Programming Exercises

The exercises in this section are optional and do not report to the performance dashboard. Instructors can decide whether to assign these exercises and students can check the correctness of their programs using the Check Exercise tool.

Sections 9.2–9.3

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 data fields named `width` and `height`.
- A constructor that creates a rectangle with the specified `width` and `height`. The default values are `1` and `2` for the `width` and `height`, respectively.
- 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 `35.7`. Display the width, height, area, and perimeter of each rectangle in this order.

Sections 9.4–9.6

9.2 *(The `Stock` class) Design a class named `Stock` to represent a company's stock that contains:*

- A private string data field named `symbol` for the stock's symbol.
- A private string data field named `name` for the stock's name.
- A private float data field named `previousClosingPrice` that stores the stock price for the previous day.

- A private float data field named `currentPrice` that stores the stock price for the current time.
- A constructor that creates a stock with the specified symbol, name, previous price, and current price.
- A getter method for returning the stock name.
- A getter method for returning the stock symbol.
- Getter and setter methods for getting/setting the stock's previous price.
- Getter and setter methods for getting/setting the stock's current price.
- A method named `getChangePercent()` that returns the percentage changed from `previousClosingPrice` to `currentPrice`.

Draw the UML diagram for the class, and then implement the class. Write a test program that creates a `Stock` object with the stock symbol INTC, the name Intel Corporation, the previous closing price of `20.5`, and the new current price of `20.35`, and display the price-change percentage.

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

- A private int data field named `id` for the account.
- A private float data field named `balance` for the account.
- A private float data field named `annualInterestRate` that stores the current interest rate.
- A constructor that creates an account with the specified id (default 0), initial balance (default 100), and annual interest rate (default 0).
- The accessor and mutator methods for `id`, `balance`, and `annualInterestRate`.
- 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 the monthly interest amount, not the interest rate. Use this formula to calculate the monthly interest: `balance * monthlyInterestRate`. `monthlyInterestRate` is `annualInterestRate/12`.

Note that `annualInterestRate` is a percent (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 id, balance, monthly interest rate, and monthly interest.

9.4 *(The `Fan` class) Design a class named `Fan` to represent a fan. The class contains:*

- Three constants named `SLOW`, `MEDIUM`, and `FAST` with the values `1`, `2`, and `3` to denote the fan speed.
- A private int data field named `speed` that specifies the speed of the fan.
- A private Boolean data field named `on` that specifies whether the fan is on (the default is `False`).
- A private float data field named `radius` that specifies the radius of the fan.
- A private string data field named `color` that specifies the color of the fan.
- The accessor and mutator methods for all four data fields.
- A constructor that creates a fan with the specified speed (default `SLOW`), radius (default `5`), color (default `blue`), and on (default `False`).

Draw the UML diagram for the class and then implement the class. Write a test program that creates two `Fan` objects. For the first object, assign the maximum speed, radius `10`, color `yellow`, and turn it on. Assign medium speed, radius `5`, color `blue`, and turn it off for the second object. Display each object's `speed`, `radius`, `color`, and `on` properties.

***9.5** (*Geometry: n-sided regular polygon)* An n-sided regular polygon's sides all have the same length and all of its 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.
- A private float data field named `side` that stores the length of the side.
- A private float data field named `x` that defines the x-coordinate of the center of the polygon with default value `0`.
- A private float data field named `y` that defines the y-coordinate of the center of the polygon with default value `0`.
- A constructor that creates a regular polygon with the specified n (default `3`), side (default `1`), x (default `0`), and y (default `0`).

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

Draw the UML diagram for the class, and then implement the class. Write a test program that creates three `RegularPolygon` objects, created using `RegularPolygon()`, using `RegularPolygon(6, 4)` and `RegularPolygon(10, 4, 5.6, 7.8)`. For each object, display its perimeter and area.

***9.6** (*Algebra: quadratic equations*) Design a class named `QuadraticEquation` for a quadratic equation
$ax^2 + bx + x = 0$. The class contains:

- The private data fields `a`, `b`, and `c` that represent three coefficients.
- A constructor for the arguments for `a`, `b`, and `c`.
- Three getter methods for `a`, `b`, and `c`.
- A method named `getDiscriminant()` that returns the discriminant, which is $b^2 - 4ac$.
- The methods named `getRoot1()` and `getRoot2()` for returning the two roots of the equation using these formulas:
  $r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$ and $r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$

These methods are useful only if the discriminant is nonnegative. Let these methods return `None` if the discriminant is negative.
Draw the UML diagram for the class, and then implement the class. Write a test program that prompts the user to enter values for a, b, and c and displays the result based on the discriminant. If the discriminant is positive, display the two roots. If the discriminant is `0`, display the one root. Otherwise, display "The equation has no roots."

***9.7** (*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:

- The private data fields `a`, `b`, `c`, `d`, `e`, and `f` with getter methods.
- A constructor with the arguments for `a`, `b`, `c`, `d`, `e`, and `f`.
- A method named `isSolvable()` that returns true if $ad - bc$ is not `0`.
- The 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."

