Programming Techniques-JAVA BCNS1102C

Lecture 4-5: Conditional Expression

Lecturer: Mr. Ajit. Gopee

E-mail: ajit.gopee@utm.ac.mu

OBJECTIVES

In this chapter you'll learn:

- Basic problem-solving techniques.
- To develop algorithms through the process of top-down, stepwise refinement.
- To use the `if` and `if`...`else` selection statements to choose among alternative actions.
- To use the `while` repetition statement to execute statements in a program repeatedly.
- To use counter-controlled repetition and sentinel-controlled repetition.
- To use the compound assignment, increment and decrement operators.
- The portability of primitive data types.

Control structures

- Sequential execution: Statements in a program execute one after the other in the order in which they are written.
- Transfer of control: Various Java statements, enable you to specify that the next statement to execute is not necessarily the next one in sequence.
- Bohm and Jacopini
  - Demonstrated that programs could be written *without any `goto` statements.*
  - All programs can be written in terms of only three control structures—the sequence structure, the selection structure and the repetition structure.
- When we introduce Java's control structure implementations, we'll refer to them in the terminology of the *Java Language Specification as "control statements."*

4.4 Control Structures (Cont.)

- Sequence structure
  - Built into Java.
  - Unless directed otherwise, the computer executes Java statements one after the other in the order in which they’re written.
  - The activity diagram in Fig. 4.1 illustrates a typical sequence structure in which two calculations are performed in order.
  - Java lets you have as many actions as you want in a sequence structure.
  - Anywhere a single action may be placed, we may place several actions in sequence.

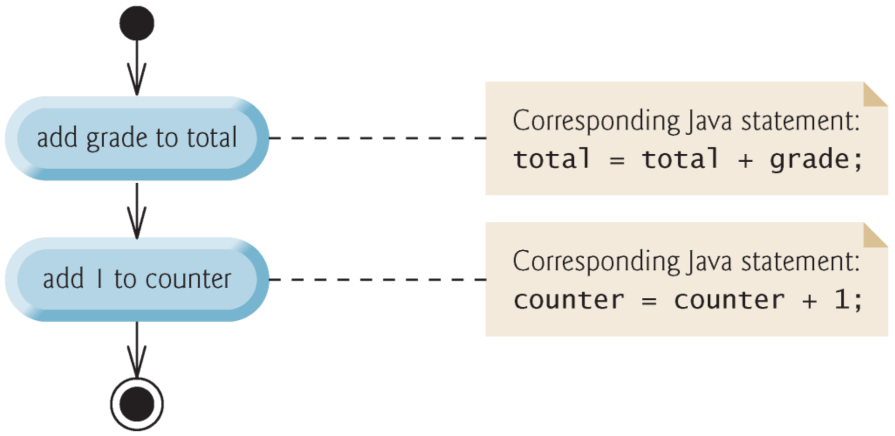


Fig. 4.1 | Sequence structure activity diagram.

4.4 Control Structures (Cont.)

- UML activity diagram
- Models the workflow (also called the activity) of a portion of a software system.
- May include a portion of an algorithm, like the sequence structure in Fig. 4.1.
- Composed of symbols
  - action-state symbols (rectangles with their left and right sides replaced with outward arcs)
  - diamonds
  - small circles
- Symbols connected by transition arrows, which represent the flow of the activity—the order in which the actions should occur.
- Help you develop and represent algorithms.
- Clearly show how control structures operate.

4.4 Control Structures (Cont.)

- Sequence structure activity diagram in Fig. 4.1.
- Two action states that represent actions to perform.
- Each contains an action expression that specifies a particular action to perform.
- Arrows represent transitions (order in which the actions represented by the action states occur).
- Solid circle at the top represents the initial state—the beginning of the workflow before the program performs the modeled actions.
- Solid circle surrounded by a hollow circle at the bottom represents the final state—the end of the workflow after the program performs its actions.

4.4 Control Structures (Cont.)

- UML notes
  - Like comments in Java.
  - Rectangles with the upper-right corners folded over.
  - Dotted line connects each note with the element it describes.
  - Activity diagrams normally do not show the Java code that implements the activity. We do this here to illustrate how the diagram relates to Java code.
- More information on the UML
  - visit `www.uml.org`

4.4 Control Structures (Cont.)

- Three types of selection statements.
- `if` statement:
  - Performs an action, if a condition is true; skips it, if false.
  - Single-selection statement—selects or ignores a single action (or group of actions).
- `if`…`else` statement:
  - Performs an action if a condition is true and performs a different action if the condition is false.
  - Double-selection statement—selects between two different actions (or groups of actions).
- `switch` statement
  - Performs one of several actions, based on the value of an expression.
  - Multiple-selection statement—selects among many different actions (or groups of actions).

4.4 Control Structures (Cont.)

- Three repetition statements (also called looping statements)
  - Perform statements repeatedly while a loop-continuation condition remains true.
- `while` and `for` statements perform the action(s) in their bodies zero or more times
  - if the loop-continuation condition is initially false, the body will not execute.
- The `do`…`while` statement performs the action(s) in its body one or more times.
- `if`, `else`, `switch`, `while`, `do` and `for` are keywords.

4.4 Control Structures (Cont.)

- Every program is formed by combining the sequence statement, selection statements (three types) and repetition statements (three types) as appropriate for the algorithm the program implements.
- Can model each control statement as an activity diagram.
  - Initial state and a final state represent a control statement's entry point and exit point, respectively.
  - Single-entry/single-exit control statements
  - Control-statement stacking—connect the exit point of one to the entry point of the next.
  - Control-statement nesting—a control statement inside another.

```
package compare_2_numbers;
import java.util.Scanner;

public class Compare_2_numbers {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Scanner input = new Scanner(System.in);

        int num1;
        int num2;

        System.out.print("Please enter first number");
        num1 = input.nextInt();

        System.out.print("Please enter second number");
        num2 = input.nextInt();
if (num1 == num2)
            System.out.printf("%d == %d\n", num1, num2);

        if (num1 != num2)
            System.out.printf("%d != %d\n", num1, num2);

        if (num1 < num2)
            System.out.printf("%d < %d\n", num1, num2);

        if (num1 > num2)
            System.out.printf("%d > %d\n", num1, num2);
        if (num1 <= num2)
            System.out.printf("%d <= %d\n", num1, num2);

        if (num1 >= num2)
            System.out.printf("%d >= %d\n", num1, num2);
    }// end of main

}/// end of class Compare_2_numbers
```

if statement

An if statement always begins with keyword if, followed by a condition in parentheses.

- Expects one statement in its body, but may contain multiple statements if they are enclosed in a set of braces ({}).
- The indentation of the body statement is not required, but it improves the program’s readability by emphasizing that statements are part of the body.

Note that there is no semicolon (;) at the end of the first line of each if statement.

- Such a semicolon would result in a logic error at execution time.

