

Chapter 2 Elementary Programming



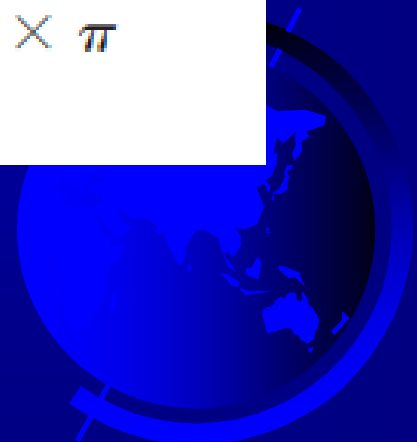
Introducing with an Example

Computing the Area of a Circle *Algorithm*.

1. Read in the radius.
2. Compute the area using the following formula:

$$area = radius \times radius \times \pi$$

3. Display the area.



Trace a Program Execution

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

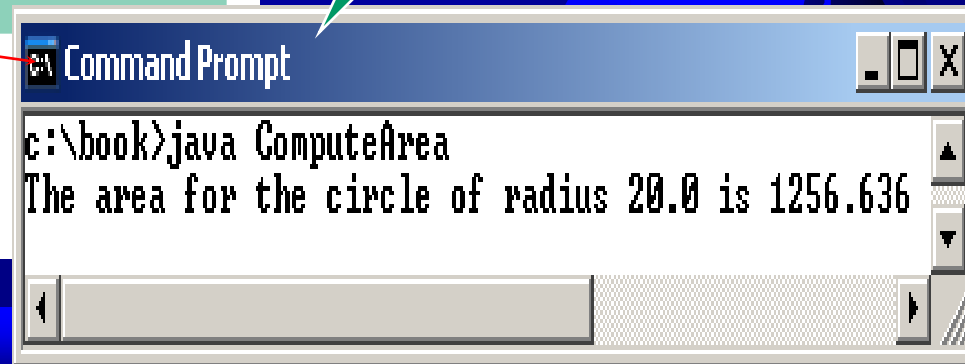
memory

System class:

- predefined in java.lang package,
implicitly imported

System.out:

- standard output device
- Simply use the println method to
display a value to the console





Caution

A string constant cannot cross lines in the source code. Thus the following statement would result in a compile error:

```
System.out.println("Introduction to Java Programming,  
by Y. Daniel Liang");
```

To fix the error, break the string into separate substrings, and use the concatenation operator (+) to combine them:

```
System.out.println("Introduction to Java Programming, "  
"by Y. Daniel Liang");
```

Reading Input from the Console

LISTING 2.2 ComputeAreaWithConsoleInput.java

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

Console input

- not directly supported in Java
- but use the Scanner class to create an object to read input from keyboard

Enter a number for radius: 2.5
The area for the circle of radius

LISTING 2.3 ComputeAverage.java

import class

```
1 import java.util.Scanner; // Scanner is in the java.util package
2
3 public class ComputeAverage {
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 three numbers
9         System.out.print("Enter three numbers: ");
10        double number1 = input.nextDouble();
11        double number2 = input.nextDouble();
12        double number3 = input.nextDouble();
13
14        // Compute average
15        double average = (number1 + number2 + number3) / 3;
16
17        // Display result
18        System.out.println("The average of " + number1 + " " + number2
19            + " " + number3 + " is " + average);
20    }
21 }
```

create a Scanner

read a double

enter input in
one line



```
Enter three numbers: 1 2 3 ↵ Enter
The average of 1.0 2.0 3.0 is 2.0
```

enter input in
multiple lines



```
Enter three numbers: 10.5 ↵ Enter
11 ↵ Enter
11.5 ↵ Enter
The average of 10.5 11.0 11.5 is 11.0
```

Identifiers

a sequence of characters

- consist of letters, digits, underscores (_), and dollar signs (\$).
- start with a letter, an underscore (_), or a dollar sign (\$). It cannot start with a digit.

cannot be

- a reserved word.
- true, false, or null

can be of any length.

Valid identifiers?

- \$2, ComputeArea, radius, class, showMessageDialog, 2A, d+4

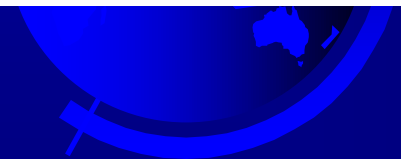


Variables

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

Note

By convention, variable names are in lowercase.



