**Chapter 4**

**Exercises 1**

**5.7** What is the value of x after each of the following statements is executed?

a) x = Math.abs(7.5);

**ANS:** x = 7.5

b) x = Math.floor(7.5);

**ANS:** x = 7

c) x = Math.abs(0.0);

**ANS:** x = 0.0

d) x = Math.ceil(0.0);

**ANS:** x = 0

e) x = Math.abs(-6.4);

**ANS:** x = 6.4

f) x = Math.ceil(-6.4);

**ANS:** x = -6

g) x = Math.ceil(-Math.abs(-8 + Math.floor(-5.5)));

**ANS:** x = -14

**5.8 (Parking Charges)** A parking garage charges a $2.00 minimum fee to park for up to three hours. The garage charges an additional $0.50 per hour for each hour or part thereof in excess of three hours. The maximum charge for any given 24-hour period is $10.00. Assume that no car parks for longer than 24 hours at a time. Write an application that calculates and displays the parking charges for each customer who parked in the garage yesterday. You should enter the hours parked for each customer. The program should display the charge for the current customer and should calculate and display the running total of yesterday’s receipts. It should use the method calculateCharges to determine the charge for each customer.

**5.9 (Rounding Numbers)** Math.floor can be used to round values to the nearest integer—e.g.,

y = Math.floor(x + 0.5);

will round the number x to the nearest integer and assign the result to y. Write an application that reads double values and uses the preceding statement to round each of the numbers to the nearest integer. For each number processed, display both the original number and the rounded number.

**5.10 (Rounding Numbers)** To round numbers to specific decimal places, use a statement like

y = Math.floor(x \* 10 + 0.5) / 10;

which rounds x to the tenths position (i.e., the first position to the right of the decimal point), or

y = Math.floor(x \* 100 + 0.5) / 100;

which rounds x to the hundredths position (i.e., the second position to the right of the decimal point). Write an application that defines four methods for rounding a number x in various ways:

a) roundToInteger(number)

b) roundToTenths(number)

c) roundToHundredths(number)

d) roundToThousandths(number)

For each value read, your program should display the original value, the number rounded to the nearest integer, the number rounded to the nearest tenth, the number rounded to the nearest hundredth and the number rounded to the nearest thousandth.

**5.12** Write statements that assign random integers to the variable n in the following ranges:

a) 1 ≤n ≤ 2.

b) 1 ≤n ≤ 100.

c) 0 ≤n ≤ 9.

d) 1000 ≤n ≤ 1112.

e) –1 ≤n ≤ 1.

f) –3 ≤n ≤ 11.

**5.13** Write statements that will display a random number from each of the following sets:

a) 2, 4, 6, 8, 10.

b) 3, 5, 7, 9, 11.

