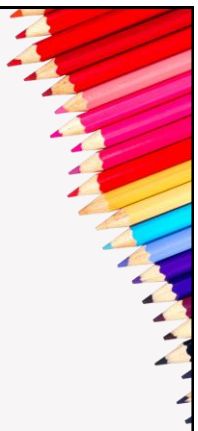
Chapter 2
Elementary Programming

Motivations



- ***The focus of this chapter is on learning elementary programming techniques to solve problems.***
- In the preceding chapter, you learned how to create, compile, and run a Java program.
- Starting from this chapter, you will learn ***how to solve practical problems programmatically*.**
- Through these problems, you will learn Java primitive data types and related subjects, such as variables, constants, data types, operators, expressions, and input and output.

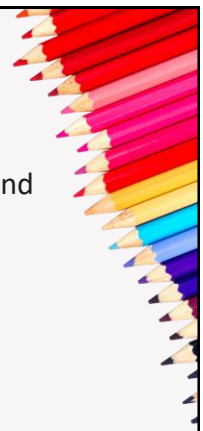
2

Objectives (1)

- To write Java programs to perform simple computations (§2.2).
- To obtain input from the console using the Scanner class (§2.3).
- To use identifiers to name variables, constants, methods, and classes (§2.4).
- To use variables to store data (§§2.5–2.6).
- To program with assignment statements and assignment expressions (§2.6).
- To use constants to store permanent data (§2.7).
- To name classes, methods, variables, and constants by following their naming conventions (§2.8).

3

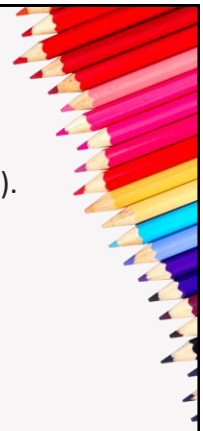
Objectives (2)



- To explore Java numeric primitive data types: byte, short, int, long, float, and double (§2.9).
- To read a byte, short, int, long, float, or double value from the keyboard (§2.9.1).
- To perform operations using operators +, -, *, /, and % (§2.9.2).
- To perform exponent operations using Math.pow(a, b) (§2.9.3).
- To write integer literals, floating-point literals, and literals in scientific notation (§2.10).
- To use JShell to quickly test Java code. (§2.11).
- To write and evaluate numeric expressions (§2.12).

4

Objectives (3)



- To obtain the current system time using System.currentTimeMillis() (§2.13).
- To use augmented assignment operators (§2.14).
- To distinguish between postincrement and preincrement and between postdecrement and predecrement (§2.15).
- To cast the value of one type to another type (§2.16).
- To describe the software development process and apply it to develop the loan payment program (§2.17).
- To write a program that converts a large amount of money into smaller units (§2.18).
- To avoid common errors and pitfalls in elementary programming (§2.19).

5

2.2 Writing a Simple Program

- *Writing a program involves designing a strategy for solving the problem and then using a programming language to implement that strategy.*
- Let's first consider the simple problem of computing the area of a circle.
- How do we write a program for solving this problem?

- Writing a program involves designing algorithms and translating algorithms into programming instructions, or code.
- An *algorithm* describes how a problem is solved by listing the actions that need to be taken and the order of their execution.
- When you *code*—that is, when you write a program—you translate an algorithm into a program.

6

Writing a Simple Program

```
public class ComputeArea {
  public static void main(String[] args) {
    // Step 1: Read in radius

    // Step 2: Compute area

    // Step 3: Display the area
  }
}
```

ComputeArea

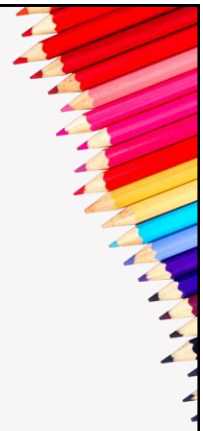
Run

- Two important issues:
  - Reading the radius.
  - Storing the radius in the program.
- Declaring variables
  - *A variable represents a value stored in the computer's memory.*
  - Choose descriptive names for variables and specify their data types.
- Listing 2.1 Computing the Area of a Circle

2.3 Reading Input from the Console

- *Reading input from the console enables the program to accept input from the user.*
- Java uses **System.out** to refer to the standard output device and **System.in** to the standard input device.
- By default, the output device is the display monitor and the input device is the keyboard.
- Console input is not directly supported in Java, but you can use the **Scanner** class to create an object to read input from **System.in.**

Reading Input from the Console



1. import
   - specific import, e.g.

