Objectives

- To write Java programs to perform simple calculations
- To use identifiers to name variables, constants, methods, and classes
- To use variables to store data
- To program with assignment statements and expressions
- To use constants to store permanent data
- To declare Java primitive data types
- To use Java operators to write expressions
- To represent a string using the String type.
- To obtain input using the JOptionPane input dialog boxes
- To obtain input from console
- To be familiar with Java programming style, and naming conventions
- To distinguish syntax errors, runtime errors, and logic errors
- To debug logic errors
The Basics: Anatomy of a Java Program

What are key components of a Java program?

- Classes
- Comments
- Identifiers and reserved words
- Modifiers
- Statements
- Blocks
- Methods
- The main method

Classes

- The class is the essential Java construct. A class is a template or blueprint for objects.
- To program in Java, you must understand classes and be able to write and use them.
- The mystery of the class will continue to be unveiled throughout this book. For now, though, understand that a program is defined by using one or more classes.
Identifiers and Reserved Words

° Identifiers: a Java word represents variables, constants, class names, method names, parameter/arguments.

° Reserved words or keywords are words that have a specific meaning to the compiler and cannot be used for other purposes (such as variables) in the program.
  • For example, when the compiler sees the word class, it understands that the word after class is the name for the class.
  • Other reserved words in Listing 1.1 are public, static, and void. Their use will be introduced later in the book.
  • Java is case-sensitive, public vs. Public
    - But better avoid using reserved words as identifiers in any forms

Modifiers

° Java uses certain reserved words called modifiers that specify the properties of the data, methods, and classes and how they can be used.

° Examples of modifiers are public and static. Other modifiers are private, final, abstract, and protected. A public data field, method, or class can be accessed by other programs. A private data field or method cannot be accessed by other programs.Modifiers are discussed in Chapter 6, “Objects and Classes.”
Statements

° A statement represents an action or a sequence of actions.

° The statement System.out.println("Welcome to Java!") in the program in Listing 1.1 is a statement to display the greeting "Welcome to Java!"

° Every statement in Java ends with a semicolon (;).

Blocks

° A pair of braces ({} and {}) in a program forms a block that groups components of a program.
Methods

What is System.out.println? It is a method: a collection of statements that performs a sequence of operations to display a message on the console.

Method abstraction: it can be used even without fully understanding the details of how it works.

- Command and query methods

It is used by invoking a statement with a string argument. The string argument is enclosed within parentheses. In this case, the argument is "Welcome to Java!" You can call the same println method with a different argument to print a different message.

```
public class welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```

The main Method

The main method provides the control of program flow. The Java interpreter executes the application by invoking the main method.

- The main method looks like this:

```
public static void main(String[] args) {
    // Statements;
}
```

```
public class welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```
Example: Computing the Area of a Circle

Listing 2.1 Computing the area of a circle of radius 20

```java
/** This program computes the area of the circle */
public class ComputeArea {
    // Main method
    public static void main(String[] args) {
        // declaring variables
        double radius;
        double area;
        // Assign a radius
        radius = 20; // assignment syntax: variable = expression
        // Compute area
        area = radius * radius * 3.14159;
        // Display results
        System.out.println("The area for the circle of radius "+radius+" is "+area);
    }
}
```

Trace a Program Execution

```java
public class ComputeArea {
    /* Main method */
    public static void main(String[] args) {
        double radius; // declaring variable
        double area; // declaring variable
        // Assign a radius (initialization)
        radius = 20;
        // Compute area
        area = radius * radius * 3.14159;
        // Display results
        System.out.println("The area for the circle of radius "+radius+" is "+area);
    }
}
```
Using Constants