4.5 `if` Single-Selection Statement

- Pseudocode

```
If student's grade is greater than or equal to 60
    Print "Passed"
```

- If the condition is false, the Print statement is ignored, and the next pseudocode statement in order is performed.
- Indentation
  - Optional, but recommended
  - Emphasizes the inherent structure of structured programs
- The preceding pseudocode *If* in Java:

```
if ( studentGrade >= 60 )
   System.out.println( "Passed" );
```

- Corresponds closely to the pseudocode.

4.5 `if` Single-Selection Statement (Cont.)

- Figure 4.2 `if` statement UML activity diagram.
- Diamond, or decision symbol, indicates that a decision is to be made.
- Workflow continues along a path determined by the symbol's guard conditions, which can be true or false.
- Each transition arrow emerging from a decision symbol has a guard condition (in square brackets next to the arrow).
- If a guard condition is true, the workflow enters the action state to which the transition arrow points.

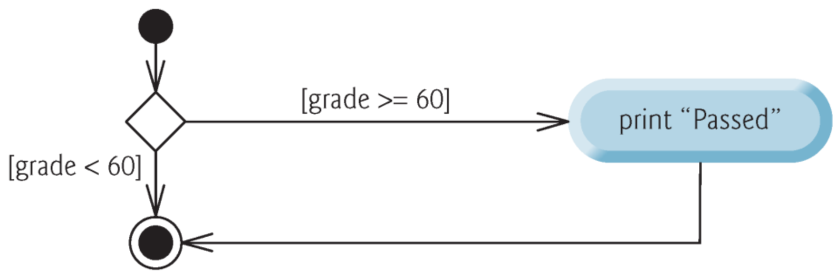


Fig. 4.2 | `if` single-selection statement UML activity diagram.

4.6 if…else Double-Selection Statement

- if…else double-selection statement—specify an action to perform when the condition is true and a different action when the condition is false.
- Pseudocode

```
If student's grade is greater than or equal to 60
    Print "Passed"
Else
    Print "Failed"
```

- The preceding *If…Else pseudocode statement in Java:*

```
if ( grade >= 60 )
   System.out.println( "Passed" );
else
   System.out.println( "Failed" );
```

- Note that the body of the `else` is also indented.

Good Programming Practice 4.1

Indent both body statements of an `if...else` *statement. Many IDEs do this for you.*

Good Programming Practice 4.2

If there are several levels of indentation, each level should be indented the same additional amount of space.

4.6 `if…else` Double-Selection Statement (Cont.)

- Figure 4.3 illustrates the flow of control in the `if…else` statement.
- The symbols in the UML activity diagram (besides the initial state, transition arrows and final state) represent action states and decisions.

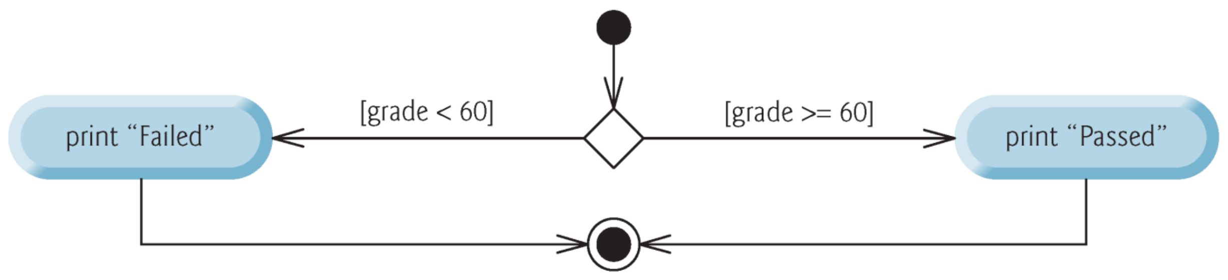


Fig. 4.3 | `if...else` double-selection statement UML activity diagram.

4.6 `if...else` Double-Selection Statement (Cont.)

- Conditional operator (`?:`)—shorthand `if`…`else`.
- Ternary operator (takes three operands)
- Operands and `?:` form a conditional expression
- Operand to the left of the `?` is a `boolean` expression—evaluates to a `boolean` value (`true` or `false`)
- Second operand (between the `?` and `:`) is the value if the `boolean` expression is `true`
- Third operand (to the right of the `:`) is the value if the `boolean` expression evaluates to `false`.

4.6 `if...else` Double-Selection Statement (Cont.)

- Example:

```
System.out.println(
   studentGrade >= 60 ? "Passed" : "Failed" );
```

- Evaluates to the string `"Passed"` if the `boolean` expression `studentGrade >= 60` is true and to the string `"Failed"` if it is false.

Good Programming Practice 4.3

Conditional expressions are more difficult to read than `if...else` *statements and should be used to replace only simple* `if...else` *statements that choose between two values.*

4.6 `if...else` Double-Selection Statement (Cont.)

- Can test multiple cases by placing `if`…`else` statements inside other `if`…`else` statements to create nested *`if...else` statements*.
- Pseudocode:

```
If student's grade is greater than or equal to 90
    Print "A"
else
    If student's grade is greater than or equal to 80
        Print "B"
    else
        If student's grade is greater than or equal to 70
            Print "C"
        else
            If student's grade is greater than or equal to 60
                Print "D"
            else
                Print "F"
```

4.6 `if...else` Double-Selection Statement (Cont.)

- This pseudocode may be written in Java as

```
if ( studentGrade >= 90 )
   System.out.println( "A" );
else
   if ( studentGrade >= 80 )
      System.out.println( "B" );
   else
      if ( studentGrade >= 70 )
         System.out.println( "C" );
      else
         if ( studentGrade >= 60 )
            System.out.println( "D" );
         else
            System.out.println( "F" );
```

- If `studentGrade` >= 90, the first four conditions will be true, but only the statement in the `if` part of the first `if`…`else` statement will execute. After that, the `else` part of the “outermost” `if`…`else` statement is skipped.

4.6 if…else Double-Selection Statement (Cont.)

- Most Java programmers prefer to write the preceding nested if…else statement as

```
if ( studentGrade >= 90 )
   System.out.println( "A" );
else if ( studentGrade >= 80 )
   System.out.println( "B" );
else if ( studentGrade >= 70 )
   System.out.println( "C" );
else if ( studentGrade >= 60 )
   System.out.println( "D" );
else
   System.out.println( "F" );
```

- The two forms are identical except for the spacing and indentation, which the compiler ignores.

4.6 if…else Double-Selection Statement (Cont.)

- The Java compiler always associates an `else` with the immediately preceding `if` unless told to do otherwise by the placement of braces (`{` and `}`).
- Referred to as the dangling-`else` problem.
- The following code is not what it appears:

```
if ( x > 5 )
   if ( y > 5 )
      System.out.println( "x and y are > 5" );
else
   System.out.println( "x is <= 5" );
```

- Beware! This nested `if`…`else` statement does not execute as it appears. The compiler actually interprets the statement as

```
if ( x > 5 )
   if ( y > 5 )
      System.out.println( "x and y are > 5" );
   else
      System.out.println( "x is <= 5" );
```

4.6 `if…else` Double-Selection Statement (Cont.)

