Quiz

Answer the quiz for this chapter online at www.cs.armstrong.edu/liang/intro10e/quiz.html.

MyProgrammingLab™

Programming Exercises

Note

A common error for the exercises in this chapter is that students don't implement the methods to meet the requirements even though the output from the main program is correct. For an example of this type of error see www.cs.armstrong.edu/liang/CommonMethodErrorJava.pdf.

Sections 6.2–6.9

6.1 (*Math: pentagonal numbers*) A pentagonal number is defined as $n(3n-1)/2$ for $n = 1, 2, \ldots$, and so on. Therefore, the first few numbers are 1, 5, 12, 22, ... Write a method with the following header that returns a pentagonal number:

```
public static int getPentagonalNumber(int n)
```

Write a test program that uses this method to display the first 100 pentagonal numbers with 10 numbers on each line.

***6.2** (*Sum the digits in an integer*) Write a method that computes the sum of the digits in an integer. Use the following method header:

```
public static int sumDigits(long n)
```

For example, `sumDigits(234)` returns `9` (2 + 3 + 4). (*Hint*: Use the `%` operator to extract digits, and the `/` operator to remove the extracted digit. For instance, to extract 4 from 234, use `234 % 10` (= 4). To remove 4 from 234, use `234 / 10` (= 23). Use a loop to repeatedly extract and remove the digit until all the digits are extracted. Write a test program that prompts the user to enter an integer and displays the sum of all its digits.

****6.3** (*Palindrome integer*) Write the methods with the following headers

```
// Return the reversal of an integer, i.e., reverse(456) returns 654
public static int reverse(int number)

// Return true if number is a palindrome
public static boolean isPalindrome(int number)
```

Use the `reverse` method to implement `isPalindrome`. A number is a palindrome if its reversal is the same as itself. Write a test program that prompts the user to enter an integer and reports whether the integer is a palindrome.

VideoNote

Reverse an integer

***6.4** (*Display an integer reversed*) Write a method with the following header to display an integer in reverse order:

```
public static void reverse(int number)
```

For example, `reverse(3456)` displays `6543`. Write a test program that prompts the user to enter an integer and displays its reversal.

***6.5** (*Sort three numbers*) Write a method with the following header to display three numbers in increasing order:

```
public static void displaySortedNumbers(
  double num1, double num2, double num3)
```

Write a test program that prompts the user to enter three numbers and invokes the method to display them in increasing order.

*6.6 (*Display patterns*) Write a method to display a pattern as follows:

```
        1
      2 1
    3 2 1
...
n n-1 ... 3 2 1
```

The method header is

```
public static void displayPattern(int n)
```

*6.7 (*Financial application: compute the future investment value*) Write a method that computes future investment value at a given interest rate for a specified number of years. The future investment is determined using the formula in Programming Exercise 2.21.

Use the following method header:

```
public static double futureInvestmentValue(
  double investmentAmount, double monthlyInterestRate, int years)
```

For example, `futureInvestmentValue(10000, 0.05/12, 5)` returns `12833.59`.

Write a test program that prompts the user to enter the investment amount (e.g., 1000) and the interest rate (e.g., 9%) and prints a table that displays future value for the years from 1 to 30, as shown below:

```
The amount invested: 1000 Enter
Annual interest rate: 9 Enter
Years     Future Value
1             1093.80
2             1196.41
...
29           13467.25
30           14730.57
```

6.8 (*Conversions between Celsius and Fahrenheit*) Write a class that contains the following two methods:

```
/** Convert from Celsius to Fahrenheit */
public static double celsiusToFahrenheit(double celsius)

/** Convert from Fahrenheit to Celsius */
public static double fahrenheitToCelsius(double fahrenheit)
```

The formula for the conversion is:

```
fahrenheit = (9.0 / 5) * celsius + 32
celsius = (5.0 / 9) * (fahrenheit - 32)
```

Write a test program that invokes these methods to display the following tables:

Celsius	Fahrenheit	Fahrenheit	Celsius
40.0	104.0	120.0	48.89
39.0	102.2	110.0	43.33
...			
32.0	89.6	40.0	4.44
31.0	87.8	30.0	-1.11

