to radius
        radius = 20;

        // Compute area
        area = radius * radius * 3.14159;

        // Display result
        System.out.println("The area for the circle of radius " + radius + " is " + area);
    }
}
```

Output:

```
The area for the circle of radius 20.0 is 1256.636
```

2.3. Reading Input from the Console

Introduction to Console Input

- Instead of hardcoding values, you can prompt the user to enter values. This makes the program more flexible and interactive. The Scanner class, which is part of the java.util package, is used for this purpose.
- The Scanner class is used to read input from the console.
- To read input from the console, you need to create an instance of the Scanner class.

Using the Scanner Class

Creating a Scanner Object:

To read input from the console, you need to create an instance of the Scanner class. This is done using the following code:

```java
Scanner input = new Scanner(System.in);
```

Using the Scanner Class

Reading Different Types of Input

- To read an integer:

```
int number = input.nextInt();
```

- To read a double:

```
double value = input.nextDouble();
```

- To read a string:

```
String text = input.nextLine();
```

Example:

```
import java.util.Scanner;

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

        // Prompt the user to enter a radius
        System.out.print("Enter a number for radius: ");
        double radius = input.nextDouble();

        // Compute area
        double area = radius * radius * 3.14159;

        // Display result
        System.out.println("The area for the circle of radius " + radius + " is " + area);
    }
}
```

Output:

```
Enter a number for radius: 5
The area for the circle of radius 5.0 is 78.53975
```

2.4. Identifiers

Identifier Rules

- An identifier can consist of letters, digits, underscores (_), and dollar signs ($).
- Must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
- An identifier cannot be a reserved word in Java (such as `class`, `public`, `int`, etc.).
- Identifiers can be of any length.

Examples of Valid and Invalid Identifiers

Valid Identifiers:

`$2` , `ComputeArea` , `area` , `radius` , `print` , `MAX_VALUE` , `numberOfStudents` , `totalAmount` , `total_amount` , `_value` , `$_value` , `value$` , `value_` , `value2` , `value_2` , `value2_`

Invalid Identifiers:

`2A`, `d+4`, `total amount`, `total-amount`, `total.amount`, `total&amount`, `total*amount`, `total/amount`, `total%amount`, `total#amount`, `total@amount`, `total!amount`, `total^amount`, `total(amount`, `total)amount`, `total[amount`, `total]amount`, `total{amount`, `total}amount`, `total|amount`, `total\amount`, `total:amount`, `total;amount`, `total"amount`, `total'amount`, `total<amount`, `total>amount`, `total,amount`, `total?amount`, `total=amount`, `total+amount`, `total-amount`, `` total`amount ``.

Java is Case Sensitive

- Java is case-sensitive, meaning that uppercase and lowercase letters are treated as distinct characters.

Example:

`value`, `Value`, and `VALUE` are considered different identifiers in Java.

Descriptive Identifiers

- Use descriptive identifiers to make your code more readable and maintainable.
- Choose meaningful names that reflect the purpose of the variable or method.
- Avoid using single-letter variable names (e.g., `x`, `y`, `z`) unless they are used as loop counters.

Example:

```java
int numberOfStudents;
double totalAmount;
String studentName;
```

Avoid Using `$` Character

- While the `$` character is allowed in identifiers, it is not recommended to use it in Java programming.
- The `$` character is often used by Java compilers to generate class files, and using it in identifiers can lead to confusion.
- It is best to avoid using the `$` character in identifiers to maintain code clarity and readability.

Example:

```java
int total$Amount; // Valid but not recommended
```

2.5. Variables

What are Variables?

- Variables are used to store data in a program. They have a name, a data type, and a value.
- Variables can be used to store different types of data, such as numbers, text, and objects.

Declaring Variables

A variable must be declared before it can be used. Declaration involves specifying the variable's name and data type.

Syntax:

```
dataType variableName;
```

Explanation:

- `dataType` is the type of data the variable will store (e.g., `int`, `double`, `String`).
- `variableName` is the name of the variable (e.g., `count`, `radius`).

Example:

```java
int count;
double radius;
```

Initializing Variables

Variables can be initialized at the time of declaration.

Example:

```java
int count = 1;
double radius = 2.5;
```

Multiple Declarations

Variables of the same data type can be declared together, separated by commas.

Example:

```java
int i, j, k;
```

Variable Scope

- The scope of a variable is the part of the program where the variable can be accessed.
- Variables must be declared within their scope before they can be used.

Example:

```java
public class Test {
    public static void main(String[] args) {
        int x = 1; // x is declared within the main method
        System.out.println(x); // x can be accessed here
    }
}
```

Assigning Values to Variables

- Values can be assigned to variables using the assignment operator `=`.
- Syntax: `variableName = expression;`

Example:

```
count = 10;
radius = 5.5;
```

2.6. Assignment Statements and Expressions

Assignment Statements

- After declaring a variable, you can assign it a value using the assignment operator `=`.
- Syntax: `variable = expression;`

Example:

```
int y = 1; // Assign 1 to variable y
double radius = 1.0; // Assign 1.0 to variable radius
int x = 5 * (3 / 2); // Assign the result of the expression to x
x = y + 1; // Assign the addition of y and 1 to x
double area = radius * radius * 3.14159; // Compute area
```

Expressions

- An expression represents a computation involving values, variables, and operators that evaluates to a value.
- In an assignment statement, the expression on the right-hand side of the assignment operator is evaluated first, and then the value is assigned to the variable on the left-hand side.

Example:

```
x = x + 1; // The result of x + 1 is assigned to x
```

Assignment Operators

- The assignment operator `=` is used to assign a value to a variable.
- Other assignment operators include `+=`, `-=`, `*=`, `/=`, and `%=`.
- These operators combine an arithmetic operation with the assignment operation.

Example:

```
x += 1; // Equivalent to x = x + 1
y -= 2; // Equivalent to y = y - 2
z *= 3; // Equivalent to z = z * 3
```

Assignment Expressions

In Java, an assignment statement is also an expression that evaluates to the value assigned to the variable.

Example:

```java
System.out.println(x = 1); // Assigns 1 to x and prints 1
```

Chained Assignments

You can assign the same value to multiple variables in one statement using chained assignments.

Example:

```
i = j = k = 1; // Assigns 1 to k, then j, then i
```

Type Compatibility

The data type of the variable on the left must be compatible with the data type of the value on the right.

Example:

```
int x = 1.0; // Incorrect because 1.0 is a double value and x is an int
```

2.7. Named Constants

What are Named Constants?

- A named constant is an identifier that represents a permanent value that doesn't change.
- Named constants are also known as "final variables" in Java.

Declaring Named Constants

- Named constants are declared using the `final` keyword followed by the data type and the constant's name.
- Syntax: `final dataType CONSTANT_NAME = value;`

Example: Declaring a Named Constant

```java
final double PI = 3.14159;
```

Naming Conventions

- By convention, the names of constants are written in all uppercase letters with underscores separating words.

Example: `MAX_SPEED`, `NUMBER_OF_DAYS`.

Benefits of Using Constants

- **Readability**: Constants make the code more readable by providing meaningful names for values.
- **Maintainability**: If the value needs to be changed, it can be updated in one place rather than throughout the code.
- **Error Reduction**: Using constants helps prevent errors caused by using incorrect or inconsistent values.

Example: Benefits of Using Constants

```java
final double PI = 3.14159;
double radius = 5.0;
double area = PI * radius * radius;
System.out.println("The area of the circle is " + area);
```

Explanation:

- The value of `PI` is defined once and used throughout the program.
- If the value of `PI` needs to be changed, it can be updated in one place.
- Using `PI` instead of the literal value makes the code more readable.

2.8. Naming Conventions

Variable and Method Names

- Use lowercase letters for variable and method names.
- If a name consists of several words, concatenate them into one word and capitalize the first letter of each subsequent word (camelCase).

Example: `numberOfStudents`, `computeArea`.

Class Names

- Capitalize the first letter of each word in a class name.

Example: `ComputeArea`, `System`.

Constant Names

- Use all uppercase letters for constants.
- Separate words with underscores (_).

Example: `PI`, `MAX_VALUE`.

Avoiding Conflicts

- Do not use class names that are already used in the Java library to avoid conflicts.

Example: Avoid using `System` as a class name.

2.9. Numeric Data Types and Operations

Numeric Data Types

- Java has six numeric types for integers and floating-point numbers: `byte`, `short`, `int`, `long`, `float`, and `double`.
- Each data type has a specific range and storage size.

Example:

- `byte`: -128 to 127
- `short`: -32,768 to 32,767
- `int`: -2,147,483,648 to 2,147,483,647
- `long`: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
- `float`: ±3.4028235E38 (6-9 significant digits)
- `double`: ±1.7976931348623157E308 (15-17 significant digits)

Reading Numeric Values from the Keyboard

- The `Scanner` class can be used to read numeric values from the keyboard. Methods include:
    - `nextByte()`
    - `nextShort()`
    - `nextInt()`
    - `nextLong()`
    - `nextFloat()`
    - `nextDouble()`

Arithmetic Operators

- Java supports the following arithmetic operators: `+`, `-`, `*`, `/`, and `%`.
- The modulus operator `%` returns the remainder of a division operation.
- The division operator `/` performs integer division if both operands are integers.
- To force floating-point division, cast one of the operands to a floating-point type.

- The division of two integers truncates the fractional part.
- To perform floating-point division, use floating-point numbers.
- The result of a division operation is a floating-point number.
- The division of two integers results in an integer if both operands are integers.

Example: Division Operations