- To force the nested `if…else` statement to execute as it was originally intended, we must write it as follows:

```
if ( x > 5 )
{
   if ( y > 5 )
      System.out.println( "x and y are > 5"
);
}
else
   System.out.println( "x is <= 5" );
```

- The braces indicate that the second `if` is in the body of the first and that the `else` is associated with the *first* *`if`*.

4.6 `if...else` Double-Selection Statement (Cont.)

- The `if` statement normally expects only one statement in its body.
- To include several statements in the body of an `if` (or the body of an `else` for an `if...else` statement), enclose the statements in braces.
- Statements contained in a pair of braces form a block.
- A block can be placed anywhere that a single statement can be placed.
- Example: A block in the `else` part of an `if...else` statement:

```
if ( grade >= 60 )
   System.out.println("Passed");
else
{
   System.out.println("Failed");
   System.out.println("You must take this course
again.");
}
```

4.6 `if…else` Double-Selection Statement (Cont.)

- Syntax errors (e.g., when one brace in a block is left out of the program) are caught by the compiler.
- A logic error (e.g., when both braces in a block are left out of the program) has its effect at execution time.
- A fatal logic error causes a program to fail and terminate prematurely.
- A nonfatal logic error allows a program to continue executing but causes it to produce incorrect results.

4.6 `if...else` Double-Selection Statement (Cont.)

- Just as a block can be placed anywhere a single statement can be placed, it’s also possible to have an empty statement.
- The empty statement is represented by placing a semicolon (`;`) where a statement would normally be.

Common Programming Error 4.1

Placing a semicolon after the condition in an `if` *or* `if…else` *statement leads to a logic error in single-selection* `if` *statements and a syntax error in double-selection* `if…else` *statements (when the* `if`*-part contains an actual body statement).*

4.7 `while` Repetition Statement

- Repetition statement—repeats an action while a condition remains true.
- Pseudocode

```
While there are more items on my shopping list
    Purchase next item and cross it off my list
```

- The repetition statement’s body may be a single statement or a block.
- Eventually, the condition will become false. At this point, the repetition terminates, and the first statement after the repetition statement executes.

4.7 while Repetition Statement (Cont.)

- Example of Java’s while repetition statement: find the first power of 3 larger than 100. Assume int variable product is initialized to 3.

```
while ( product <= 100 )
   product = 3 * product;
```

- Each iteration multiplies product by 3, so product takes on the values 9, 27, 81 and 243 successively.
- When variable product becomes 243, the while-statement condition—product <= 100—becomes false.
- Repetition terminates. The final value of product is 243.
- Program execution continues with the next statement after the while statement.

Common Programming Error 4.2

Not providing in the body of a `while` *statement an action that eventually causes the condition in the* `while` *to become false normally results in a logic error called an* ***infinite loop*** *(the loop never terminates).*

4.7 `while` Repetition Statement (Cont.)

- The UML activity diagram in Fig. 4.4 illustrates the flow of control in the preceding `while` statement.
- The UML represents both the merge symbol and the decision symbol as diamonds.
- The merge symbol joins two flows of activity into one.

4.7 `while` Repetition Statement (Cont.)

- The decision and merge symbols can be distinguished by the number of "incoming" and "outgoing" transition arrows.
  - A decision symbol has one transition arrow pointing to the diamond and two or more pointing out from it to indicate possible transitions from that point. Each transition arrow pointing out of a decision symbol has a guard condition next to it.
  - A merge symbol has two or more transition arrows pointing to the diamond and only one pointing from the diamond, to indicate multiple activity flows merging to continue the activity. None of the transition arrows associated with a merge symbol has a guard condition.

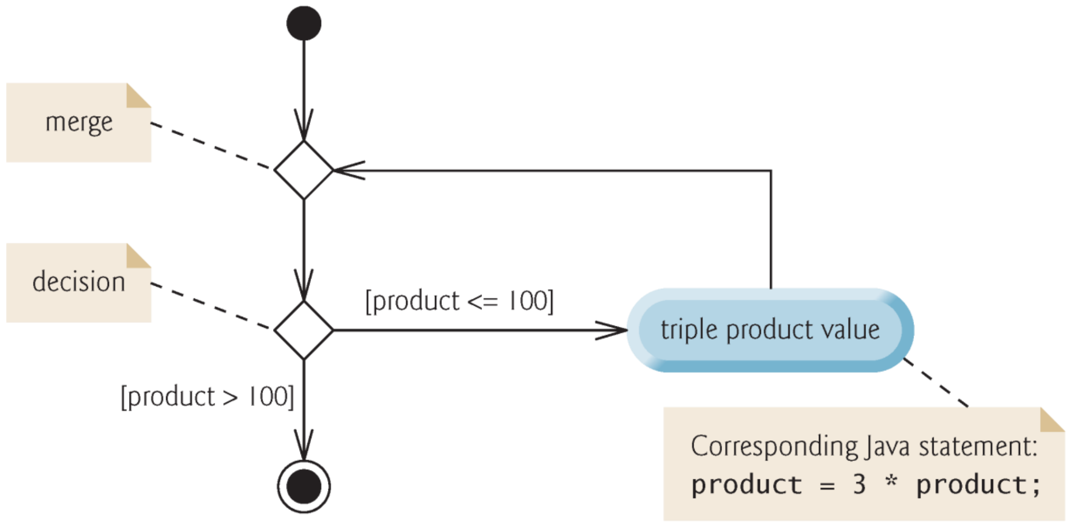


Fig. 4.4 | `while` repetition statement UML activity diagram.

4.8 Formulating Algorithms: Counter-Controlled Repetition

- *A class of ten students took a quiz. The grades (integers in the range 0 to 100) for this quiz are available to you. Determine the class average on the quiz.*
- The class average is equal to the sum of the grades divided by the number of students.
- The algorithm for solving this problem on a computer must input each grade, keep track of the total of all grades input, perform the averaging calculation and print the result.
- Use counter-controlled repetition to input the grades one at a time.
- A variable called a counter (or control variable) controls the number of times a set of statements will execute.
- Counter-controlled repetition is often called definite repetition, because the number of repetitions is known before the loop begins executing.

4.8 Formulating Algorithms: Counter-Controlled Repetition (Cont.)

- A total is a variable used to accumulate the sum of several values.
- A counter is a variable used to count.
- Variables used to store totals are normally initialized to zero before being used in a program.

```
Set total to zero
Set grade counter to one

While grade counter is less than or equal to ten
    Prompt the user to enter the next grade
    Input the next grade
    Add the grade into the total
    Add one to the grade counter

Set the class average to the total divided by ten
Print the class average
```

Fig. 4.5 | Pseudocode algorithm that uses counter-controlled repetition to solve the class-average problem.

4.8 Formulating Algorithms: Counter-Controlled Repetition (Cont.)

- Variables declared in a method body are local variables and can be used only from the line of their declaration to the closing right brace of the method declaration.
- A local variable’s declaration must appear before the variable is used in that method.
- A local variable cannot be accessed outside the method in which it’s declared.