6.9 (*Conversions between feet and meters*) Write a class that contains the following two methods:

```
/** Convert from feet to meters */
public static double footToMeter(double foot)

/** Convert from meters to feet */
public static double meterToFoot(double meter)
```

The formula for the conversion is:

```
meter = 0.305 * foot
foot = 3.279 * meter
```

Write a test program that invokes these methods to display the following tables:

Feet	Meters	Meters	Feet
1.0	0.305	20.0	65.574
2.0	0.610	25.0	81.967
...			
9.0	2.745	60.0	196.721
10.0	3.050	65.0	213.115

6.10 (*Use the* `isPrime` *Method*) Listing 6.7, PrimeNumberMethod.java, provides the `isPrime(int number)` method for testing whether a number is prime. Use this method to find the number of prime numbers less than `10000`.

6.11 (*Financial application: compute commissions*) Write a method that computes the commission, using the scheme in Programming Exercise 5.39. The header of the method is as follows:

```
public static double computeCommission(double salesAmount)
```

Write a test program that displays the following table:

Sales Amount	Commission
10000	900.0
15000	1500.0
...	
95000	11100.0
100000	11700.0

6.12 (*Display characters*) Write a method that prints characters using the following header:

```
public static void printChars(char ch1, char ch2, int
 numberPerLine)
```

This method prints the characters between `ch1` and `ch2` with the specified numbers per line. Write a test program that prints ten characters per line from `1` to `Z`. Characters are separated by exactly one space.

***6.13** (*Sum series*) Write a method to compute the following series:

$$m(i) = \frac{1}{2} + \frac{2}{3} + \ldots + \frac{i}{i+1}$$

Write a test program that displays the following table:

i	m(i)
1	0.5000
2	1.1667
...	
19	16.4023
20	17.3546

***6.14** (*Estimate π*) π can be computed using the following series:

VideoNote
Estimate π

$$m(i) = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots + \frac{(-1)^{i+1}}{2i-1}\right)$$

Write a method that returns `m(i)` for a given `i` and write a test program that displays the following table:

i	m(i)
1	4.0000
101	3.1515
201	3.1466
301	3.1449
401	3.1441
501	3.1436
601	3.1433
701	3.1430
801	3.1428
901	3.1427

***6.15** (*Financial application: print a tax table*) Listing 3.5 gives a program to compute tax. Write a method for computing tax using the following header:

```
public static double computeTax(int status, double taxableIncome)
```

Use this method to write a program that prints a tax table for taxable income from $50,000 to $60,000 with intervals of $50 for all the following statuses:

Taxable Income	Single	Married Joint or Qualifying Widow(er)	Married Separate	Head of a House
50000	8688	6665	8688	7353
50050	8700	6673	8700	7365
...				
59950	11175	8158	11175	9840
60000	11188	8165	11188	9853

Hint: round the tax into integers using `Math.round` (i.e., `Math.round(computeTax(status, taxableIncome)`).

***6.16** (*Number of days in a year*) Write a method that returns the number of days in a year using the following header:

```
public static int numberOfDaysInAYear(int year)
```

Write a test program that displays the number of days in year from 2000 to 2020.

Sections 6.10–6.11

***6.17** (*Display matrix of 0s and 1s*) Write a method that displays an *n*-by-*n* matrix using the following header:

```
public static void printMatrix(int n)
```

Each element is 0 or 1, which is generated randomly. Write a test program that prompts the user to enter `n` and displays an *n*-by-*n* matrix. Here is a sample run:

```
Enter n: 3 Enter
0 1 0
0 0 0
1 1 1
```

****6.18** (*Check password*) Some websites impose certain rules for passwords. Write a method that checks whether a string is a valid password. Suppose the password rules are as follows:

- A password must have at least eight characters.
- A password consists of only letters and digits.
- A password must contain at least two digits.

Write a program that prompts the user to enter a password and displays `Valid Password` if the rules are followed or `Invalid Password` otherwise.