Declaring and Initializing in One Step

```
int x = 1;
```

```
- int x = 1;
```

```
- int x = 1;
```

```
double d = 1.4;
```

```
char a = 'A';
```

```
...
```



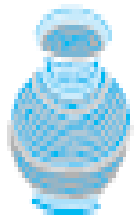
Constants

Must be declared and initialized.

```
final datatype CONSTANTNAME = VALUE;
```

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

```
final int SIZE = 3;
```



Caution

By convention, constants are named in uppercase: **PI**, not **pi** or **Pi**.

Numerical Data Types

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

NOTE

Calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy.

- `System.out.println (1.0 - 0.9);`
displays 0.099999999999999998, not 0.1.
- `System.out.println (1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);`
displays 0.500000000000000001, not 0.5,

Integers are stored precisely. Therefore, calculations with integers yield a precise integer result.



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 取余	20 % 3	2

Integer Division

$5 / 2$ yields ?

$5.0 / 2$ yields ?

$5 \% 2$ yields 1

Remainder is very useful in programming.

One 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.



Problem: Displaying Time

Write a program that obtains minutes and remaining seconds from an amount of time in seconds.

LISTING DisplayTime.java

import Scanner
create a Scanner

```
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 }
```

read an integer

divide
remainder



Enter an integer for seconds: 500
500 seconds is 8 minutes and 20 seconds

Number Literals

A *literal* is a constant value that appears directly in the program.

```
int i = 34;
```

```
long x = 1000000;
```

```
double d = 5.0;
```



Scientific Notation

Floating-point literals can also be specified in scientific notation

for example, 1.234e+2, same as 1.234e2,

is equivalent to 123.4

1.234e-2 is equivalent to 0.01234.

E (or e) represents an exponent and it can be either in lowercase or uppercase.



Problem: Converting Temperatures

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

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

LISTING FahrenheitToCelsius.java

```
1 import java.util.Scanner;
2
3 public class FahrenheitToCelsius {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6
7         System.out.print("Enter a degree in Fahrenheit: ");
8         double fahrenheit = input.nextDouble();
9
10        // Convert Fahrenheit to Celsius
11        double celsius = (5.0 / 9) * (fahrenheit - 32);
12        System.out.println("Fahrenheit " + fahrenheit + " is " +
13            celsius + " in Celsius");
14    }
15 }
```

divide

Enter a degree in Fahrenheit: 100
Fahrenheit 100.0 is 37.77777777777778 in Celsius



Shorthand Operators

TABLE 1 Shorthand Operators

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

Increment and Decrement Operators

TABLE Increment and Decrement Operators

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

Increment and Decrement Operators, cont.

```
int i = 10;  
int newNum = 10 * i++;
```

```
int i = 10;  
int newNum = 10 * (++i);
```



Increment and Decrement Operators, cont.

```
int i = 10;
```

```
int newNum = 10 * i++;
```

Same effect as

```
int newNum = 10 * i;  
i = i + 1;
```

```
int i = 10;
```

```
int newNum = 10 * (++i);
```

Same effect as

```
i = i + 1;  
int newNum = 10 * i;
```

Numeric Type Conversion

Consider the following statements:

```
int i = 100;
```

```
long k = i * 3 + 4;
```

```
double d = i * 3.1 + k / 2;
```



Type Casting

Implicit casting

```
double d = 3; (by default, 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;`

range increases

byte, short, int, long, float, double

Keeping Two Digits After Decimal Points

$$(197.55 * 0.06 = 11.853)$$

LISTING SalesTax.java

```
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 }
```

Enter purchase amount: 197.55
Sales tax is 11.85

tax * 100 is 1185.3
(int)(tax * 100) is 1185
(int)(tax * 100) / 100.0 is 11.85

$$tax: 197.55 * 0.06 = \underline{11.853}$$

Computing Loan Payments

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

```
monthlyInterestRate = yearlyInterestRate / 12;  
totalPayment = monthlyPayment * numberOfYears * 12;
```

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

`pow(a, b)` method in the `Math` class can be used to compute a^b .