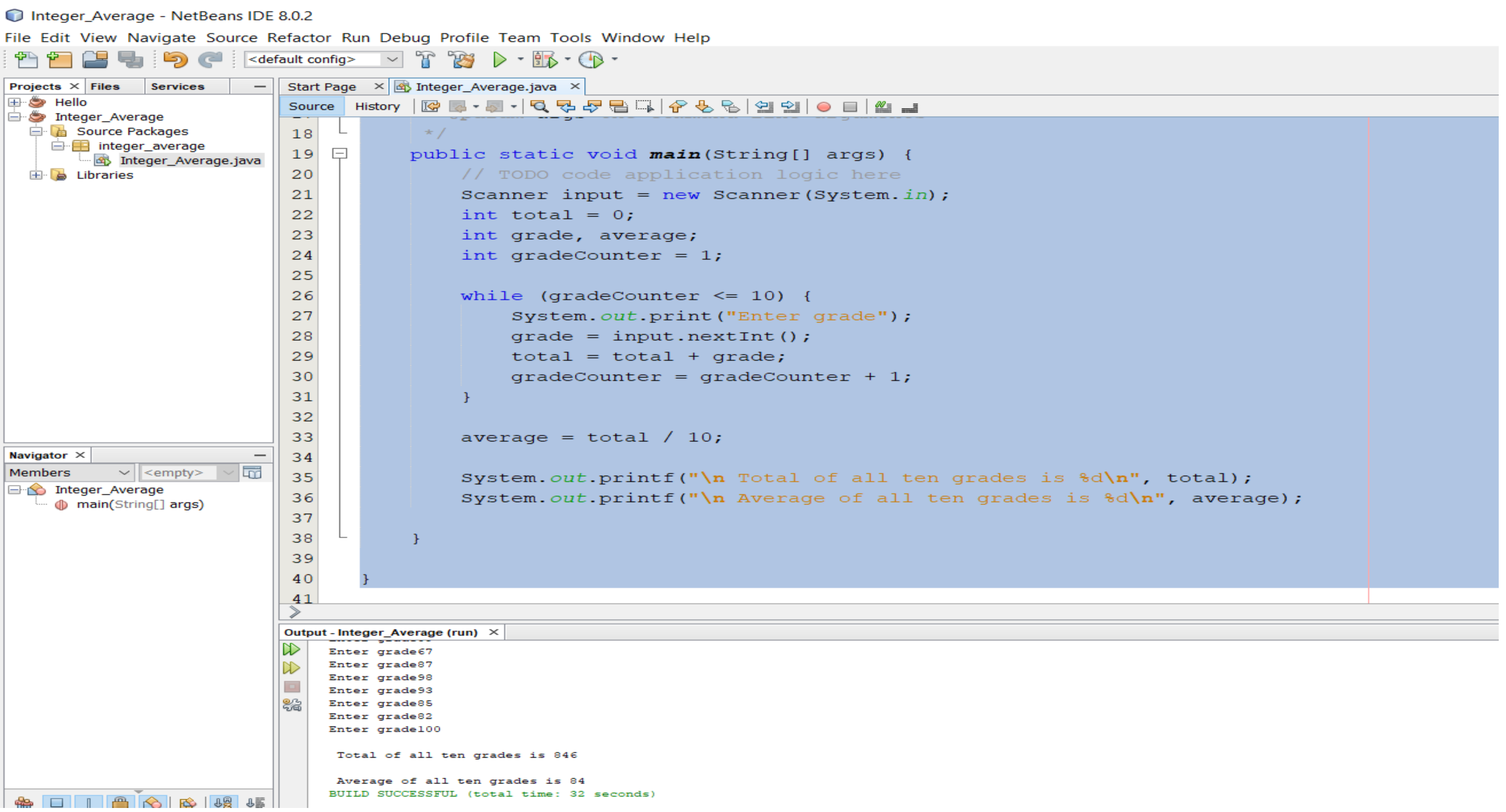
Common Programming Error 4.3

Using the value of a local variable before it's initialized results in a compilation error. All local variables must be initialized before their values are used in expressions.

Error-Prevention Tip 4.1

Initialize each counter and total, either in its declaration or in an assignment statement. Totals are normally initialized to 0. Counters are normally initialized to 0 or 1, depending on how they're used (we'll show examples of when to use 0 and when to use 1).

4.9 Integer Division Average



```
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Scanner input = new Scanner(System.in);
        int total = 0;
        int grade, average;
        int gradeCounter = 1;

        while (gradeCounter <= 10) {
            System.out.print("Enter grade");
            grade = input.nextInt();
            total = total + grade;
            gradeCounter = gradeCounter + 1;
        }

        average = total / 10;

        System.out.printf("\n Total of all ten grades is %d\n", total);
        System.out.printf("\n Average of all ten grades is %d\n", average);

    }

}
```

```
Enter grade67
Enter grade87
Enter grade98
Enter grade93
Enter grade85
Enter grade82
Enter grade100

 Total of all ten grades is 846

 Average of all ten grades is 84
BUILD SUCCESSFUL (total time: 32 seconds)
```