***6.19** (*The `MyTriangle` class*) Create a class named `MyTriangle` that contains the following two methods:

```
/** Return true if the sum of any two sides is
 *  greater than the third side. */
public static boolean isValid(
  double side1, double side2, double side3)
```

```
/** Return the area of the triangle. */
public static double area(
  double side1, double side2, double side3)
```

Write a test program that reads three sides for a triangle and computes the area if the input is valid. Otherwise, it displays that the input is invalid. The formula for computing the area of a triangle is given in Programming Exercise 2.19.

*6.20 (*Count the letters in a string*) Write a method that counts the number of letters in a string using the following header:

```
public static int countLetters(String s)
```

Write a test program that prompts the user to enter a string and displays the number of letters in the string.

*6.21 (*Phone keypads*) The international standard letter/number mapping for telephones is shown in Programming Exercise 4.15. Write a method that returns a number, given an uppercase letter, as follows:

```
int getNumber(char uppercaseLetter)
```

Write a test program that prompts the user to enter a phone number as a string. The input number may contain letters. The program translates a letter (uppercase or lowercase) to a digit and leaves all other characters intact. Here is a sample run of the program:

```
Enter a string: 1-800-Flowers ↵ Enter
1-800-3569377
```

```
Enter a string: 1800flowers ↵ Enter
18003569377
```

**6.22 (*Math: approximate the square root*) There are several techniques for implementing the `sqrt` method in the `Math` class. One such technique is known as the *Babylonian method*. It approximates the square root of a number, `n`, by repeatedly performing a calculation using the following formula:

```
nextGuess = (lastGuess + n / lastGuess) / 2
```

When `nextGuess` and `lastGuess` are almost identical, `nextGuess` is the approximated square root. The initial guess can be any positive value (e.g., `1`). This value will be the starting value for `lastGuess`. If the difference between `nextGuess` and `lastGuess` is less than a very small number, such as `0.0001`, you can claim that `nextGuess` is the approximated square root of `n`. If not, `nextGuess` becomes `lastGuess` and the approximation process continues. Implement the following method that returns the square root of `n`.

```
public static double sqrt(long n)
```

*6.23 (*Occurrences of a specified character*) Write a method that finds the number of occurrences of a specified character in a string using the following header:

```
public static int count(String str, char a)
```

For example, `count("Welcome", 'e')` returns `2`. Write a test program that prompts the user to enter a string followed by a character and displays the number of occurrences of the character in the string.

Sections 6.10–6.12

**6.24 (*Display current date and time*) Listing 2.7, ShowCurrentTime.java, displays the current time. Improve this example to display the current date and time. The calendar example in Listing 6.12, PrintCalendar.java, should give you some ideas on how to find the year, month, and day.

**6.25 (*Convert milliseconds to hours, minutes, and seconds*) Write a method that converts milliseconds to hours, minutes, and seconds using the following header:

```
public static String convertMillis(long millis)
```

The method returns a string as *`hours:minutes:seconds`*. For example, `convertMillis(5500)` returns a string 0:0:5, `convertMillis(100000)` returns a string `0:1:40`, and `convertMillis(555550000)` returns a string `154:19:10`.

Comprehensive

**6.26 (*Palindromic prime*) A *palindromic prime* is a prime number and also palindromic. For example, 131 is a prime and also a palindromic prime, as are 313 and 757. Write a program that displays the first 100 palindromic prime numbers. Display 10 numbers per line, separated by exactly one space, as follows:

```
2 3 5 7 11 101 131 151 181 191
313 353 373 383 727 757 787 797 919 929
...
```

**6.27 (*Emirp*) An *emirp* (prime spelled backward) is a nonpalindromic prime number whose reversal is also a prime. For example, 17 is a prime and 71 is a prime, so 17 and 71 are emirps. Write a program that displays the first 100 emirps. Display 10 numbers per line, separated by exactly one space, as follows:

```
13 17 31 37 71 73 79 97 107 113
149 157 167 179 199 311 337 347 359 389
...
```

**6.28 (*Mersenne prime*) A prime number is called a *Mersenne prime* if it can be written in the form $2^p - 1$ for some positive integer p. Write a program that finds all Mersenne primes with $p \leq 31$ and displays the output as follows:

p	2^p - 1
2	3
3	7
5	31
...	