```

enter loan amount    22    double loanAmount = input.nextDouble();
                    23
                    24    // Calculate payment
monthlyPayment      25    double monthlyPayment = loanAmount * monthlyInterestRate / (1
                    26    - 1 / Math.pow(1 + monthlyInterestRate, numberOfYears * 12));
totalPayment        27    double totalPayment = monthlyPayment * numberOfYears * 12;
                    28

```

The Math class is in the **java.lang** package.

All classes in the java.lang package are implicitly imported.

```

35 }

```



Enter yearly interest rate, for example 8.25: 5.75

Enter number of years as an integer, for example 5: 15

Enter loan amount, for example 120000.95: 250000

The monthly payment is 2076.02

The total payment is 373684.53

Programming Style and Documentation

Appropriate Comments

Naming Conventions

Proper Indentation and Spacing
Lines

Block Styles



Appropriate Comments

Include a summary at the beginning of the program to explain what the program does, its key features, its supporting data structures, and any unique techniques it uses.

- Include your name, class section, instructor, date, and a brief description of the program.

line comments (//): **inside a method**

block comment /*...*/

javadoc comments (/** ... */): **entire class/method.**



Naming Conventions

Choose meaningful and descriptive names.

Variables and method names:

- Use lowercase.
- If the name consists of several words, use lowercase for the first word, and capitalize the first letter of each subsequent word.

For example, method name:

computeArea



Naming Conventions, cont.

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,

MAX_VALUE



Proper Indentation and Spacing

Indentation

Spacing

- Use whitespace

```
int i= 3+4 * 4;
```

← Bad style

```
int i = 3 + 4 * 4;
```

← Good style

- Use blank line to separate segments of the code.



Block Styles

End-of-line style

- Save space, help avoid some subtle errors
- This book, Java API source code

Next-line style

- Align braces vertically, make program easy to read

Both styles are acceptable

Mixing style is not recommended

*Next-line
style*

```
public class Test
{
    public static void main(String[] args)
    {
        System.out.println("Block Styles");
    }
}
```

*End-of-line
style*

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Block Styles");
    }
}
```

JOptionPane Input

Two ways of obtaining input.

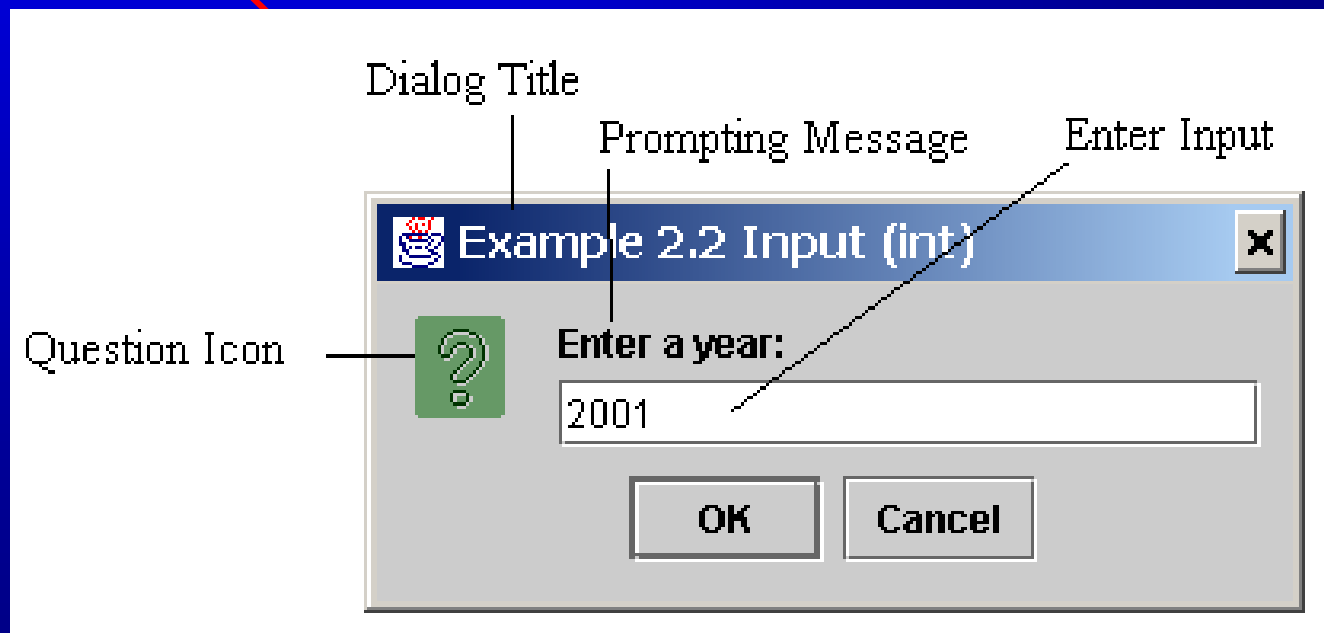
1. Using the **Scanner** class (console input)
2. Using **JOptionPane** (input dialogs), GUI



