Programming Exercises

Chapter 6: Methods

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.

****6.3** (*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.

***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.

6.9 (*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

***6.13** (*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

***6.17** (*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
```

***6.23** (*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.

****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`.

****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
...
```

****6.29** (*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)
...
```

****6.31** (*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.

4388576018402626

$2 * 2 = 4$
$2 * 2 = 4$
$4 * 2 = 8$
$1 * 2 = 2$
$6 * 2 = 12 \ (1 + 2 = 3)$
$5 * 2 = 10 \ (1 + 0 = 1)$
$8 * 2 = 16 \ (1 + 6 = 7)$
$4 * 2 = 8$

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
```

****6.33** (*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
```

6.35 (*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
```

6.37 (*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)`.

Chapter 9: Objects and Classes

9.1 (*The `Rectangle` class*) ~~Following the example of the `Circle` class in Section 9.2~~, design a class named `Rectangle` to represent a rectangle. The class contains:

- Two `double` data fields named `width` and `height` that specify the width and height of the rectangle. The default values are `1` for both `width` and `height`.
- A no-arg constructor that creates a default rectangle.
- A constructor that creates a rectangle with the specified `width` and `height`.
- A method named `getArea()` that returns the area of this rectangle.
- A method named `getPerimeter()` that returns the perimeter.

~~Draw the UML diagram for the class and then implement the class.~~ Write a test program that creates two `Rectangle` objects—one with width `4` and height `40` and the other with width `3.5` and height `35.9`. Display the width, height, area, and perimeter of each rectangle in this order.

***9.3** (*Use the `Date` class*) Write a program that creates a `Date` object, sets its elapsed time to `10000`, `100000`, `1000000`, `10000000`, `100000000`, `1000000000`, `10000000000`, and `100000000000`, and displays the date and time using the `toString()` method, respectively.

***9.5** (*Use the `GregorianCalendar` class*) Java API has the `GregorianCalendar` class in the `java.util` package, which you can use to obtain the year, month, and day of a date. The no-arg constructor constructs an instance for the current date, and the methods `get(GregorianCalendar.YEAR)`, `get(GregorianCalendar.MONTH)`, and `get(GregorianCalendar.DAY_OF_MONTH)` return the year, month, and day. Write a program to perform two tasks:

- Display the current year, month, and day.
- The `GregorianCalendar` class has the `setTimeInMillis(long)`, which can be used to set a specified elapsed time since January 1, 1970. Set the value to `1234567898765L` and display the year, month, and day.

9.7 (*The `Account` class*) Design a class named `Account` that contains:

- A private `int` data field named `id` for the account (default `0`).
- A private `double` data field named `balance` for the account (default `0`).
- A private `double` data field named `annualInterestRate` that stores the current interest rate (default `0`). Assume all accounts have the same interest rate.
- A private `Date` data field named `dateCreated` that stores the date when the account was created.
- A no-arg constructor that creates a default account.
- A constructor that creates an account with the specified id and initial balance.
- The accessor and mutator methods for `id`, `balance`, and `annualInterestRate`.
- The accessor method for `dateCreated`.
- A method named `getMonthlyInterestRate()` that returns the monthly interest rate.
- A method named `getMonthlyInterest()` that returns the monthly interest.
- A method named `withdraw` that withdraws a specified amount from the account.
- A method named `deposit` that deposits a specified amount to the account.

~~Draw the UML diagram for the class and then implement the class~~. (*Hint*: The method `getMonthlyInterest()` is to return monthly interest, not the interest rate. Monthly interest is `balance * monthlyInterestRate`. `monthlyInterestRate` is `annualInterestRate / 12`. Note that `annualInterestRate` is a percentage, e.g., like 4.5%. You need to divide it by 100.)

Write a test program that creates an `Account` object with an account ID of 1122, a balance of $20,000, and an annual interest rate of 4.5%. Use the `withdraw` method to withdraw $2,500, use the `deposit` method to deposit $3,000, and print the balance, the monthly interest, and the date when this account was created.