```java
int x = 5;
int y = 2;
System.out.println(x / y); // Displays 2
System.out.println((double) x / y); // Displays 2.5
```

Operations on Mixed Types

- When performing operations with mixed data types, Java automatically converts the operands to the appropriate type.

Example: Operations on Mixed Types

```java
int i = 5;
double d = 2.5;
System.out.println(i * d); // Displays 12.5
```

Exponentiation

- Use `Math.pow(a, b)` to compute `a` raised to the power of `b`.

Example:

```
double result = Math.pow(2, 3); // result is 8.0
```

2.10. Numeric Literals

Integer Literals

- Integer literals can be assigned to integer variables as long as they fit within the variable's range. If the literal is too large, it causes a compile error.
- By default, an integer literal is an `int` type. To denote a `long` literal, append the letter `L` or `l` to it.

Example: Integer Literals

```
int number = 34;
long bigNumber = 2147483648L;
```

Binary, Octal, and Hexadecimal Literals

- To denote a binary integer literal, use a leading `0b` or `0B`. For octal literals, use a leading `0`. For hexadecimal literals, use a leading `0x` or `0X`.

Example: Binary, Octal, and Hexadecimal Literals

```java
int binary = 0b1111;  // 15 in binary
int octal = 07777;    // 4095 in octal
int hex = 0xFFFF;     // 65535 in hexadecimal
```

Floating-Point Literals

- Floating-point literals are written with a decimal point and are by default of the `double` type. To make a number a `float`, append the letter `f` or `F`. To make a number a `double`, append the letter `d` or `D` (though this is optional).

Example: Floating-Point Literals

```java
double a = 5.0;
float b = 5.0f;
```

Scientific Notation

- Floating-point literals can be written in scientific notation in the form of `a * 10^b`. In Java, this is represented as `aEb` or `aE+b`.

Example: Scientific Notation

```java
double c = 1.23456E2;   // 123.456
double d = 1.23456E-2;  // 0.0123456
```

Underscores in Numeric Literals

- To improve readability, underscores can be used to separate groups of digits in numeric literals.

Example:

```java
long largeValue = 23_234_454_519L;
double preciseValue = 23.24_4545_4519_3415;
```

2.11. JShell

JShell Overview

- JShell is an interactive tool that allows you to execute Java statements and expressions one at a time.
- This feature is commonly known as REPL (Read-Evaluate-Print Loop), which evaluates expressions, executes statements, and displays results.

Launching JShell:

- To use JShell, you need to have JDK 9 or higher installed.
- Open a command prompt or terminal and type `jshell` to launch it.

Using JShell

You can enter Java statements directly at the JShell prompt.

Example: Entering a Statement

```
jshell> int x = 5;
jshell> System.out.println(x);
```

Inspecting Variables

- JShell provides commands to inspect variables and their values.
- Use `/vars` to list all declared variables.

Example: Listing Variables

```
jshell> /vars
```

Editing Code

- You can use the `/edit` command to edit the code you have entered from the JShell prompt.
- This opens an edit pane where you can make changes.

Automatic Variable Creation

If you enter a value without specifying a variable, JShell will automatically create a variable for it.

Example: Automatic Variable Creation

~~~
```java
jshell> 6.8
$7 ==> 6.8
~~~

Exiting JShell

- To exit JShell, type `/exit`.

Example: Exiting JShell

```
jshell> /exit
```

2.12. Evaluating Expressions and Operator Precedence

Arithmetic to Java Expressions

- Writing a numeric expression in Java involves translating an arithmetic expression using Java operators.