Getting Input from Input Dialog Boxes

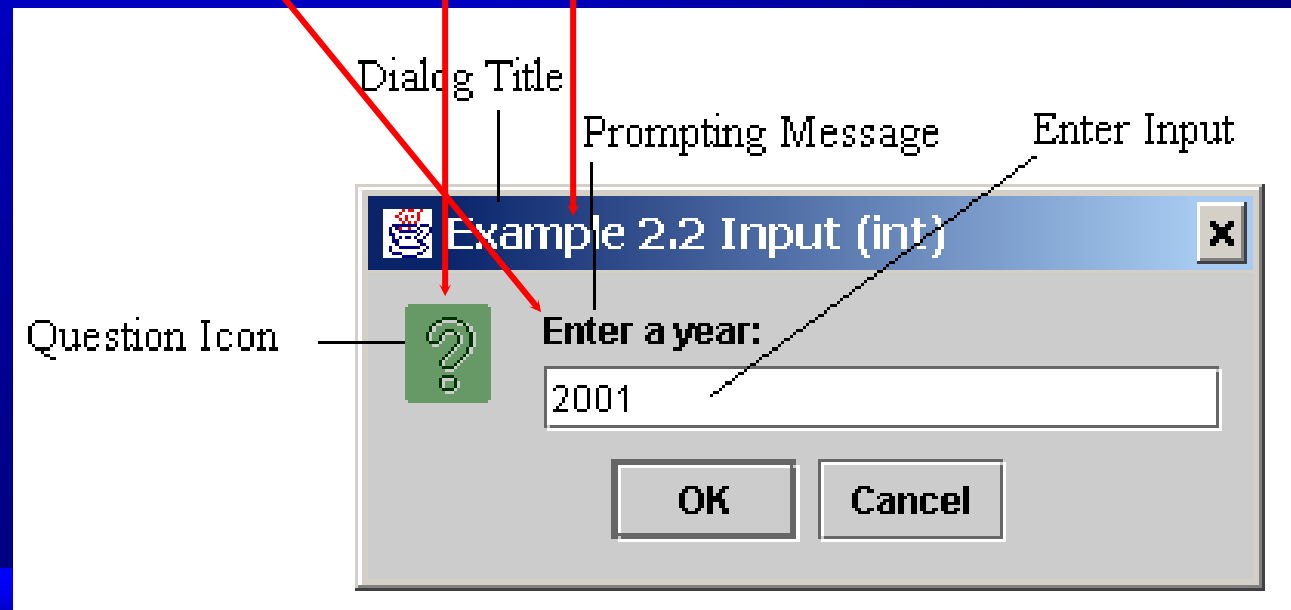
```
import javax.swing.*;
```

```
String input = JOptionPane.showInputDialog(  
    "Enter a year:");
```



Getting Input from Input Dialog Boxes

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



Conclusion: two Ways to Invoke the
JOptionPane.showInputDialog() Method :

1. **String string =
JOptionPane.showInputDialog(null, x,
y, JOptionPane.QUESTION_MESSAGE);**

where x is a string for the prompting message, and y is a string for the title of the input dialog box.

2. **JOptionPane.showInputDialog(x);**

where x is a string for the prompting message.



Converting Strings to Integers

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.

```
int intValue = Integer.parseInt(intString);
```



Converting Strings to Doubles

To convert a string into a double value,

double doubleValue = Double.parseDouble(doubleString);

– where doubleString is a numeric string such as “123.45”.

The `Integer` and `Double` classes are both included in the `java.lang` package, and thus they are automatically imported.

Problem: Computing Loan Payments Using Input Dialogs

the input is entered from the input dialogs and the output is displayed in an message/output dialog.