     ```
     import java.util.Scanner;
     ```

   - wildcard import, e.g.

     ```
     import java.util.*;
     ```

2. Create a *Scanner* object

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

3. Use the method *nextDouble()* to obtain to a double value. For example,

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

9

LISTING 2.2 ComputeAreaWithConsoleInput.java

```
1  import java.util.Scanner; // Scanner is in the java.util package
2
3  public class ComputeAreaWithConsoleInput {
4    public static void main(String[] args) {
5      // Create a Scanner object
6      Scanner input = new Scanner(System.in);
7
8      // Prompt the user to enter a radius
9      System.out.print("Enter a number for radius: ");
10     double radius = input.nextDouble();
11
12     // Compute area
13     double area = radius * radius * 3.14159;
14
15     // Display results
16     System.out.println("The area for the circle of radius " +
17       radius + " is " + area);
18   }
19 }
```

ComputeAreaWithConsoleInput

Run

ComputeAverage

Run

2.4 Identifiers

- *Identifiers are the names that identify the elements such as classes, methods, and variables in a program.*
- An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($).
- An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
- An identifier cannot be a reserved word. (See Appendix A, “Java Keywords,” for a list of reserved words).
- An identifier cannot be **`true`**, **`false`**, or **`null`**.
- An identifier can be of any length.

11

2.5 Variables

- *Variables are used to represent values that may be changed in the program.*

```
// Compute the first area
radius = 1.0;
area = radius * radius * 3.14159;
System.out.println("The area is " + area + " for radius "+radius);

// Compute the second area
radius = 2.0;
area = radius * radius * 3.14159;
System.out.println("The area is " + area + " for radius "+radius);
```

12

Declaring Variables

- Variables are for representing data of a certain type.
- The *variable declaration* tells the compiler to allocate appropriate memory space for the variable based on its data type.
- The syntax for declaring a variable is

```
datatype variableName;
```

```
int count;           // Declare count to be an integer variable
double radius;       // Declare radius to be a double variable
double interestRate; // Declare interestRate to be a double variable
```

- If variables are of the same type, they can be declared together

```
datatype variable1, variable2, ..., variablen;
```

```
int i, j, k; // Declare i, j, and k as int variables
```

13

Assignment Statements

- A variable declared in a method must be assigned a value before it can be used.

```
x = 1;
      // Assign 1 to x;
radius = 1.0;
      // Assign 1.0 to radius;
a = 'A';
      // Assign 'A' to a;
```

Initializing Variables

- You can declare a variable and initialize it in one step.
- Whenever possible, declare a variable and assign its initial value in one step. This will make the program easy to read and avoid programming errors.

```
double d = 1.4;
int count = 1;
// equivalent to the next 2 statements:
int count;
count = 1;
//use a shorthand form to declare and initialize variables of the same type together:
int i = 1, j = 2;
```

2.6 Assignment Statements and Assignment Expressions

- *An assignment statement designates a value for a variable.* The syntax for assignment statements is as follows:

  variable = expression;

  - An expression represents a computation involving values, variables, and operators that, taking them together, evaluates to a value.
  - The data type of the variable on the left must be compatible with the data type of the value on the right.

- *An assignment statement can be used as an expression in Java.*

```
int y = 1;
double radius = 1.0;
int x = 5 * (3 / 2);
x = y + 1;
double area = radius * radius * 3.14159;
```

```
System.out.println(x = 1);
```

2.7 Named Constants

- *A named constant is an identifier that represents a permanent value.*
- Here is the syntax for declaring a constant:

```
final datatype CONSTANTNAME = VALUE;

final double PI = 3.14159;
final int SIZE = 3;
```

ComputeAreaWithConstant

17

2.8 Naming Conventions

- Choose meaningful and descriptive names.
- Variables and method names:
  - Use lowercase. 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.
  - For example, 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`**

18

2.9 Numeric Data Types and Operations

- *Java has six numeric types for integers and floating-point numbers with operators **+, -, *, /, and %.***
- Every data type has a range of values.
- The compiler allocates memory space for each variable or constant according to its data type.
- Java provides eight primitive data types for *numeric values*, *characters*, and *Boolean values*.
- Java uses four types for integers: **byte, short, int, and long.**
- Java uses two types for floating-point numbers: **float and double.**