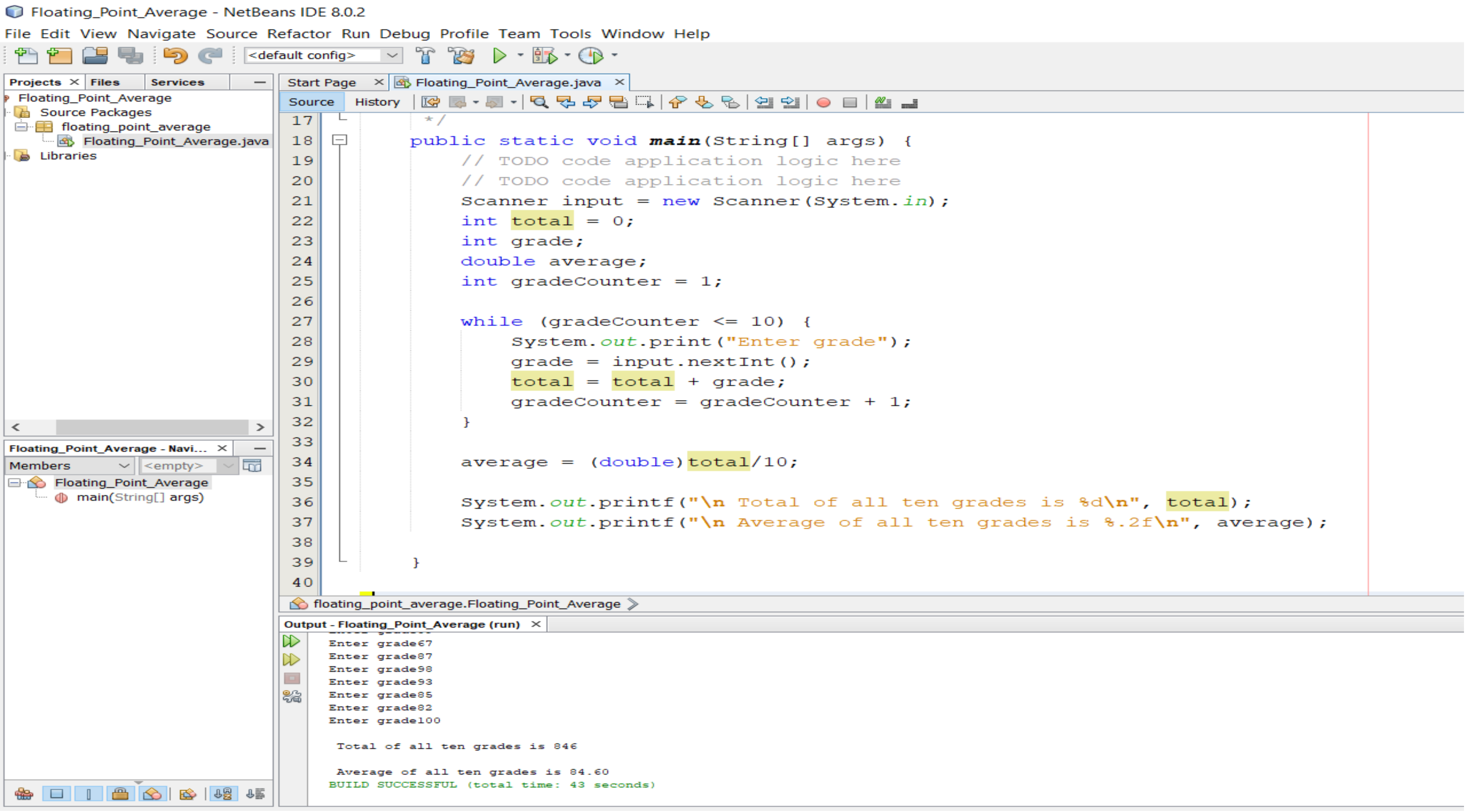
```
Welcome to the grade book for
CS101 Introduction to Java Programming!

Enter grade: 67
Enter grade: 78
Enter grade: 89
Enter grade: 67
Enter grade: 87
Enter grade: 98
Enter grade: 93
Enter grade: 85
Enter grade: 82
Enter grade: 100

Total of all 10 grades is 846
Class average is 84
```

Fig. 4.7 | `GradeBookTest` class creates an object of class `GradeBook` (Fig. 4.6) and invokes its `determineClassAverage` method. (Part 2 of 2.)

Common Programming Error 4.4

Assuming that integer division rounds (rather than truncates) can lead to incorrect results. For example, 7 ÷ 4, which yields 1.75 in conventional arithmetic, truncates to 1 in integer arithmetic, rather than rounding to 2.

4.10 Floating Point Division Average


Floating_Point_Average - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Projects Files Services
Floating_Point_Average
Source Packages
floating_point_average
Floating_Point_Average.java
Libraries
Start Page
Floating_Point_Average.java
Source History
Floating_Point_Average - Navi...
Members <empty>
Floating_Point_Average
main(String[] args)
floating_point_average.Floating_Point_Average
Output - Floating_Point_Average (run)

```java
     */
    public static void main(String[] args) {
        // TODO code application logic here
        // TODO code application logic here
        Scanner input = new Scanner(System.in);
        int total = 0;
        int grade;
        double average;
        int gradeCounter = 1;

        while (gradeCounter <= 10) {
            System.out.print("Enter grade");
            grade = input.nextInt();
            total = total + grade;
            gradeCounter = gradeCounter + 1;
        }

        average = (double)total/10;

        System.out.printf("\n Total of all ten grades is %d\n", total);
        System.out.printf("\n Average of all ten grades is %.2f\n", average);

    }

```

```
Enter grade67
Enter grade87
Enter grade98
Enter grade93
Enter grade85
Enter grade82
Enter grade100

 Total of all ten grades is 846

 Average of all ten grades is 84.60
BUILD SUCCESSFUL (total time: 43 seconds)
```

4.9 Formulating Algorithms: Sentinel-Controlled Repetition

- *Develop a class-averaging program that processes grades for an arbitrary number of students each time it is run.*
- Sentinel-controlled repetition is often called indefinite repetition because the number of repetitions is not known before the loop begins executing.
- A special value called a sentinel value (also called a signal value, a dummy value or a flag value) can be used to indicate "end of data entry."
- A sentinel value must be chosen that cannot be confused with an acceptable input value.

4.9 Formulating Algorithms: Sentinel-Controlled Repetition (Cont.)

- Top-down, stepwise refinement
- Begin with a pseudocode representation of the top—a single statement that conveys the overall function of the program:
    - *Determine the class average for the quiz*
- The top is a *complete representation of a program*. Rarely conveys sufficient detail from which to write a Java program.

4.9 Formulating Algorithms: Sentinel-Controlled Repetition (Cont.)

- Divide the top into a series of smaller tasks and list these in the order in which they’ll be performed.
- First refinement:

```
Initialize variables
Input, sum and count the quiz grades
Calculate and print the class average
```

- This refinement uses only the sequence structure—the steps listed should execute in order, one after the other.

4.9 Formulating Algorithms: Sentinel-Controlled Repetition (Cont.)

- Second refinement: commit to specific variables.
- The pseudocode statement
  - *Initialize variables*
- can be refined as follows:
  - *Initialize total to zero*
  - *Initialize counter to zero*

4.9 Formulating Algorithms: Sentinel-Controlled Repetition (Cont.)

- The pseudocode statement

  Input, sum and count the quiz grades
- requires a repetition structure that successively inputs each grade.
- We do not know in advance how many grades are to be processed, so we’ll use sentinel-controlled repetition.

4.9 Formulating Algorithms: Sentinel-Controlled Repetition (Cont.)

- The second refinement of the preceding pseudocode statement is then

```
Prompt the user to enter the first grade
Input the first grade (possibly the sentinel)
While the user has not yet entered the sentinel
    Add this grade into the running total
    Add one to the grade counter
    Prompt the user to enter the next grade
    Input the next grade (possibly the sentinel)
```