**6.29 (*Twin primes*) Twin primes are a pair of prime numbers that differ by 2. For example, 3 and 5 are twin primes, 5 and 7 are twin primes, and 11 and 13 are twin primes. Write a program to find all twin primes less than 1,000. Display the output as follows:

```
(3, 5)
(5, 7)
...
```

**6.30 (*Game: craps*) Craps is a popular dice game played in casinos. Write a program to play a variation of the game, as follows:

Roll two dice. Each die has six faces representing values 1, 2, …, and 6, respectively. Check the sum of the two dice. If the sum is 2, 3, or 12 (called *craps*), you lose; if the sum is 7 or 11 (called *natural*), you win; if the sum is another value (i.e., 4, 5, 6, 8, 9, or 10), a point is established. Continue to roll the dice until either a 7 or the same point value is rolled. If 7 is rolled, you lose. Otherwise, you win.

Your program acts as a single player. Here are some sample runs.

```
You rolled 5 + 6 = 11
You win
```

```
You rolled 1 + 2 = 3
You lose
```

```
You rolled 4 + 4 = 8
point is 8
You rolled 6 + 2 = 8
You win
```

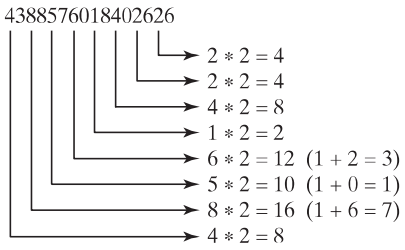
```
You rolled 3 + 2 = 5
point is 5
You rolled 2 + 5 = 7
You lose
```

**6.31 (*Financial: credit card number validation*) Credit card numbers follow certain patterns. A credit card number must have between 13 and 16 digits. It must start with:

- 4 for Visa cards
- 5 for Master cards
- 37 for American Express cards
- 6 for Discover cards

In 1954, Hans Luhn of IBM proposed an algorithm for validating credit card numbers. The algorithm is useful to determine whether a card number is entered correctly or whether a credit card is scanned correctly by a scanner. Credit card numbers are generated following this validity check, commonly known as the *Luhn check* or the *Mod 10 check,* which can be described as follows (for illustration, consider the card number 4388576018402626):

1. Double every second digit from right to left. If doubling of a digit results in a two-digit number, add up the two digits to get a single-digit number.



2. Now add all single-digit numbers from Step 1.

$$4 + 4 + 8 + 2 + 3 + 1 + 7 + 8 = 37$$

3. Add all digits in the odd places from right to left in the card number.

$$6 + 6 + 0 + 8 + 0 + 7 + 8 + 3 = 38$$

4. Sum the results from Step 2 and Step 3.

$$37 + 38 = 75$$

5. If the result from Step 4 is divisible by 10, the card number is valid; otherwise, it is invalid. For example, the number 4388576018402626 is invalid, but the number 4388576018410707 is valid.

Write a program that prompts the user to enter a credit card number as a `long` integer. Display whether the number is valid or invalid. Design your program to use the following methods:

```
/** Return true if the card number is valid */
public static boolean isValid(long number)

/** Get the result from Step 2 */
public static int sumOfDoubleEvenPlace(long number)

/** Return this number if it is a single digit, otherwise,
 * return the sum of the two digits */
public static int getDigit(int number)

/** Return sum of odd-place digits in number */
public static int sumOfOddPlace(long number)

/** Return true if the digit d is a prefix for number */
public static boolean prefixMatched(long number, int d)

/** Return the number of digits in d */
public static int getSize(long d)

/** Return the first k number of digits from number. If the
 * number of digits in number is less than k, return number. */
public static long getPrefix(long number, int k)
```

Here are sample runs of the program: (You may also implement this program by reading the input as a string and processing the string to validate the credit card.)

```
Enter a credit card number as a long integer:
  4388576018410707 Enter
4388576018410707 is valid
```

```
Enter a credit card number as a long integer:
  4388576018402626 Enter
4388576018402626 is invalid
```

**6.32 (*Game: chance of winning at craps*) Revise Exercise 6.30 to run it 10,000 times and display the number of winning games.