****9.9** (*Geometry: n-sided regular polygon*) In an *n*-sided regular polygon, all sides have the same length and all angles have the same degree (i.e., the polygon is both equilateral and equiangular). Design a class named `RegularPolygon` that contains:

- A private `int` data field named `n` that defines the number of sides in the polygon with default value `3`.
- A private `double` data field named `side` that stores the length of the side with default value `1`.
- A private `double` data field named `x` that defines the *x*-coordinate of the polygon's center with default value `0`.
- A private `double` data field named `y` that defines the *y*-coordinate of the polygon's center with default value `0`.
- A no-arg constructor that creates a regular polygon with default values.
- A constructor that creates a regular polygon with the specified number of sides and length of side, centered at (`0`, `0`).
- A constructor that creates a regular polygon with the specified number of sides, length of side, and *x*- and *y*-coordinates.
- The accessor and mutator methods for all data fields.
- The method `getPerimeter()` that returns the perimeter of the polygon.
- The method `getArea()` that returns the area of the polygon. The formula for computing the area of a regular polygon is $Area = \dfrac{n \times s^2}{4 \times \tan\left(\dfrac{\pi}{n}\right)}$.

~~Draw the UML diagram for the class and then implement the class.~~ Write a test program that creates three `RegularPolygon` objects, created using the no-arg constructor, using `RegularPolygon(6, 4)`, and using `RegularPolygon(10, 4, 5.6, 7.8)`. For each object, display its perimeter and area.

***9.11** (*Algebra: 2 × 2 linear equations*) Design a class named `LinearEquation` for a 2 × 2 system of linear equations:

$$\begin{aligned} ax + by &= e \\ cx + dy &= f \end{aligned} \qquad x = \frac{ed - bf}{ad - bc} \qquad y = \frac{af - ec}{ad - bc}$$

The class contains:

- Private data fields `a`, `b`, `c`, `d`, `e`, and `f`.
- A constructor with the arguments for `a`, `b`, `c`, `d`, `e`, and `f`.
- Six getter methods for `a`, `b`, `c`, `d`, `e`, and `f`.
- A method named `isSolvable()` that returns true if $ad - bc$ is not 0.
- Methods `getX()` and `getY()` that return the solution for the equation.

~~Draw the UML diagram for the class and then implement the class.~~ Write a test program that prompts the user to enter `a`, `b`, `c`, `d`, `e`, and `f` and displays the result. If $ad - bc$ is 0, report that "The equation has no solution." ~~See Programming Exercise 3.3 for sample runs~~.

****9.13** (*The `Location` class*) Design a class named `Location` for locating a maximal value and its location in a two-dimensional array. The class contains public data fields `row`, `column`, and `maxValue` that store the maximal value and its indices in a two-dimensional array with `row` and `column` as `int` types and `maxValue` as a `double` type.

Write the following method that returns the location of the largest element in a two-dimensional array:

```
public static Location locateLargest(double[][] a)
```

The return value is an instance of `Location`. Write a test program that prompts the user to enter a two-dimensional array and displays the location of the largest element in the array. Here is a sample run:

```
Enter the number of rows and columns in the array: 3 4 ↵Enter
Enter the array:
23.5 35 2 10 ↵Enter
4.5 3 45 3.5 ↵Enter
35 44 5.5 9.6 ↵Enter
The location of the largest element is 45 at (1, 2)
```

Chapter 10: Objects-Oriented Thinking

***10.1** (*The* `Time` *class*) Design a class named `Time`. The class contains:

- The data fields `hour`, `minute`, and `second` that represent a time.
- A no-arg constructor that creates a `Time` object for the current time. (The values of the data fields will represent the current time.)
- A constructor that constructs a `Time` object with a specified elapsed time since midnight, January 1, 1970, in milliseconds. (The values of the data fields will represent this time.)
- A constructor that constructs a `Time` object with the specified hour, minute, and second.
- Three getter methods for the data fields `hour`, `minute`, and `second`, respectively.
- A method named `setTime(long elapseTime)` that sets a new time for the object using the elapsed time. For example, if the elapsed time is `555550000` milliseconds, the hour is `10`, the minute is `19`, and the second is `10`.