4.9 Formulating Algorithms: Sentinel-Controlled Repetition (Cont.)

- The pseudocode statement

```
Calculate and print the class average
```

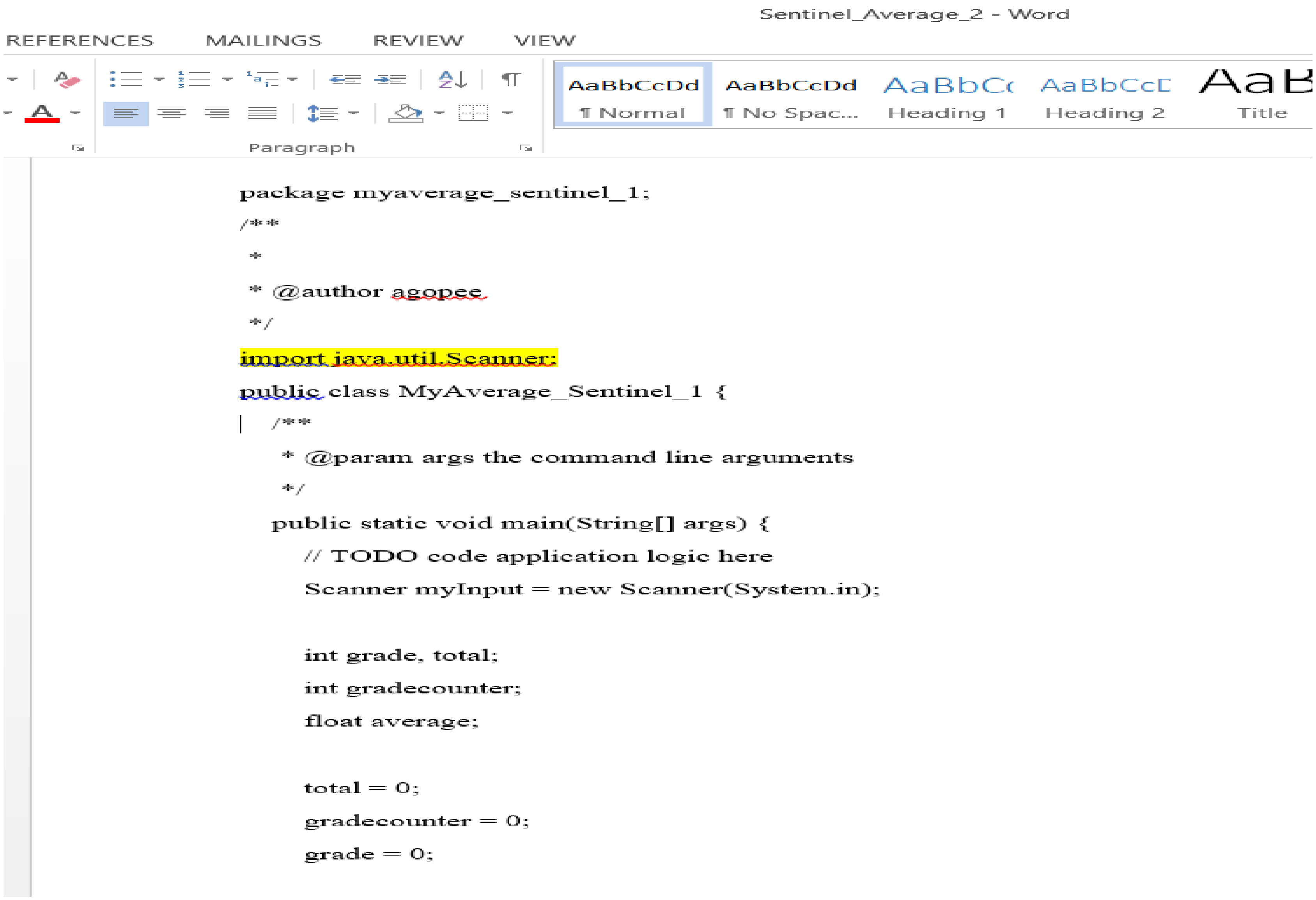
- can be refined as follows:

```
If the counter is not equal to zero
    Set the average to the total divided by the counter
    Print the average
else
    Print "No grades were entered"
```

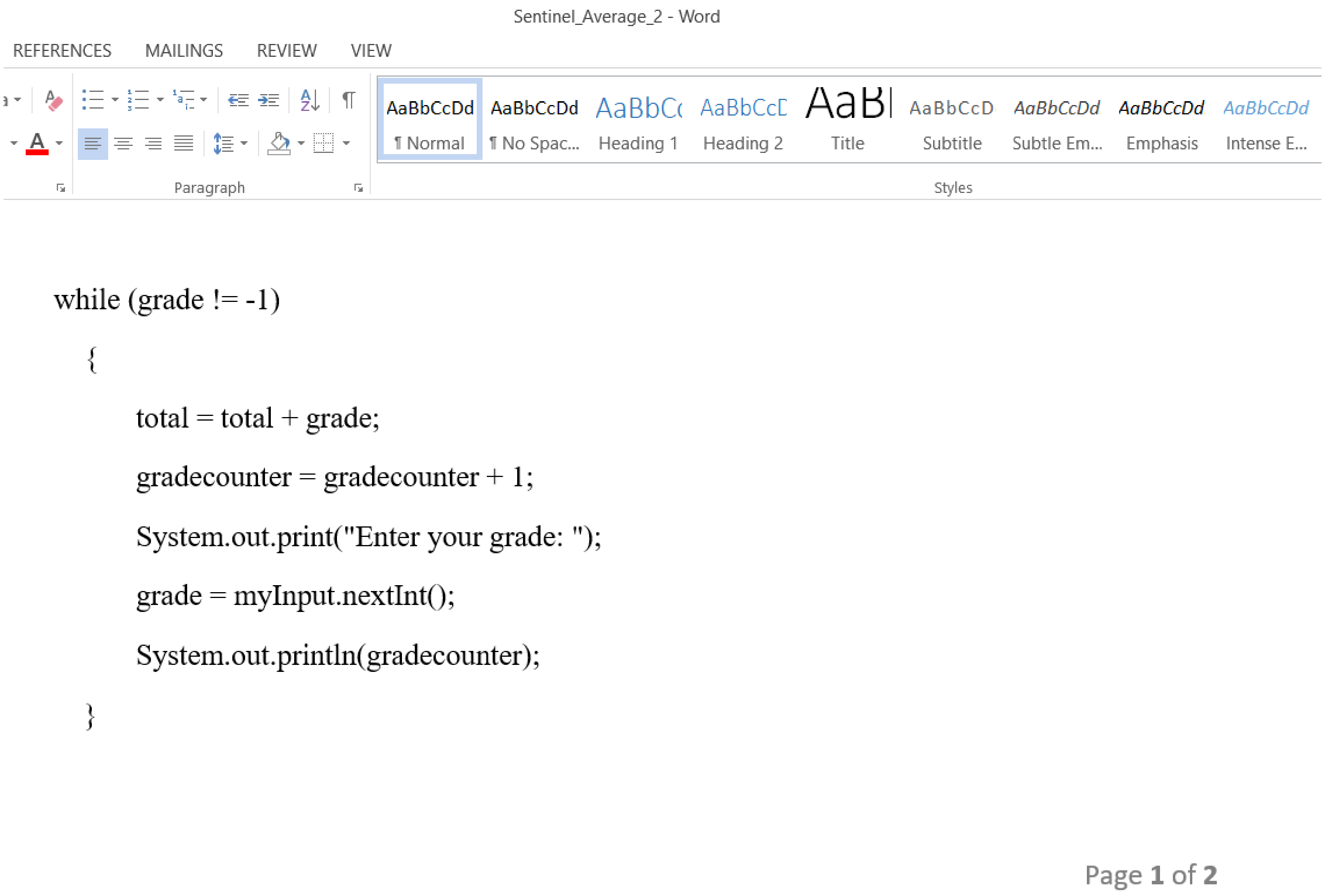
- Test for the possibility of division by zero—a logic error that, if undetected, would cause the program to fail or produce invalid output.

```
Initialize total to zero
Initialize counter to zero

Prompt the user to enter the first grade
Input the first grade (possibly the sentinel)

While the user has not yet entered the sentinel
    Add this grade into the running total
    Add one to the grade counter
    Prompt the user to enter the next grade
    Input the next grade (possibly the sentinel)

If the counter is not equal to zero
    Set the average to the total divided by the counter
    Print the average
else
    Print "No grades were entered"
```