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

Programming Errors

Syntax Errors (compile error)

- Detected by the compiler

Runtime Errors

- Causes the program to abort

Logic Errors

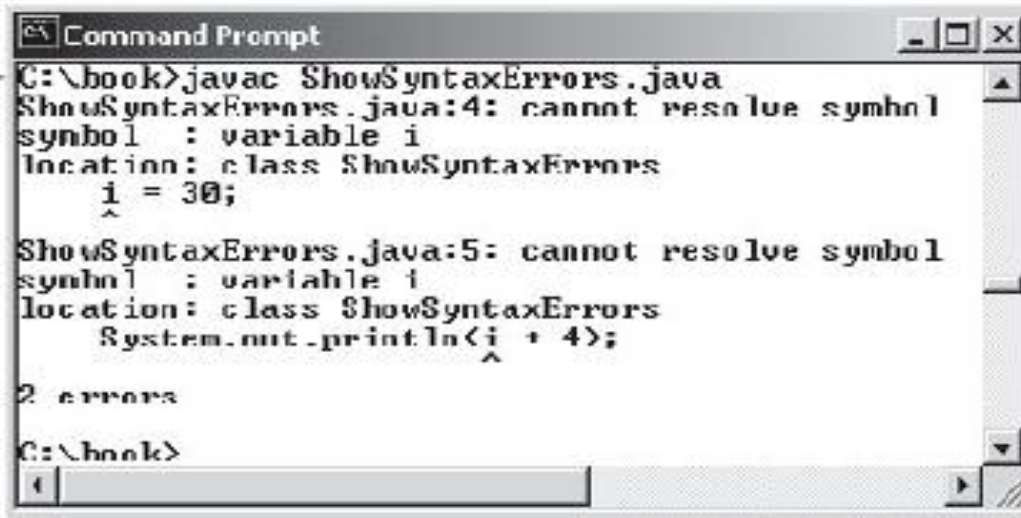
- Produces incorrect result



Syntax Errors

```
public class ShowSyntaxErrors {  
    public static void main(String[] args) {  
        i = 30;  
        System.out.println(i + 4);  
    }  
}
```

Compile →



```
Command Prompt  
C:\book>javac ShowSyntaxErrors.java  
ShowSyntaxErrors.java:4: cannot resolve symbol  
symbol : variable i  
location: class ShowSyntaxErrors  
    i = 30;  
    ^  
ShowSyntaxErrors.java:5: cannot resolve symbol  
symbol : variable i  
location: class ShowSyntaxErrors  
    System.out.println(i + 4);  
                        ^  
2 errors  
C:\book>
```



Two errors are detected. Both are the result of not declaring variable i.

Runtime Errors

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

Run →



The screenshot shows a Windows Command Prompt window titled "Command Prompt". The prompt is at "C:\book>". The user has entered the command "java ShowRuntimeErrors". The output shows an exception: "Exception in thread "main" java.lang.ArithmeticException: / by zero" followed by "at ShowRuntimeErrors.main(ShowRuntimeErrors.java:4)". The prompt is now "C:\book>".

```
Command Prompt  
C:\book>java ShowRuntimeErrors  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
    at ShowRuntimeErrors.main(ShowRuntimeErrors.java:4)  
C:\book>
```

runtime errors is division by zero.

Logic Errors

Logic errors occur when a program does not perform the way it was intended to. Errors of this kind occur for many different reasons. For example, suppose you wrote the following program to add `number1` to `number2`.

```
// ShowLogicErrors.java: The program contains a logic error
public class ShowLogicErrors {
    public static void main(String[] args) {
        // Add number1 to number2
        int number1 = 3;
        int number2 = 3;
        number2 += number1 + number2;
        System.out.println("number2 is " + number2);
    }
}
```

Debugging

Logic errors are called *bugs*.

The process of finding and correcting errors is called *debugging*

A common approach to debugging is to use a combination of methods to *narrow down to the part* of the program *where the bug is located*.

- You can hand-trace the program (i.e., catch errors by reading the program)
- or you can insert print statements in order to show the values of the variables or the execution flow of the program. This approach might work for a short, simple program.
- But for a large, complex program, the most effective approach for debugging is to use a debugger utility.



Debugger

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.