~~Draw the UML diagram for the class and then implement the class~~. Write a test program that creates two `Time` objects (using `new Time()` and `new Time(555550000)`) and displays their hour, minute, and second in the format hour:minute:second.

(*Hint*: The first two constructors will extract the hour, minute, and second from the elapsed time. For the no-arg constructor, the current time can be obtained using `System.currentTimeMillis()`, as shown in Listing 2.7, ShowCurrentTime.java.)

10.3 (*The* `MyInteger` *class*) Design a class named `MyInteger`. The class contains:

- An `int` data field named `value` that stores the `int` value represented by this object.
- A constructor that creates a `MyInteger` object for the specified `int` value.
- A getter method that returns the `int` value.
- The methods `isEven()`, `isOdd()`, and `isPrime()` that return `true` if the value in this object is even, odd, or prime, respectively.
- The static methods `isEven(int)`, `isOdd(int)`, and `isPrime(int)` that return `true` if the specified value is even, odd, or prime, respectively.
- The static methods `isEven(MyInteger)`, `isOdd(MyInteger)`, and `isPrime(MyInteger)` that return `true` if the specified value is even, odd, or prime, respectively.
- The methods `equals(int)` and `equals(MyInteger)` that return `true` if the value in this object is equal to the specified value.
- A static method `parseInt(char[])` that converts an array of numeric characters to an `int` value.
- A static method `parseInt(String)` that converts a string into an `int` value.

~~Draw the UML diagram for the class and then implement the class~~. Write a client program that tests all methods in the class.

***10.5** (*Displaying the prime factors*) Write a program that prompts the user to enter a positive integer and displays all its smallest factors in decreasing order. For example, if the integer is `120`, the smallest factors are displayed as `5`, `3`, `2`, `2`, `2`. Use the `StackOfIntegers` class to store the factors (e.g., `2`, `2`, `2`, `3`, `5`) and retrieve and display them in reverse order.

***10.17** (*Square numbers*) Find the first ten square numbers that are greater than `Long.MAX_VALUE`. A square number is a number in the form of n^2. For example, 4, 9, and 16 are square numbers. Find an efficient approach to run your program fast.

***10.19** (*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 $p \leq 100$ and displays the output as shown below. (*Hint*: You have to use `BigInteger` to store the number, because it is too big to be stored in `long`. Your program may take several hours to run.)

```
       p        2^p - 1

       2           3
       3           7
       5          31
...
```

10.21 (*Divisible by* `5` *or* `6`) Find the first ten numbers greater than `Long.MAX_VALUE` that are divisible by `5` or `6`.

****10.23** (*Implement the* `String` *class*) The `String` class is provided in the Java library. Provide your own implementation for the following methods (name the new class `MyString2`):

```
public MyString2(String s);
public int compare(String s);
public MyString2 substring(int begin);
public MyString2 toUpperCase();
public char[] toChars();
public static MyString2 valueOf(boolean b);
```

****10.27** (*Implement the* `StringBuilder` *class*) The `StringBuilder` class is provided in the Java library. Provide your own implementation for the following methods (name the new class `MyStringBuilder1`):

```
public MyStringBuilder1(String s);
public MyStringBuilder1 append(MyStringBuilder1 s);
public MyStringBuilder1 append(int i);
public int length();
public char charAt(int index);
public MyStringBuilder1 toLowerCase();
public MyStringBuilder1 substring(int begin, int end);
public String toString();
```

no star = easy
* = moderate
** = hard
*** = challenging

From Introduction to Java Programming, Comprehensive Version (8th Edition) by Y. Daniel Liang