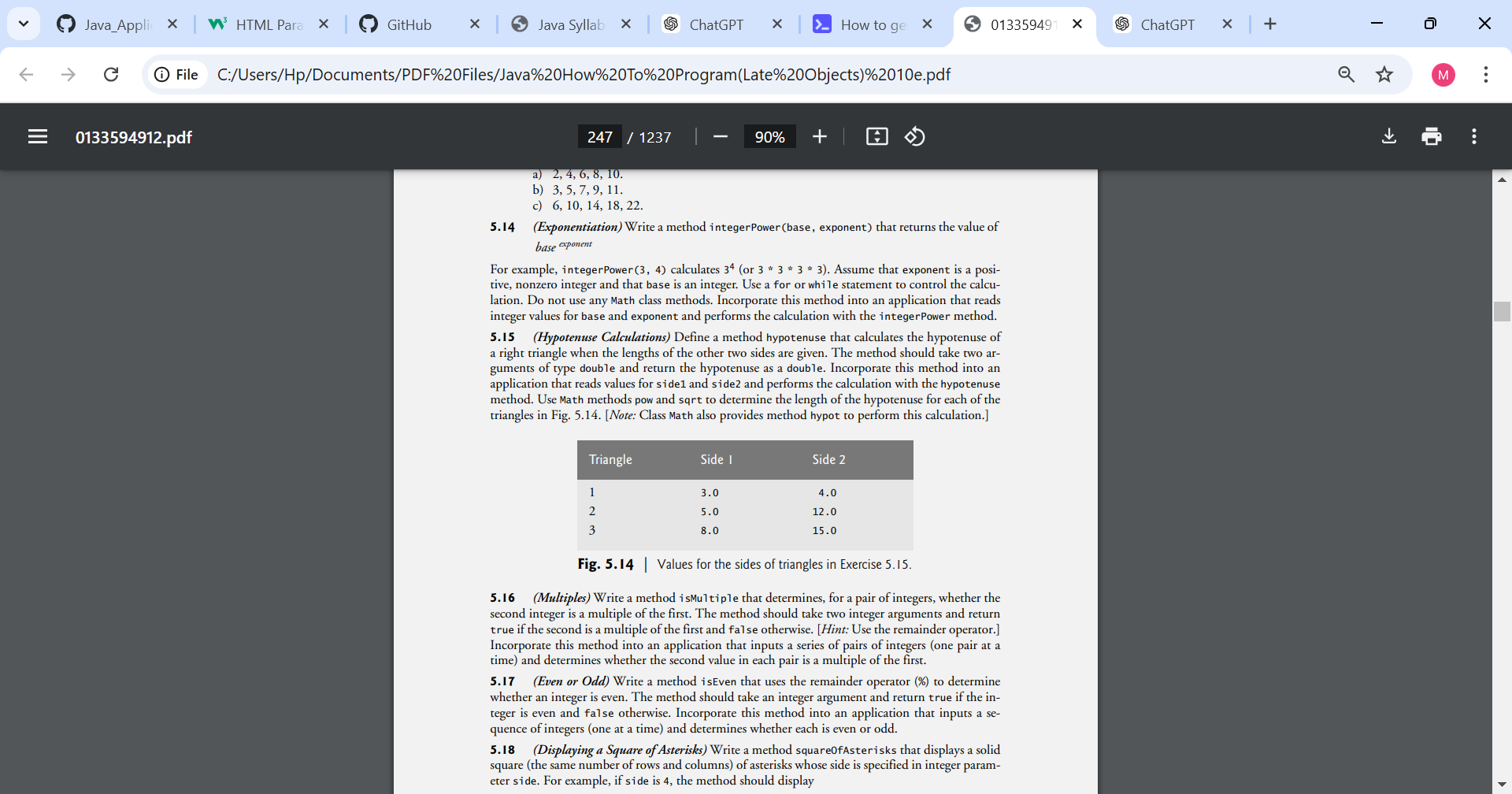
c) 6, 10, 14, 18, 22.

**5.14** (Exponentiation) Write a method integerPower(base, exponent) that returns the value of

base *exponent*

For example, integerPower(3, 4) calculates 34 (or 3\*3\*3\*3). Assume that exponent is a positive, nonzero integer and that base is an integer. Use a for or while statement to control the calculation. Do not use any Math class methods. Incorporate this method into an application that reads integer values for base and exponent and performs the calculation with the integerPower method.

**5.15 (Hypotenuse Calculations**) Define a method hypotenuse that calculates the hypotenuse of a right triangle when the lengths of the other two sides are given. The method should take two arguments of type double and return the hypotenuse as a double. Incorporate this method into an application that reads values for side1 and side2 and performs the calculation with the hypotenuse method. Use Math methods pow and sqrt to determine the length of the hypotenuse for each of the triangles in Fig. 5.14. [Note: Class Math also provides method hypot to perform this calculation.]



**5.16 (Multiples)** Write a method isMultiple that determines, for a pair of integers, whether the second integer is a multiple of the first. The method should take two integer arguments and return true if the second is a multiple of the first and false otherwise. [Hint: Use the remainder operator.] Incorporate this method into an application that inputs a series of pairs of integers (one pair at a time) and determines whether the second value in each pair is a multiple of the first.

**5.17 (Even or Odd)** Write a method isEven that uses the remainder operator (%) to determine whether an integer is even. The method should take an integer argument and return true if the integer is even and false otherwise. Incorporate this method into an application that inputs a sequence of integers (one at a time) and determines whether each is even or odd.

**5.18 (Displaying a Square of Asterisks)** Write a method squareOfAsterisks that displays a solid square (the same number of rows and columns) of asterisks whose side is specified in integer parameter side. For example, if side is 4, the method should display

\*\*\*\*

\*\*\*\*

\*\*\*\*

\*\*\*\*

Incorporate this method into an application that reads an integer value for side from the user and outputs the asterisks with the squareOfAsterisks method.

**5.19 (Displaying a Square of Any Character)** Modify the method created in Exercise 5.18 to receive a second parameter of type char called fillCharacter. Form the square using the char provided as an argument. Thus, if side is 5 and fillCharacter is #, the method should display

#####

#####

#####

#####

#####

Use the following statement (in which input is a Scanner object) to read a character from the user at the keyboard:

char fill = input.next().charAt(0);

**5.20 (Circle Area)** Write an application that prompts the user for the radius of a circle and uses a method called circleArea to calculate the area of the circle.