19

Numerical Data Types

Name	*Range*	*Storage Size*
`byte`	-2^7 to $2^7 - 1$ (−128 to 127)	8-bit signed
`short`	-2^{15} to $2^{15} - 1$ (−32768 to 32767)	16-bit signed
`int`	-2^{31} to $2^{31} - 1$ (−2147483648 to 2147483647)	32-bit signed
`long`	-2^{63} to $2^{63} - 1$	64-bit signed
	(i.e., −9223372036854775808 to 9223372036854775807)	
`float`	Negative range: −3.4028235E + 38 to −1.4E − 45	32-bit IEEE 754
	Positive range: 1.4E − 45 to 3.4028235E + 38	
`double`	Negative range: −1.7976931348623157E + 308 to −4.9E − 324	64-bit IEEE 754
	Positive range: 4.9E − 324 to 1.7976931348623157E + 308	

20

Reading Numbers from the Keyboard

```
Scanner input = new Scanner(System.in);
int value = input.nextInt();
```

Method	*Description*
`nextByte()`	reads an integer of the `byte` type.
`nextShort()`	reads an integer of the `short` type.
`nextInt()`	reads an integer of the `int` type.
`nextLong()`	reads an integer of the `long` type.
`nextFloat()`	reads a number of the `float` type.
`nextDouble()`	reads a number of the `double` type.

21

Numeric Operators

Name	*Meaning*	*Example*	*Result*
`+`	Addition	34 + 1	35
`-`	Subtraction	34.0 – 0.1	33.9
`*`	Multiplication	300 * 30	9000
`/`	Division	1.0 / 2.0	0.5
`%`	Remainder	20 % 3	2

22

Problem: Displaying Time

- Write a program that obtains minutes and remaining seconds from seconds.

LISTING 2.5 DisplayTime.java

```
1  import java.util.Scanner;
2
3  public class DisplayTime {
4    public static void main(String[] args) {
5      Scanner input = new Scanner(System.in);
6      // Prompt the user for input
7      System.out.print("Enter an integer for seconds: ");
8      int seconds = input.nextInt();
9
10     int minutes = seconds / 60; // Find minutes in seconds
11     int remainingSeconds = seconds % 60; // Seconds remaining
12     System.out.println(seconds + " seconds is " + minutes +
13       " minutes and " + remainingSeconds + " seconds");
14   }
15 }
```

DisplayTime

Run

23

Exponent Operations

- The Math.pow(a, b) method can be used to compute a^b.
- The *pow* method is defined in the *Math* class in the Java API.
- You invoke the method using the syntax *Math.pow(a, b)*, which returns the result of a^b .

```
System.out.println(Math.pow(2, 3)); // Displays 8.0
System.out.println(Math.pow(4, 0.5)); // Displays 2.0
System.out.println(Math.pow(2.5, 2)); // Displays 6.25
System.out.println(Math.pow(2.5, -2)); // Displays 0.16
```

24

2.10 Numeric Literals

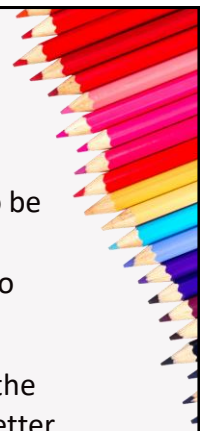
- A literal is a constant value that appears directly in the program.
  - For example, 34, 1,000,000, and 5.0 are literals in the following statements:

```
int  i = 34;
long x = 1000000;
double d = 5.0;
```

25

Integer Literals

- An integer literal can be assigned to an integer variable as long as it can fit into the variable.
- A compilation error would occur if the literal were too large for the variable to hold.
  - For example, the statement byte b = 1000 would cause a compilation error, because 1000 cannot be stored in a variable of the byte type.
- An integer literal is assumed to be of the int type, whose value is between -231 (-2147483648) to 231–1 (2147483647).
- To denote an integer literal of the long type, append it with the letter L or l.
  - L is preferred because l (lowercase L) can easily be confused with 1 (the digit one).



26

Floating-Point Literals

- Floating-point literals are written with a decimal point.
- By default, a floating-point literal is treated as a double type value.
  - For example, 5.0 is considered a double value, not a float value.
- You can make a number a float by appending the letter **f** or **F**, and make a number a double by appending the letter **d** or **D**.
  - For example, you can use 100.2f or 100.2F for a float number, and 100.2d or 100.2D for a double number.
- Floating-point literals can also be specified in **scientific notation**.
  - For example, 1.23456e+2, same as 1.23456e2, is equivalent to 123.456, and 1.23456e-2 is equivalent to 0.0123456.