Example: Arithmetic Expression

$$\frac{3+4x}{5} - 10 \cdot (y-5) \cdot (a+b+c)/x + 9 \cdot \left(\frac{4}{x} + \frac{9+x}{y}\right)$$

```
(3 + 4 * x) / 5 - 10 * (y - 5) * (a + b + c) / x + 9 * (4 / x + (9 + x) / y)
```

Operator Precedence

- Operators within parentheses are evaluated first. If parentheses are nested, the expression in the inner parentheses is evaluated first.
- The order of precedence for operators is as follows:
  - **First**: Multiplication (`*`), Division (`/`), and Remainder (`%`)
  - **Second**: Addition (`+`) and Subtraction (`-`)

Example: Operator Precedence

Here's how the expression `3 + 4 * 4 + 5 * (4 + 3) - 1` is evaluated step by step:

1. `3 + 4 * 4 + 5 * (4 + 3) - 1`
2. `3 + 4 * 4 + 5 * 7 - 1`
3. `3 + 16 + 5 * 7 - 1`
4. `3 + 16 + 35 - 1`
5. `19 + 35 - 1`
6. `54 - 1`
7. `53`

Example: A sample program that converts Fahrenheit to Celsius using the formula:

$$\text{Celsius} = \frac{5}{9} \cdot (\text{Fahrenheit} - 32)$$

```java
import java.util.Scanner;

public class FahrenheitToCelsius {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter a degree in Fahrenheit: ");
        double fahrenheit = input.nextDouble();

        // Convert Fahrenheit to Celsius
        double celsius = (5.0 / 9) * (fahrenheit - 32);
        System.out.println("Fahrenheit " + fahrenheit + " is " + celsius + " in Celsius");
    }
}
```

Output:

```
Enter a degree in Fahrenheit: 98.6
Fahrenheit 98.6 is 37.0 in Celsius
```

2.13. Case Study: Displaying the Current Time

Practice.

2.14. Augmented and Assignment Operators

Augmented Assignment Operators

- These operators combine arithmetic operations (`+`, `-`, `*`, `/`, `%`) with the assignment operator (`=`).
- Syntax: `variable op= expression;`

Example: Augmented Assignment Operators

- `count += 1;` is equivalent to `count = count + 1;`
- `sum -= 5;` is equivalent to `sum = sum - 5;`
- `product *= 2;` is equivalent to `product = product * 2;`
- `quotient /= 3;` is equivalent to `quotient = quotient / 3;`
- `remainder %= 4;` is equivalent to `remainder = remainder % 4;`

Expression Evaluation

The expression on the right-hand side of the augmented assignment operator is evaluated first, and then the result is assigned to the variable on the left-hand side.

Example: Expression Evaluation

```
x /= 4 + 5.5 * 1.5; // Same as x = x / (4 + 5.5 * 1.5);
```

In Java, you can use augmented assignment operators to make your code more concise and readable.

Example: Augmented Assignment Operators

```java
int a = 6;
a += a + 1; // a becomes 13
System.out.println(a); // Displays 13

double b = 6.5;
b /= 2; // b becomes 3.25
System.out.println(b); // Displays 3.25
```

2.15. Increment and Decrement Operators

Increment and Decrement Operators

- `++` (increment) increases the value of a variable by 1.
- `--` (decrement) decreases the value of a variable by 1.

Postfix and Prefix Forms

- **Postfix**: The operator is placed after the variable (e.g., `i++` or `i--`). The current value of the variable is used in the expression, then the variable is incremented or decremented.
- **Prefix**: The operator is placed before the variable (e.g., `++i` or `--i`). The variable is incremented or decremented first, then the new value is used in the expression.

Example: Postfix and Prefix Forms

```java
int i = 3, j = 3;
i++;  // Postfix increment: i becomes 4 after this line
--j;  // Prefix decrement: j becomes 2 before this line
```

Differences Between Postfix and Prefix

The effect of postfix and prefix increment or decrement operators is the same when used alone in statements, but they behave differently in complex expressions.

Example: Differences Between Postfix and Prefix

```
int i = 1;
int j = ++i;  // Prefix: j is 2, i is 2
int k = i++; // Postfix: k is 2, i is 3
```

Usage: The difference is more noticeable when these operators are used in complex expressions:

```
int i = 10;
int newNum = 10 * i++;  // Postfix: newNum is 100, i becomes 11
newNum = 10 * ++i;      // Prefix: i becomes 12, newNum is 120
```

2.16. Numeric Type Conversions

Binary Operations with Different Types

When a binary operation involves two operands of different types, Java automatically converts the integer operand to a floating-point number.

Example: Binary Operations with Different Types

```
int i = 5;
double d = 2.5;
System.out.println(i + d);  // Displays 7.5
```

Explanation:

- The integer operand `i` is converted to a double before the addition operation.
- The result of the addition is a double.

Widening and Narrowing

- Widening: Assigning a value to a numeric variable that supports a larger range of values.

Example: Widening

```java
double d = 1;  // Assigning an int value to a double variable
```

- Narrowing: Assigning a value to a variable of a type with a smaller range. Narrowing requires explicit casting.