**Syntax:** Final datatype CONSTANTNAME = Value;

Listing 2.1 Computing the Area of a Circle - Revised

```java
/** This program computes the area of the circle */
public class ComputeArea {
    // Main method //
    public static void main(String[] args) {
        // declaring a constant for PI; must be declared and initialized at once
        final double PI = 3.1415;
        // declaring and initialize a variable for the radius
        double radius = 20;
        // Compute area
        double area = radius * radius * PI;
        // Display results
        System.out.println("The area for the circle of radius " +
                radius + " is " + area);
    }
}
```

The Basics: Identifiers and Reserved Words

- Identifiers: a word in a Java program that represent
  - A class name
  - A method name
  - A data field
  - A variable name
  - A parameter name
  - * must begin with a letter, underscore _, or the dollar sign $, and consist only of a combination of letters, digits, underscore signs, and dollar signs; of any length (cannot use any reserved words)

- Reserved words: part of Java (cannot be used as identifiers)

- Identifier examples (valid or invalid?)
  - Person1, 1person, counter, PersonID, theIndex, new, true, false, null, final
The Basics: Naming Conventions

° Choose meaningful and descriptive names for identifiers.

° Variables, method, and parameter names:
  • Start with a lowercase letter. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name. E.g., the variables radius and area, and the method computeArea.

° Class names:
  • Capitalize the first letter of each word in the name. For example, the class name ComputeArea.

° Constants:
  • Capitalize all letters in constants, and use underscores to connect words. For example, the constant PI and MAX_VALUE

Please STRICTLY adhere to the conventions!

The Basics: Data Types

° Variables are used to store data in a program.

° Every variable must be declared to be of some specific data type

° Primitive (scalar) type versus Reference (object) type
  • Reference types are always associated with a class, must be constructed (using the object creation operator)
  • Primitive types are NOT associated with a class definition. They do not need to be constructed, the declaration of a primitive type causes the Java system to allocate memory storage for the variable.

All computer program solving has two parts: data and operations!
### Numerical Data Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Storage Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>–2 to 2^7–1 (127)</td>
<td>8-bit signed</td>
</tr>
<tr>
<td>short</td>
<td>–2 to 2^15–1 (32767)</td>
<td>16-bit signed</td>
</tr>
<tr>
<td>int</td>
<td>–2 to 2^31–1 (2147483647)</td>
<td>32-bit signed</td>
</tr>
<tr>
<td>long</td>
<td>–2^63 to 2^63–1</td>
<td>64-bit signed</td>
</tr>
<tr>
<td></td>
<td>(i.e., -9223372036854775808 to 9223372036854775807)</td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>Negative range: -3.4028235E+38 to -1.4E-45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Positive range: 1.4E-45 to 3.4028235E+38</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>Negative range: -1.7976931348623157E+308 to -4.9E-324</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Positive range: 4.9E-324 to 1.7976931348623157E+308</td>
<td></td>
</tr>
</tbody>
</table>

What if numbers are out of the scope?

### Numeric Data Types: Integer Examples

- Numerical literal: a constant value that appears directly in the program
- Assignment operator `=`:
  ```
  int x = 10, y;
  y = x;
  ```
- `int value1, value2 = 17, value3 = -25;` // shorthand form
- `byte smallValue1 = 100;`
- `byte smallValue2 = 1000;` // error, the literal exceeds the maximum
- `long bigValue1, bigValue2 = 123456789L;` // “L” as the last char
  // otherwise, considered to be `int`
- `long bigValue3 = 2147483648;` // error, it exceeds the maximum `int`
- `long bigValue3 = 2147483648L;` // correct
Numeric Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>34 + 1</td>
<td>35</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>34.0 - 0.1</td>
<td>33.9</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>300 * 30</td>
<td>9000</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>1.0 / 2.0</td>
<td>0.5</td>
</tr>
<tr>
<td>%</td>
<td>Remainder</td>
<td>20 % 3</td>
<td>2</td>
</tr>
</tbody>
</table>

Numeric Operations: Examples

- What is the value assigned to \( w \) in the following Java segment?

```
int x = 3, y = 4, z = 5;
int w = x + y * z;
// multiplication has higher precedence than addition
```

- What is the value of \( w \) in the following Java segment?

```
int x = 3, y = 4, z = 5;
int w = x + y / z;
// integer division has higher precedence than addition
```

- What is the value of \( w \) in the following Java segment?

```
int x = 3, y = 4, z = 5;
int w = (x + y) / z;
// the result of integer division is an integer!
```
**Remainder Operator**

Remainder is very useful in programming. For example, an even number % 2 is always 0 and an odd number % 2 is always 1. So you can use this property to determine whether a number is even or odd.