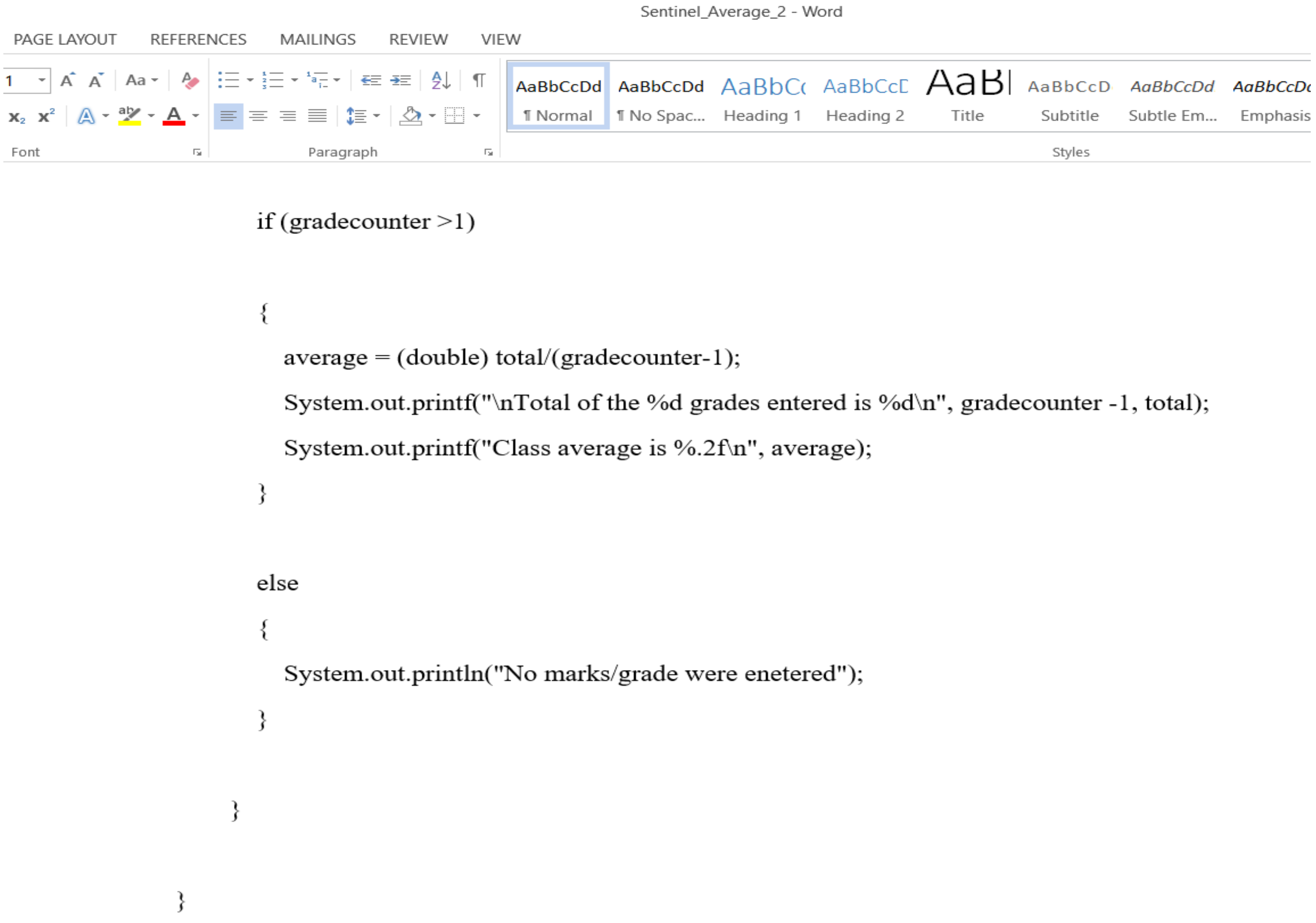
Fig. 4.8 | Class-average problem pseudocode algorithm with sentinel-controlled repetition.



```
package myaverage_sentinel_1;
/**
 *
 * @author agopee
 */
import java.util.Scanner;
public class MyAverage_Sentinel_1 {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Scanner myInput = new Scanner(System.in);

        int grade, total;
        int gradecounter;
        float average;

        total = 0;
        gradecounter = 0;
        grade = 0;
```



```
while (grade != -1)
   {
        total = total + grade;
        gradecounter = gradecounter + 1;
        System.out.print("Enter your grade: ");
        grade = myInput.nextInt();
        System.out.println(gradecounter);
   }
```



```
    if (gradecounter >1)

    {
      average = (double) total/(gradecounter-1);
      System.out.printf("\nTotal of the %d grades entered is %d\n", gradecounter -1, total);
      System.out.printf("Class average is %.2f\n", average);
    }

    else
    {
      System.out.println("No marks/grade were enetered");
    }

  }

}
```

4.9 Formulating Algorithms: Sentinel-Controlled Repetition (Cont.)

- Program logic for sentinel-controlled repetition
  - Reads the first value before reaching the `while`.
  - This value determines whether the program’s flow of control should enter the body of the `while`. If the condition of the `while` is false, the user entered the sentinel value, so the body of the `while` does not execute (i.e., no grades were entered).
  - If the condition is true, the body begins execution and processes the input.
  - Then the loop body inputs the next value from the user before the end of the loop.

Good Programming Practice 4.3

In a sentinel-controlled loop, prompts should remind the user of the sentinel.

Common Programming Error 4.5

Omitting the braces that delimit a block can lead to logic errors, such as infinite loops. To prevent this problem, some programmers enclose the body of every control statement in braces, even if the body contains only a single statement.

4.9 Formulating Algorithms: Sentinel-Controlled Repetition (Cont.)

- Integer division yields an integer result.
- To perform a floating-point calculation with integers, temporarily treat these values as floating-point numbers for use in the calculation.
- The unary cast operator `(double)` creates a temporary floating-point copy of its operand.
- Cast operator performs explicit conversion (or type cast).
- The value stored in the operand is unchanged.
- Java evaluates only arithmetic expressions in which the operands' types are identical.
- Promotion (or implicit conversion) performed on operands.
- In an expression containing values of the types `int` and `double`, the `int` values are promoted to `double` values for use in the expression.

4.9 Formulating Algorithms: Sentinel-Controlled Repetition (Cont.)

- Cast operators are available for any type.
- Cast operator formed by placing parentheses around the name of a type.
- The operator is a unary operator (i.e., an operator that takes only one operand).
- Java also supports unary versions of the plus (+) and minus (–) operators.
- Cast operators associate from right to left; same precedence as other unary operators, such as unary + and unary –.
- This precedence is one level higher than that of the multiplicative operators *, / and %.