27

Floating-Point Literals

- **NOTE**
- To improve readability, Java allows you to use underscores to separate two digits in a number literal.
- For example, the following literals are correct.

```
long value = 232_45_4519;
double amount = 23.24_4545_4519_3415;
```

- However, **45_** or **_45** is incorrect. The underscore must be placed between two digits.

double vs. float

- The double type values are more accurate than the float type values. For example,

```
System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);
```

displays `1.0 / 3.0 is 0.3333333333333333`

16 digits

```
System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);
```

displays `1.0F / 3.0F is 0.33333334`

7 digits

29

2.11 JShell (JDK 9 or higher)

- *JShell is a command line tool for quickly evaluating an expression and executing a statement.*
- JShell enables you to type a single Java statement and get it executed to see the result right away without having to write a complete class.
- This feature is commonly known as REPL (Read-Evaluate-Print Loop), which evaluates expressions and executes statements as they are entered and shows the result immediately.
- Open a Command Window and type **jshell** to launch JShell
  - /help, /var, /edit, /exit
  - A variable will be automatically created for a value.

2.12 Evaluating Expressions and Operator Precedence

- Arithmetic Expressions

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

can be translated into a Java expression as:

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

31

How to Evaluate an Expression

- The result of a Java expression and its corresponding arithmetic expression are the same.
- Therefore, you can safely apply the arithmetic rule for evaluating a Java expression.

```
3 + 4 * 4 + 5 * (4 + 3) - 1
                (1) inside parentheses first
3 + 4 * 4 + 5 * 7 - 1
            (2) multiplication
3 + 16 + 5 * 7 - 1
                (3) multiplication
3 + 16 + 35 - 1
            (4) addition
19 + 35 - 1
        (5) addition
54 - 1
    (6) subtraction
53
```

32

Problem: Converting Temperatures

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

$$celsius = (\frac{5}{9})(fahrenheit - 32)$$

- Note: you have to write

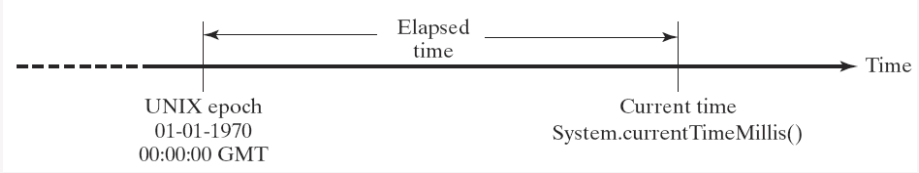
```
celsius = (5.0 / 9) * (fahrenheit – 32)
```

FahrenheitToCelsius Run

33

Problem: Displaying Current Time

- Write a program that displays current time in GMT in the format hour:minute:second such as 1:45:19.
- The *currentTimeMillis()* method in the *System* class returns the current time in milliseconds since the midnight, January 1, 1970 GMT. (1970 was the year when the Unix operating system was formally introduced.) You can use this method to obtain the current time, and then compute the current second, minute, and hour as follows.



ShowCurrentTime Run

34

2.14 Augmented Assignment Operators

Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Division assignment	`i /= 8`	`i = i / 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`

35

2.15 Increment and Decrement Operators

Operator	*Name*	*Description*	*Example (assume i = 1)*
`++var`	preincrement	Increment `var` by `1`, and use the new `var` value in the statement	`int j = ++i;` `// j is 2, i is 2`
`var++`	postincrement	Increment `var` by `1`, but use the original `var` value in the statement	`int j = i++;` `// j is 1, i is 2`
`--var`	predecrement	Decrement `var` by `1`, and use the new `var` value in the statement	`int j = --i;` `// j is 0, i is 0`
`var--`	postdecrement	Decrement `var` by `1`, and use the original `var` value in the statement	`int j = i--;` `// j is 1, i is 0`

36

Increment and Decrement Operators, cont.

- Using increment and decrement operators makes expressions short, but it also makes them complex and difficult to read.
- Avoid using these operators in expressions that modify multiple variables, or the same variable for multiple times such as this:

```
int k = ++i + i
```

Expressions and Statements

- Prior to Java 2, all the expressions can be used as statements.
- Since Java 2, only the following types of expressions can be statements:

```
variable op= expression;
   // Where op is +, -, *, /, or %
++variable;
variable++;
--variable;
variable--;
```

2.16 Numeric Type Conversions

- Can you perform binary operations with two operands of different types?
  - Yes.
  - If an integer and a floating-point number are involved in a binary operation, Java automatically converts the integer to a floating-point value.
- You can always assign a value to a numeric variable whose type supports a larger range of values; You cannot, however, assign a value to a variable of a type with a smaller range unless you use *type casting*.