Suppose today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression:

\[
(6 + 10) \mod 7 = 2
\]

After 10 days

---

**Example: Displaying Time**

Write a program that obtains hours and minutes from seconds.

```java
/** Listing 2.2: This program obtains minutes and remaining seconds from given seconds */
import javax.swing.JOptionPane;
public class DisplayTime {
    public static void main(String[] args) {
        int seconds = 500;
        int minutes = seconds / 60;
        int remainingSeconds = seconds % 60;
        JOptionPane.showMessageDialog(null, seconds + " seconds is " + minutes + " minutes and " + remainingSeconds + " seconds");
    }
}
```

How to display the current time in GMT? `System.currentTimeMillis()`
### Numeric Data Types: Floating Point Examples

- A floating-point literal is written with a decimal point.

```java
float f1 = 10.0; // compile error, 10.0 is in double
float f2 = 10.0F; // correct float number
double d1 = 10.0; // correct, higher precision
float f3 = d1; // error, cast needed for conversion
```

- System.out.println(“1.0 / 3.0 is ” + 1.0 / 3.0); what is the output? System.out.println(“1.0F / 3.0F is ” + 1.0F / 3.0F); what is the output?

- Calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy (integer numbers are stored precisely). For example:

  ```java
  System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);
  displays 0.5000000000000001, not 0.5
  ```

### Arithmetic Expressions

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

is translated to

\[
\frac{(3 + 4x)}{5} - \frac{10(y - 5)(a + b + c)}{x} + 9\left(\frac{4}{x} + \frac{9 + x}{y}\right)
\]
Example: Converting Temperatures

Write a program that converts a Fahrenheit degree to Celsius using the formula:

\[ celsius = \left( \frac{5}{9} \right)(fahrenheit - 32) \]

```java
/* Listing 2.3: this program converts F degree to C degree */
public class FahrenheitToCelsius {
    public static void main(String[] args) {
        double fahrenheit = 100; // Say 100;
        double celsius = (5 / 9) * (fahrenheit - 32);
        System.out.println("Fahrenheit " + fahrenheit + " is " + celsius + " in Celsius");
    }
}
```

What could be wrong in this program?

Shortcut Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>i ++</td>
<td>i = i + 1</td>
</tr>
<tr>
<td>--</td>
<td>i --</td>
<td>i = i – 1</td>
</tr>
<tr>
<td>+=</td>
<td>i += 8</td>
<td>i = i + 8</td>
</tr>
<tr>
<td>-=</td>
<td>f -= 8.0</td>
<td>f = f – 8.0</td>
</tr>
<tr>
<td>*=</td>
<td>i *= 8</td>
<td>i = i * 8</td>
</tr>
<tr>
<td>/=</td>
<td>i /= 8</td>
<td>i = i / 8</td>
</tr>
<tr>
<td>%=</td>
<td>i %= 8</td>
<td>i = i % 8</td>
</tr>
</tbody>
</table>

* note: (1) “++” and “--” for int & FP & char variables;
(2) shortcuts are not for any expressions or literals, e.g., (i+j)++, 17++
More Numeric Assignment Examples

Suppose variable `mySalary` has been declared and initialized as follows:

```java
int mySalary = 17;
```

If we need to add 1 to the value of `mySalary`, what we do by `"+"`, `"++"`?