4.10 Formulating Algorithms: Nested Control Statements

- This case study examines nesting one control statement within another.
- A college offers a course that prepares students for the state licensing exam for real estate brokers. Last year, ten of the students who completed this course took the exam. The college wants to know how well its students did on the exam. You’ve been asked to write a program to summarize the results. You’ve been given a list of these 10 students. Next to each name is written a 1 if the student passed the exam or a 2 if the student failed.

4.10 Formulating Algorithms: Nested Control Statements (Cont.)

- This case study examines nesting one control statement within another.
- Your program should analyze the results of the exam as follows:
  - Input each test result (i.e., a 1 or a 2). Display the message “Enter result” on the screen each time the program requests another test result.
  - Count the number of test results of each type.
  - Display a summary of the test results, indicating the number of students who passed and the number who failed.
  - If more than eight students passed the exam, print the message “Bonus to instructor!”

```
Initialize passes to zero
Initialize failures to zero
Initialize student counter to one

While student counter is less than or equal to 10
    Prompt the user to enter the next exam result
    Input the next exam result

    If the student passed
        Add one to passes
    Else
        Add one to failures

    Add one to student counter

Print the number of passes
Print the number of failures

If more than eight students passed
    Print "Bonus to instructor!"
```

Fig. 4.11 | Pseudocode for examination-results problem.

Error-Prevention Tip 4.3

Initializing local variables when they're declared helps you avoid any compilation errors that might arise from attempts to use uninitialized variables. While Java does not require that local-variable initializations be incorporated into declarations, it does require that local variables be initialized before their values are used in an expression.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

package no_of_passes;

/**
 *
 * @author user
 */
import java.util.Scanner;
public class No_of_passes {
```

```
/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    // TODO code application logic here
    Scanner input = new Scanner(System.in);

    int passes = 0;
    int failures = 0;
    int studentCounter = 1;
    int result;

    while (studentCounter <=10)
    {
        System.out.print("Enter result (1 - pass, 2 - fail):");
        result = input.nextInt();


        if (result == 1)
        passes = passes + 1;
        else
        failures = failures + 1;
        studentCounter = studentCounter + 1;
    }
    System.out.printf("Passed: %d\n Failed: %d\n", passes, failures);
    if (passes > 8)
        System.out.println("Bonus to instructor");


}
```

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 6
Failed: 4
```

Fig. 4.12 | Analysis of examination results using nested control statements. (Part 4 of 4.)

4.11 Compound Assignment Operators

- Compound assignment operators abbreviate assignment expressions.
- Statements like

  variable = variable operator expression;

  where operator is one of the binary operators `+`, `-`, `*`, `/` or `%` can be written in the form

  variable operator= expression;
- Example:

  ```
  c = c + 3;
  ```

  can be written with the addition compound assignment operator, +=, as

  ```
  c += 3;
  ```
- The `+=` operator adds the value of the expression on its right to the value of the variable on its left and stores the result in the variable on the left of the operator.

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`c = c + 7`	10 to `c`
`-=`	`d -= 4`	`d = d - 4`	1 to `d`
`*=`	`e *= 5`	`e = e * 5`	20 to `e`
`/=`	`f /= 3`	`f = f / 3`	2 to `f`
`%=`	`g %= 9`	`g = g % 9`	3 to `g`

Fig. 4.13 | Arithmetic compound assignment operators.

4.12 Increment and Decrement Operators

- Unary increment operator, ++, adds one to its operand
- Unary decrement operator, −−, subtracts one from its operand
- An increment or decrement operator that is prefixed to (placed before) a variable is referred to as the prefix increment or prefix decrement operator, respectively.
- An increment or decrement operator that is postfixed to (placed after) a variable is referred to as the postfix increment or postfix decrement operator, respectively.

Operator	Operator name	Sample expression	Explanation
`++`	prefix increment	`++a`	Increment `a` by `1`, then use the new value of `a` in the expression in which `a` resides.
`++`	postfix increment	`a++`	Use the current value of `a` in the expression in which `a` resides, then increment `a` by `1`.
`--`	prefix decrement	`--b`	Decrement `b` by `1`, then use the new value of `b` in the expression in which `b` resides.
`--`	postfix decrement	`b--`	Use the current value of `b` in the expression in which `b` resides, then decrement `b` by `1`.

Fig. 4.14 | Increment and decrement operators.

4.12 Increment and Decrement Operators (Cont.)

- Using the prefix increment (or decrement) operator to add (or subtract) 1 from a variable is known as preincrementing (or predecrementing) the variable.
- Preincrementing (or predecrementing) a variable causes the variable to be incremented (decremented) by 1; then the new value is used in the expression in which it appears.
- Using the postfix increment (or decrement) operator to add (or subtract) 1 from a variable is known as postincrementing (or postdecrementing) the variable.
- This causes the current value of the variable to be used in the expression in which it appears; then the variable’s value is incremented (decremented) by 1.

```java
// Fig. 4.15: Increment.java
// Prefix increment and postfix increment operators.

public class Increment
{
   public static void main( String[] args )
   {
      int c;

      // demonstrate postfix increment operator
      c = 5; // assign 5 to c
      System.out.println( c );   // prints 5
      System.out.println( c++ ); // prints 5 then postincrements
      System.out.println( c );   // prints 6

      System.out.println(); // skip a line

      // demonstrate prefix increment operator
      c = 5; // assign 5 to c
      System.out.println( c );   // prints 5
      System.out.println( ++c ); // preincrements then prints 6
      System.out.println( c );   // prints 6
   } // end main
} // end class Increment
```

Uses current value, then increments **c**

Increments **c** then uses new value

Fig. 4.15 | Preincrementing and postincrementing. (Part 1 of 2.)

```
5
5
6

5
6
6
```

Fig. 4.15 | Preincrementing and postincrementing. (Part 2 of 2.)

Common Programming Error 4.7

Attempting to use the increment or decrement operator on an expression other than one to which a value can be assigned is a syntax error. For example, writing `++(x + 1)` *is a syntax error, because* `(x + 1)` *is not a variable.*

Operators						Associativity	Type
`++`	`--`					right to left	unary postfix
`++`	`--`	`+`	`-`	`(` *type* `)`		right to left	unary prefix
`*`	`/`	`%`				left to right	multiplicative
`+`	`-`					left to right	additive
`<`	`<=`	`>`	`>=`			left to right	relational
`==`	`!=`					left to right	equality
`?:`						right to left	conditional
`=`	`+=`	`-=`	`*=`	`/=`	`%=`	right to left	assignment

Fig. 4.16 | Precedence and associativity of the operators discussed so far.

4.13 Primitive Types

- Java requires all variables to have a type.
- Java is a strongly typed language.
- Primitive types in Java are portable across all platforms.
- Instance variables of types `char`, `byte`, `short`, `int`, `long`, `float` and `double` are all given the value `0` by default. Instance variables of type `boolean` are given the value `false` by default.
- Reference-type instance variables are initialized by default to the value `null`.