39

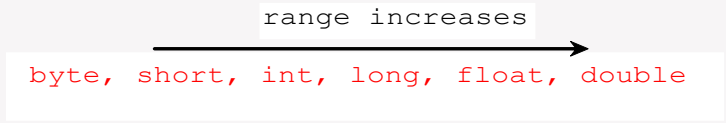
Conversion Rules

- 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.

40

Type Casting

- *Casting is* an operation that converts a value of one data type into a value of another data type.
- Casting a type with a small range to a type with a larger range is known as *widening a type.*
- *Casting a type with* a large range to a type with a smaller range is known as *narrowing a type.*
- Java will ***automatically*** widen a type, but you must narrow a type ***explicitly***.



41

Type Casting

- Implicit casting

```
double d = 3; (type widening)
```

- Explicit casting

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

- What is wrong?

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

42

Casting in an Augmented Expression

- In Java, an augmented expression of the form **`x1 op= x2`** is implemented as **`x1 = (T)(x1 op x2)`**, where T is the type for x1.
- Therefore, the following code is correct.

```
int sum = 0;
sum += 4.5; // sum becomes 4 after this statement
```

`sum += 4.5` is equivalent to

`sum = (int)(sum + 4.5)`.

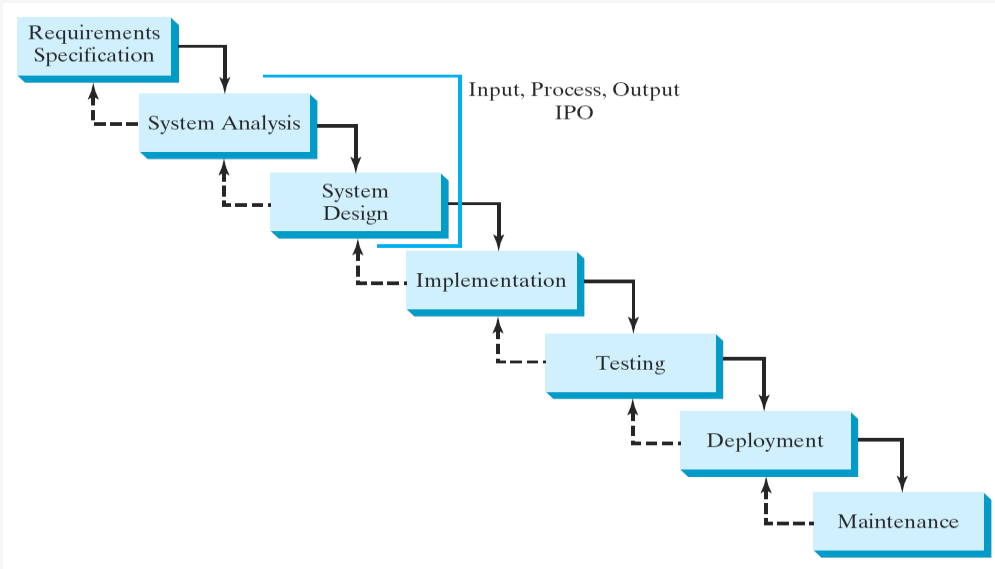
Problem: Keeping Two Digits After Decimal Points

- Write a program that displays the sales tax with two digits after the decimal point.

```
1  import java.util.Scanner;
2
3  public class SalesTax {
4    public static void main(String[] args) {
5      Scanner input = new Scanner(System.in);
6
7      System.out.print("Enter purchase amount: ");
8      double purchaseAmount = input.nextDouble();
9
10     double tax = purchaseAmount * 0.06;
11     System.out.println("Sales tax is $" + (int)(tax * 100) / 100.0);
12   }
13 }
```

SalesTax Run

2.17 Software Development Process



Problem: 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.

$$monthlyPayment = \frac{loanAmount \times monthlyInterestRate}{1 - \frac{1}{(1 + monthlyInterestRate)^{numberOfYears \times 12}}}$$

$$totalPayment = monthlyPayment \times numberOfYears \times 12$$

ComputeLoan Run

46

Problem: Counting Monetary Units

- This program lets the user enter the amount in decimal representing dollars and cents and output a report listing the monetary equivalent in single dollars, quarters, dimes, nickels, and pennies. Your program should report maximum number of dollars, then the maximum number of quarters, and so on, in this order.