If we wish to subtract 1 from `mySalary`, what we do by `"-"` or `"--"`?

If we wish to add (or subtract) a value other than 1 from `mySalary`. We may use `"+"` or `"+="` (or `"-"` or `"-="`). Examples:

We can use the same shortcut operators for multiplication, division, and remainder. Examples:

```java
mySalary *= 2;
mySalary /= 3;
```

Numeric Type Conversion and Rules

What if we want to mix numeric values of different types in a computation? Consider the following statements:

```java
byte i = 100;
long k = i * 3 + 4;
double d = i * 3.1 + k / 2;
```

When performing a binary operation involving two operands of different types, Java automatically converts the operand based on the following rules:

1. If one of the operands is double, the other is converted into double.
2. Otherwise, if one of the operands is float, the other is converted into float.
3. Otherwise, if one of the operands is long, the other is converted into long.
4. Otherwise, both operands are converted into int.
Type Casting

Implicit casting

```java
double d = 3; // type widening
float f = -34; // correct or incorrect?
```

Explicit casting

```java
int i = (int)3.0; // type narrowing
int i = (int)3.9; // Fraction part is truncated
```

**What is wrong?**

```java
int x = 5 / 2.0;
```

range increases

byte, short, int, long, float, double

---

Example: Keeping Three Digits After Decimal Points

Change the following program to keep three digits after decimal point.

```java
/** Listing 2.4: this program calculates the sales tax with formatted FP output */

public class SalesTax {
    public static void main(String[] args) {
        double purchaseAmount = 197.55;
        double tax = purchaseAmount * 0.076;
        System.out.println((int) (tax * 100) / 100.0);
    }
}
```

Or, we can use a Java class `DecimalFormat` that is in a packet called `text`. 
**Formatting Floating Point Output**

Write a program that displays the sales tax with *three (or more)* digits after the decimal point.

```java
/* this program calculates the sales tax – revised listing 2.4 */
import java.text.*;
    // or full path as java.text.DecimalFormat w/o import
public class SalesTax {
    public static void main(String[] args) {
        double purchaseAmount = 197.55;
        double tax = purchaseAmount * 0.076;
        DecimalFormat df = new DecimalFormat("0.###");
        System.out.println("tax is " + df.format(tax));
    }
}
```

**Character Data Type**

- Type `char` is used to represent an individual character, a letter of the alphabet, and a numeral or a special symbol.

- We use *single quotes* to delimit char literals.
  - `char letter1 = 'A';`
  - `char letter2 = "A";`  // anything wrong?
  - `char letter3 = 'a';`
  - `char numChar = '4';`
  - `char specialLetter = "\n";`

- ASCII: a 7-bit encoding scheme for letters, digits, and control characters

- Java uses a coding scheme for characters called Unicode, which uses 16 bits to do coding – up to 65,536 unique characters

**NOTE:** The increment and decrement operators can also be used on `char` variables to get the next or preceding Unicode character.

  ```java
  char ch = 'a';
  System.out.println(++ch);  // what will be the output?
  ```
**ASCII Character Set**

ASCII Character Set is a subset of the Unicode from \u0000 to \u007f

**TABLE B.2** ASCII Character Set in the Hexadecimal Index

<table>
<thead>
<tr>
<th>Description</th>
<th>Escape Sequence</th>
<th>Unicode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backspace</td>
<td>\b</td>
<td></td>
</tr>
<tr>
<td>Tab</td>
<td>\t</td>
<td></td>
</tr>
<tr>
<td>Linefeed</td>
<td>\n</td>
<td></td>
</tr>
<tr>
<td>Carriage return</td>
<td>\r</td>
<td></td>
</tr>
<tr>
<td>Backslash</td>
<td>\</td>
<td></td>
</tr>
<tr>
<td>Single Quote</td>
<td>'</td>
<td></td>
</tr>
<tr>
<td>Double Quote</td>
<td>&quot;</td>
<td></td>
</tr>
</tbody>
</table>
The String Type

The char type only represents one character. To represent a string of characters, use the data type called String. For example,

```java
String message = "Welcome to Java";
```

String is actually a predefined class in the Java library just like the System class and JOptionPane class.