**5.21 (Separating Digits)** Write methods that accomplish each of the following tasks:

a) Calculate the integer part of the quotient when integer a is divided by integer b.

b) Calculate the integer remainder when integer a is divided by integer b.

c) Use the methods developed in parts (a) and (b) to write a method displayDigits that receives an integer between 1 and 99999 and displays it as a sequence of digits, separating each pair of digits by two spaces. For example, the integer 4562 should appear as

4 5 6 2

Incorporate the methods into an application that inputs an integer and calls displayDigits by passing the method the integer entered. Display the results.

**5.22 (Temperature Conversions)** Implement the following integer methods:

a) Method celsius returns the Celsius equivalent of a Fahrenheit temperature, using the calculation

celsius = 5.0 / 9.0 \* (fahrenheit - 32);

b) Method fahrenheit returns the Fahrenheit equivalent of a Celsius temperature, using the calculation fahrenheit = 9.0 / 5.0 \* celsius + 32;

c) Use the methods from parts (a) and (b) to write an application that enables the user either to enter a Fahrenheit temperature and display the Celsius equivalent or to enter a Celsius temperature and display the Fahrenheit equivalent.

**5.23 (Find the Minimum)** Write a method minimum3 that returns the smallest of three floatingpoint numbers. Use the Math.min method to implement minimum3. Incorporate the method into an application that reads three values from the user, determines the smallest value and displays the result.

**5.24 (Perfect Numbers)** An integer number is said to be a perfect number if its factors, including 1 (but not the number itself), sum to the number. For example, 6 is a perfect number, because 6 = 1 + 2 + 3. Write a method isPerfect that determines whether parameter number is a perfect number. Use this method in an application that displays all the perfect numbers between 1 and 1000. Display the factors of each perfect number to confirm that the number is indeed perfect. Challenge the computing power of your computer by testing numbers much larger than 1000. Display the results.

**5.25 (Prime Numbers)** A positive integer is prime if it’s divisible by only 1 and itself. For example, 2, 3, 5 and 7 are prime, but 4, 6, 8 and 9 are not. The number 1, by definition, is not prime.

a) Write a method that determines whether a number is prime.

b) Use this method in an application that determines and displays all the prime numbers less than 10,000. How many numbers up to 10,000 do you have to test to ensure that you’ve found all the primes?

c) Initially, you might think that n/2 is the upper limit for which you must test to see whether a number n is prime, but you need only go as high as the square root of n. Rewrite the program, and run it both ways.

**5.26 (Reversing Digits)** Write a method that takes an integer value and returns the number with its digits reversed. For example, given the number 7631, the method should return 1367. Incorporate the method into an application that reads a value from the user and displays the result.

**5.27 (Greatest Common Divisor)** Thegreatest common divisor (GCD) of two integers is the largest integer that evenly divides each of the two numbers. Write a method gcd that returns the greatest common divisor of two integers. [Hint: You might want to use Euclid’s algorithm. You can find information about it at en.wikipedia.org/wiki/Euclidean\_algorithm.] Incorporate the method into an application that reads two values from the user and displays the result.

**5.28** Write a method qualityPoints that inputs a student’s average and returns 4 if it’s 90–100, 3 if 80–89, 2 if 70–79, 1 if 60–69 and 0 if lower than 60. Incorporate the method into an application that reads a value from the user and displays the result.

**5.29 (Coin Tossing)** Write an application that simulates coin tossing. Let the program toss a coin each time the user chooses the “Toss Coin” menu option. Count the number of times each side of the coin appears. Display the results. The program should call a separate method flip that takes no arguments and returns a value from a Coin enum (HEADS and TAILS). [Note: If the program realistically simulates coin tossing, each side of the coin should appear approximately half the time.]

**5.30 (Guess the Number)** Write an application that plays “guess the number” as follows: Your program chooses the number to be guessed by selecting a random integer in the range 1 to 1000. The application displays the prompt Guess a number between 1 and 1000. The player inputs a first guess. If the player's guess is incorrect, your program should display Too high. Try again. or Too low. Try again. to help the player “zero in” on the correct answer. The program should prompt the user for the next guess. When the user enters the correct answer, display Congratulations. You guessed the number!, and allow the user to choose whether to play again. [Note: The guessing technique employed in this problem is similar to a binary search, which is discussed in Chapter 19, Searching, Sorting and Big O.]