Example: Narrowing

```java
int i = (int) 1.7;  // Assigning a double value to an int variable
```

Explicit Casting

- Syntax for casting: `(targetType) value`

Example: Explicit Casting

```java
System.out.println((int) 1.7);  // Displays 1
System.out.println((double) 1 / 2);  // Displays 0.5
```

- Casting truncates the fractional part when casting from a floating-point number to an integer.

Casting and Assignment

- Casting does not change the original value.

Example: Casting and Assignment

```
double d = 4.5;
int i = (int) d;  // i becomes 4, but d is still 4.5
```

Augmented Assignment and Casting

Example: Augmented Assignment and Casting

```java
int sum = 0;
sum += 4.5;  // sum becomes 4
```

Explanation: This is because `sum += 4.5` is equivalent to `sum = (int) (sum + 4.5)`.

Casting Literals and Variables

Example: Casting a literal

```java
int i = (int) 5.9;  // i becomes 5
```

Explanation: The literal `5.9` is cast to an integer, resulting in `5`.

Compile Error and Casting

A compile error occurs if you try to assign a larger type value to a smaller type without casting.

Example: Compile Error and Casting

```
int i = 1;
byte b = (byte) i;  // Explicit casting is required
```

2.17. Software Development Process

Requirements Specification

- This is the initial stage where the problem to be solved is identified and documented in detail.
- Involves close interaction between users and developers to understand what the software needs to do.

System Analysis

- In this stage, the data flow is analyzed, and the input and output of the system are identified.
- It helps to first identify the output and then determine the necessary input data.

System Design

- This involves designing a process for obtaining the output from the input.
- The problem is broken down into manageable components, and strategies for implementing each component are designed.
- The essence of system analysis and design is input, process, and output (IPO).

Implementation

- Translating the system design into programs.
- Separate programs are written for each component and integrated to work together.
- Includes coding, self-testing, and debugging.

Testing

- Ensures the code meets the requirements specification and identifies any bugs.
- An independent team often conducts testing to verify the product's functionality.

Deployment

- Makes the software available for use.
- Depending on the type of software, it may be installed on users' machines or on a server accessible via the Internet.

Maintenance

- Involves updating and improving the product.
- Software needs to be periodically upgraded to fix bugs and incorporate changes.

2.18. Case Study: Counting Monetary Units

Practice.

2.19. Common Errors and Pitfalls

Undeclared/Uninitialized Variables and Unused Variables

- Variables must be declared before they can be used. If a variable is used without being declared, it results in a compile error.
- Variables must be initialized before they are used. If a variable is used without being initialized, it results in a compile error.
- Unused variables should be removed from the code to improve readability and avoid confusion.
- Java does not allow the use of uninitialized variables.

Example:

- Incorrect:

```java
int x; // Error: x is not initialized
System.out.println(x); // Error: x is not initialized
```

- Correct:

```java
int x = 0; // Initialize x
System.out.println(x); // Use x
```

Integer Overflow

- Numbers are stored with a limited number of digits. When a variable is assigned a value too large for its type, it causes overflow.

Example:

- Incorrect:

```java
int x = 2147483647; // Maximum value for int
x = x + 1; // Overflow: x becomes -2147483648
```

- Correct:

```java
long x = 2147483647; // Use long to avoid overflow
x = x + 1; // No overflow
```

Round-off Errors

- Floating-point numbers have limited precision. Operations on floating-point numbers can result in round-off errors.
- To avoid round-off errors, use integer arithmetic or the `BigDecimal` class for precise calculations.

Example:

- Incorrect:

```java
double result = 1.0 - 0.9 - 0.1; // May not be exactly 0
```

- Correct:

```java
BigDecimal result = new BigDecimal("1.0").subtract(new BigDecimal("0.9")).subtract(new BigDecimal("0.1"));
```

Unintended Integer Division

- Integer division truncates the fractional part. If you want to get a floating-point result, use floating-point numbers in the division.
- To force floating-point division, cast one of the operands to a floating-point type.

Example:

- Incorrect:

```
int x = 5;
int y = 2;
double result = x / y; // Incorrect: result is 2.0, not 2.5
```

- Correct:

```
double result = (double) x / y; // Correct: result is 2.5
```

Redundant Input Objects

- Avoid creating multiple `Scanner` objects for the same input source. It can lead to unexpected behavior and errors.
- Use a single `Scanner` object for each input source and reuse it throughout the program.
- Close the `Scanner` object when you are done with it to release system resources.
- If you close a `Scanner` object, you cannot reopen it.
- If you need to read from the console again, create a new `Scanner` object.

Example:

- Incorrect:

```
Scanner input1 = new Scanner(System.in);
Scanner input2 = new Scanner(System.in);
```

- Correct:

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