The String type is not a primitive type. It is known as a reference type. Any Java class can be used as a reference type for a variable. Reference data types will be thoroughly discussed in Chapter 7, “Classes and Objects.”

For the time being, you just need to know how to declare a String variable, how to assign a string to the variable, and how to concatenate strings.

String Concatenation

- Using plus sign (+) for string concatenation
- Three strings are concatenated
  ```java
  String message = "Welcome " + "to " + "Java";
  ```
- String is concatenated with a number
  ```java
  String s = "Chapter" + 2; // what is s then?
  ```
- String Supplement is concatenated with character B
  ```java
  String s = "Supplement" + 'B'; // what is s then?
  ```
- If i = 1 and j = 2, what is the output of the following statements?
  ```java
  System.out.println("i + j is " + i + j);
  System.out.println("i + j is " + (i + j));
  ```
Obtaining Input

This book provides two ways of obtaining input from the user.

1. Using JOptionPane input dialog boxes (§2.11)
2. Using the JDK 1.5 Scanner class, obtaining input from the Console (§2.13)

/** Listing 2.4: this program calculates the sales tax with formatted FP output */

```java
public class SalesTax {
    public static void main(String[] args) {
        double purchaseAmount = 197.55;
        double tax = purchaseAmount * 0.076;
        System.out.println((int) (tax * 100) / 100.0);
    }
}
```

Getting Input from Input Dialog Boxes

String string = JOptionPane.showInputDialog(null, “Prompting Message”, “Dialog Title”, JOptionPane.QUESTION_MESSAGE));

The other way: String string = JOptionPane.showInputDialog(“Prompting Message”);
Converting Strings to Integers/Doubles

The input returned from the input dialog box is a string. If you enter a numeric value such as 123, it returns “123”. To obtain the input as a number, you have to convert a string into a number.

(1) To convert a numeric string into an int value,
   ```java
   int intValue = Integer.parseInt(intString);  // “123”
   ```

(2) To convert a string into a double value
   ```java
   double dValue = Double.parseDouble(doubleString);  // 123.45
   ```

Example: Computing Loan Payments

This program lets the user enter the interest rate, number of years, and loan amount and computes monthly payment and total payment.

\[
\text{loanAmount} \times \frac{\text{monthlyInterestRate}}{1 - \left(1 + \text{monthlyInterestRate}\right)^{\text{numberOfYears} \times 12}}
\]
import javax.swing.JOptionPane;
public class ComputeLoan {
    public static void main(String[] args) {
        /** Main method */
        // Enter yearly interest rate
        String annualInterestRateString = JOptionPane.showInputDialog("Enter yearly interest rate, e.g, 8.25: ");
        double annualInterestRate = Double.parseDouble(annualInterestRateString); // Convert string to double
        double monthlyInterestRate = annualInterestRate / 1200; // Obtain monthly interest rate

        // Enter number of years
        String numberOfYearsString = JOptionPane.showInputDialog("Enter number of years, for example 5: ");
        int numberOfYears = Integer.parseInt(numberOfYearsString); // Convert string to int

        // Enter loan amount
        String loanAmountString = JOptionPane.showInputDialog("Enter loan amount, e.g., 120000.95: ");
        double loanAmount = Double.parseDouble(loanAmountString); // Convert string to double

        // Calculate payment
        double monthlyPayment = loanAmount * monthlyInterestRate / (1 - 1 / Math.pow(1 + monthlyInterestRate,
                        numberOfYears * 12));
        double totalPayment = monthlyPayment * numberOfYears * 12;

        // Format to keep two digits after the decimal point
        monthlyPayment = (int)(monthlyPayment * 100) / 100.0;
        totalPayment = (int)(totalPayment * 100) / 100.0;

        // Display results
        String output = "The monthly payment is " + monthlyPayment + "\n\nThe total payment is " + totalPayment;
        JOptionPane.showMessageDialog(null, output);
    }
}

Way2: Getting Input Using Scanner Class

This is to obtain an input from the console.