ComputeChange

Run

47

2.19 Common Errors and Pitfalls

- Common Error 1: Undeclared/Uninitialized Variables and Unused Variables
- Common Error 2: Integer Overflow
- Common Error 3: Round-off Errors
- Common Error 4: Unintended Integer Division
- Common Pitfall 1: Redundant Input Objects

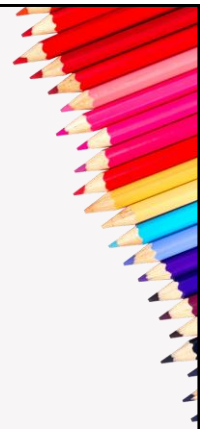
48

Common Error 1: Undeclared/Uninitialized Variables and Unused Variables

- A variable must be declared with a type and assigned a value before using it.
- A common error is not declaring a variable or initializing a variable.

```
double interestRate = 0.05;

double interest = interestrate * 45;
```



49

Common Error 2: Integer Overflow

- Numbers are stored with a limited numbers of digits. When a variable is assigned a value that is too large (*in size)* to be stored, it causes *overflow*.

```
int value = 2147483647 + 1;

// value will actually be -2147483648
```

- Java does not report warnings or errors on overflow, so be careful when working with numbers close to the maximum or minimum range of a given type.

50

Common Error 3: Round-off Errors

- A *round-off error, also called a rounding error, is the difference between the calculated* approximation of a number and its exact mathematical value.
  - For example, 1/3 is approximately 0.333 if you keep three decimal places, and is 0.3333333 if you keep seven decimal places.
- Since the number of digits that can be stored in a variable is limited, round-off errorsare inevitable.
- Calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy.

```
System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);
System.out.println(1.0 - 0.9);
```

51

Common Error 4: Unintended Integer Division

```
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2;
System.out.println(average);
```

(a)

```
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2.0;
System.out.println(average);
```

(b)

52

Common Pitfall 1: Redundant Input Objects

BAD CODE

```
Scanner input = new Scanner(System.in);
System.out.print("Enter an integer: ");
int v1 = input.nextInt();

Scanner input1 = new Scanner(System.in);
System.out.print("Enter a double value: ");
double v2 = input1.nextDouble();
```

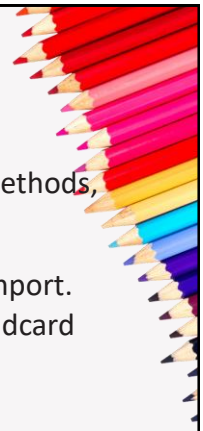
GOOD CODE

```
Scanner input = new Scanner(System.in);
System.out.print("Enter an integer: ");
int v1 = input.nextInt();
System.out.print("Enter a double value: ");
double v2 = input.nextDouble();
```

53

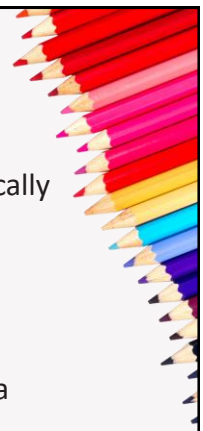
Chapter Summary

Chapter Summary



- **Identifiers** are names for naming elements such as variables, constants, methods, classes, packages in a program.
- There are two types of ***import statements***: specific import and wildcard import. The specific import specifies a single class in the import statement; the wildcard import imports all the classes in a package.
- A named constant is declared by using the keyword ***final***.
- Java provides four integer types (**byte, short, int, and long**) that represent integers of four different sizes.
- Java provides two floating-point types (**float and double**) that represent floating-point numbers of two different precisions.

Chapter Summary



- When evaluating an expression with values of mixed types, Java automatically converts the operands to appropriate types.
- You can explicitly convert a value from one type to another using the *(type)value* notation.
- Casting a variable of a type with a small range to a variable of a type with a larger range is known as widening a type.
- Casting a variable of a type with a large range to a variable of a type with a smaller range is known as narrowing a type.
- Widening a type can be performed automatically without explicit casting. Narrowing a type must be performed explicitly.

Programming Exercises

1, 2, 3, 4, 5, 6, 7, 8, 12, 13, 14, 15, 18, 19, 20, 21, 23