**5.31 (Guess the Number Modification)** Modify the program of Exercise 5.30 to count the number of guesses the player makes. If the number is 10 or fewer, display Either you know the secret or you got lucky! If the player guesses the number in 10 tries, display Aha! You know the secret! If the player makes more than 10 guesses, display You should be able to do better! Why should it take no more than 10 guesses? Well, with each “good guess,” the player should be able to eliminate half of the numbers, then half of the remaining numbers, and so on.

**5.32 (Distance Between Points)** Write method distance to calculate the distance between two points (x1, y1) and (x2, y2). All numbers and return values should be of type double. Incorporate this method into an application that enables the user to enter the coordinates of the points.

**5.33 (Craps Game Modification)** Modify the craps program of Fig. 5.8 to allow wagering. Initialize variable bankBalance to 1000 dollars. Prompt the player to enter a wager. Check that wager is less than or equal to bankBalance, and if it’s not, have the user reenter wager until a valid wager is entered. Then, run one game of craps. If the player wins, increase bankBalance by wager and display the new bankBalance. If the player loses, decrease bankBalance by wager, display the new bankBalance, check whether bankBalance has become zero and, if so, display the message "Sorry. You busted!" As the game progresses, display various messages to create some “chatter,” such as "Oh, you're going for broke, huh?" or "Aw c'mon, take a chance!" or "You're up big. Now's the time to cash in your chips!". Implement the “chatter” as a separate method that randomly chooses the string to display.

**5.34 (Table of Binary, Octal and Hexadecimal Numbers)** Write an application that displays a table of the binary, octal and hexadecimal equivalents of the decimal numbers in the range 1 through 256. If you aren’t familiar with these number systems, read online Appendix J first.

Making a Difference

As computer costs decline, it becomes feasible for every student, regardless of economic circumstance, to have a computer and use it in school. This creates exciting possibilities for improving the educational experience of all students worldwide, as suggested by the next five exercises. [Note: Check out initiatives such as the One Laptop Per Child Project (www.laptop.org). Also, research “green” laptops—what are some key “going green” characteristics of these devices? Look into the Electronic Product Environmental Assessment Tool (www.epeat.net), which can help you assess the “greenness” of desktops, notebooks and monitors to help you decide which products to purchase.]

**5.35 (Computer-Assisted Instruction)** The use of computers in education is referred to as computer-assisted instruction (CAI). Write a program that will help an elementary school student learn multiplication. Use a SecureRandom object to produce two positive one-digit integers. The program should then prompt the user with a question, such as How much is 6 times 7? The student then inputs the answer. Next, the program checks the student’s answer. If it’s correct, display the message "Very good!" and ask another multiplication question. If the answer is wrong, display the message "No. Please try again." and let the student try the same question repeatedly until the student finally gets it right. A separate method should be used to generate each new question. This method should be called once when the application begins execution and each time the user answers the question correctly.

**5.36 (Computer-Assisted Instruction: Reducing Student Fatigue)** One problem in CAI environments is student fatigue. This can be reduced by varying the computer’s responses to hold the student’s attention. Modify the program of Exercise 5.35 so that various comments are displayed for each answer as follows:

Possible responses to a correct answer:

Very good!

Excellent!

Nice work!

Keep up the good work!

Possible responses to an incorrect answer:

No. Please try again.

Wrong. Try once more.

Don't give up! No.

Keep trying.

Use random-number generation to choose a number from 1 to 4 that will be used to select one of the four appropriate responses to each correct or incorrect answer. Use a switch statement to issue the responses.

**5.37 (Computer-Assisted Instruction: Monitoring Student Performance)** More sophisticated computer-assisted instruction systems monitor the student’s performance over a period of time. The decision to begin a new topic is often based on the student’s success with previous topics. Modify the program of Exercise 5.36 to count the number of correct and incorrect responses typed by the student. After the student types 10 answers, your program should calculate the percentage that are correct. If the percentage is lower than 75%, display "Please ask your teacher for extra help.", then reset the program so another student can try it. If the percentage is 75% or higher, display "Congratulations, you are ready to go to the next level!", then reset the program so another student can try it.

**5.38 (Computer-Assisted Instruction: Difficulty Levels)** Exercises 5.35–5.37 developed a computer-assisted instruction program to help teach an elementary school student multiplication. Modify the program to allow the user to enter a difficulty level. At a difficulty level of 1, the program should use only single-digit numbers in the problems; at a difficulty level of 2, numbers as large as two digits, and so on.

**5.39 (Computer-Assisted Instruction: Varying the Types of Problems)** Modify the program of Exercise 5.38 to allow the user to pick a type of arithmetic problem to study. An option of 1 means addition problems only, 2 means subtraction problems only, 3 means multiplication problems only, 4 means division problems only and 5 means a random mixture of all these types.