1. Create a Scanner object

   Scanner scanner = new Scanner(System.in);

2. Use the methods next(), nextByte(), nextShort(), nextInt(), nextLong(),
   nextFloat(), nextDouble(), or nextBoolean() to obtain to a string (delimited
   by spaces), byte, short, int, long, float, double, or boolean value.

For example,

   System.out.print("Enter a double value: ");
   Scanner scanner = new Scanner(System.in);
   double d = scanner.nextDouble();
Listing 2.7 TestScanner.java

```java
import java.util.Scanner; // Scanner class is in the java.util package

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

        // Prompt the user to enter an integer
        System.out.print("Enter an integer:"); // print() vs println()
        int intValue = scanner.nextInt();
        System.out.println("You entered the integer "+intValue);

        // Prompt the user to enter a double value
        System.out.print("Enter a double value:");
        double doubleValue = scanner.nextDouble();
        System.out.println("You entered the double value "+doubleValue);

        // Prompt the user to enter a string
        System.out.print("Enter a string without space:"); // what if a string containing spaces
        String string = scanner.next(); // what if a string containing spaces
        System.out.println("You entered the string "+string);
    }
}
```

Programming Style and Documentation

- Appropriate comments for readability
- Descriptive names and naming conventions, Strict!
- Proper indentation and spacing lines; why `int i = 3 + 4 * 4;` not good
- Consistent block styles; next-line style vs. end-of-line style

The opening brace is at the beginning of a new line for next-line style, and at the end of the line for end-of-line style.
Programming Errors

° Syntax Errors
  • Occur during compilation, detected by the compiler

° Runtime Errors
  • Causes the program to abort abnormally

° Logic Errors
  • Produces incorrect result

Syntax Errors

```java
public ShowSyntaxErrors {
    public static void main(string[] args) {
        i = 30;
        int k = 100.0;
        System.out.println("i is " + i + " and k is " + k);
    }
}
```

Identify the 4 syntax error(s).
Runtime Errors

```java
public class ShowRuntimeErrors1 {
    public static void Main(String[] args) {
        int i = 30;
        System.out.println(i + 4);
    }
}

public class ShowRuntimeErrors2 {
    public static void main(String[] args) {
        int i = 1 / 0;
    }
}
```

Logic Errors

```java
public class ShowLogicErrors {
    // add two integer numbers: number1 + number2
    public static void main(String[] args) {
        int number1 = 3;
        int number2 = 3;
        number2 += number1 + number2;
        System.out.println("The numbers is " + number2);
    }
}
```

What you expect on the Console?
**Debugging**

- Bugs: logic errors

- Debugging: the process of finding and correcting errors is called debugging. A common approach is to use a combination of methods to narrow down to the part of the program where the bug is located.
  - hand-trace the program vs. use a debugger utility (e.g. NetBeans).

- Debugger is a program that facilitates debugging. You can use a debugger to
  - Execute a single statement at a time.
  - Trace into or stepping over a method.
  - Set breakpoints.
  - Display variables.
  - Display call stack.
  - Modify variables.
  - Example by the use of NetBeans

---

**Reading/Homework**

- Review chapter 2 of the textbook.
- Preview chapter 3 of the textbook.
- Do review questions: 2.3, 2.5, 2.8, 2.10, 2.13, 2.16, 2.24, 2.27 – 2.29
- NetBeans Tutorial and re-do chapter 2 program examples
- Do Programming Exercises (no turn-in): 2.1, 2.2, 2.4; to prepare the lab and homework assignment