**6.33 (*Current date and time*) Invoking `System.currentTimeMillis()` returns the elapsed time in milliseconds since midnight of January 1, 1970. Write a program that displays the date and time. Here is a sample run:

```
Current date and time is May 16, 2012 10:34:23
```

**6.34 (*Print calendar*) Programming Exercise 3.21 uses Zeller's congruence to calculate the day of the week. Simplify Listing 6.12, PrintCalendar.java, using Zeller's algorithm to get the start day of the month.

6.35 (*Geometry: area of a pentagon*) The area of a pentagon can be computed using the following formula:

$$Area = \frac{5 \times s^2}{4 \times \tan\left(\frac{\pi}{5}\right)}$$

Write a method that returns the area of a pentagon using the following header:

```
public static double area(double side)
```

Write a main method that prompts the user to enter the side of a pentagon and displays its area. Here is a sample run:

```
Enter the side: 5.5 Enter
The area of the pentagon is 52.04444136781625
```

*6.36 (*Geometry: area of a regular polygon*) A regular polygon is an n-sided polygon in which all sides are of the same length and all angles have the same degree (i.e., the polygon is both equilateral and equiangular). The formula for computing the area of a regular polygon is

$$Area = \frac{n \times s^2}{4 \times \tan\left(\frac{\pi}{n}\right)}$$

Write a method that returns the area of a regular polygon using the following header:

```
public static double area(int n, double side)
```

Write a main method that prompts the user to enter the number of sides and the side of a regular polygon and displays its area. Here is a sample run:

```
Enter the number of sides: 5 Enter
Enter the side: 6.5 Enter
The area of the polygon is 72.69017017488385
```

6.37 (*Format an integer*) Write a method with the following header to format the integer with the specified width.

```
public static String format(int number, int width)
```

The method returns a string for the number with one or more prefix `0`s. The size of the string is the width. For example, `format(34, 4)` returns `0034` and `format(34, 5)` returns `00034`. If the number is longer than the width, the method

returns the string representation for the number. For example, `format(34, 1)` returns `34`.

Write a test program that prompts the user to enter a number and its width and displays a string returned by invoking `format(number, width)`.

*6.38 (*Generate random characters*) Use the methods in `RandomCharacter` in Listing 6.10 to print 100 uppercase letters and then 100 single digits, printing ten per line.

6.39 (*Geometry: point position*) Programming Exercise 3.32 shows how to test whether a point is on the left side of a directed line, on the right, or on the same line. Write the methods with the following headers:

```
/** Return true if point (x2, y2) is on the left side of the
 *  directed line from (x0, y0) to (x1, y1) */
public static boolean leftOfTheLine(double x0, double y0,
  double x1, double y1, double x2, double y2)

/** Return true if point (x2, y2) is on the same
 *  line from (x0, y0) to (x1, y1) */
public static boolean onTheSameLine(double x0, double y0,
  double x1, double y1, double x2, double y2)

/** Return true if point (x2, y2) is on the
 *  line segment from (x0, y0) to (x1, y1) */
public static boolean onTheLineSegment(double x0, double y0,
  double x1, double y1, double x2, double y2)
```

Write a program that prompts the user to enter the three points for `p0`, `p1`, and `p2` and displays whether `p2` is on the left of the line from `p0` to `p1`, right, the same line, or on the line segment. Here are some sample runs:

```
Enter three points for p0, p1, and p2: 1 1 2 2 1.5 1.5 Enter
(1.5, 1.5) is on the line segment from (1.0, 1.0) to (2.0, 2.0)
```

```
Enter three points for p0, p1, and p2: 1 1 2 2 3 3 Enter
(3.0, 3.0) is on the same line from (1.0, 1.0) to (2.0, 2.0)
```

```
Enter three points for p0, p1, and p2: 1 1 2 2 1 1.5 Enter
(1.0, 1.5) is on the left side of the line
  from (1.0, 1.0) to (2.0, 2.0)
```

```
Enter three points for p0, p1, and p2: 1 1 2 2 1 -1 Enter
(1.0, -1.0) is on the right side of the line
  from (1.0, 1.0) to (2.0, 2.0)
```