***9.8** (*Stopwatch*) Design a class named `StopWatch`. The class contains:

- The private data fields `startTime` and `endTime` with getter methods.
- A constructor that initializes `startTime` with the current time.
- A method named `start()` that resets the `startTime` to the current time.
- A method named `stop()` that sets the `endTime` to the current time.
- A method named `getElapsedTime()` that returns the elapsed time for the stop watch in milliseconds.

Draw the UML diagram for the class, and then implement the class. Write a test program that measures the execution time of adding numbers from 1 to 1,000,000.

****9.9** (*Geometry: intersection*) Suppose two line segments intersect. The two endpoints for the first line segment are (`x1`, `y1`) and (`x2`, `y2`) and for the second line segment are (`x3`, `y3`) and (`x4`, `y4`). Write a program that prompts the user to enter these four endpoints and displays the intersecting point. (Hint: Use the `LinearEquation` class from Programming Exercise 9.7.)

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

- The private data fields `hour`, `minute`, and `second` that represent a time.
- A constructor that constructs a `Time` object that initializes `hour`, `minute`, and `second` using the current time.
- The getter methods for the data fields `hour`, `minute`, and `second`, respectively.
- A method named `setTime(elapseTime)` that sets a new time for the object using the elapsed time in seconds. For example, if the elapsed time is `555550` seconds, the hour is `10`, the minute is `19`, and the second is `12`.

Draw the UML diagram for the class, and then implement the class. Write a test program that creates a `Time` object for the current time and displays its hour, minute, and second. Your program then prompts the user to enter an elapsed time, sets its elapsed time in the `Time` object, and displays its hour, minute, and second.

(Hint: The initializer will extract the hour, minute, and second from the elapsed time. The current elapsed time can be obtained using `time.time()`, as shown in Listing 2.7, ShowCurrentTime.py.)

Sections 9.8–9.9

****9.11** (*The `Point` class*) Design a class named `Point` to represent a point with `x`- and `y`-coordinates. The class contains:

- Two private data fields `x` and `y` that represent the coordinates with getter methods.
- A constructor that constructs a point with specified coordinates, with default point (`0`, `0`).
- A method named `distance` that returns the distance from this point to another point of the `Point` type.
- A method named `isNearBy(p1)` that returns true if point `p1` is close to this point. Two points are close if their distance is less than `5`.
- Implement the `__str__` method to return a string in the form (x, y).

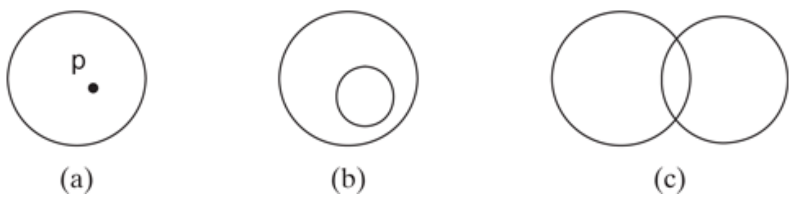
Draw the UML diagram for the class, and then implement the class. Write a test program that prompts the user to enter two points, displays the distance between them, and indicates whether they are near each other.

***9.12** (*Geometry: The `Circle2D` class*) Define the `Circle2D` class that contains:

- Two private float data fields named `x` and `y` that specify the center of the circle with getter/setter methods.
- A private data field `radius` with getter/setter methods.
- A constructor that creates a circle with the specified `x`, `y`, and `radius`. The default values are all `0`.

- A method `getArea()` that returns the area of the circle.
- A method `getPerimeter()` that returns the perimeter of the circle.
- A method `containsPoint(x, y)` that returns `True` if the specified point (`x`, `y`) is inside this circle (see Figure 9.14a).
- A method `contains(circle2D)` that returns `True` if the specified circle is inside this circle (see Figure 9.14b).
- A method `overlaps(circle2D)` that returns `True` if the specified circle overlaps with this circle (see Figure 9.14c).
- Implement the `__contains__(another)` method that returns `True` if this circle is contained in another circle.
- Implement the `__cmp__`, `__lt__`, `__le__`, `__eq__`, `__ne__`, `__gt__`, `__ge__` methods that compare two circles based on their radius.

Figure 9.14



(a) A point is inside the circle. (b) A circle is inside another circle. (c) A circle overlaps another circle.

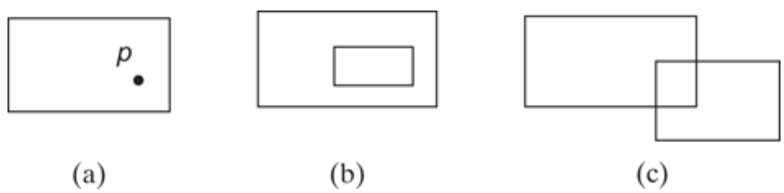
Draw the UML diagram for the class, and then implement the class. Write a test program that prompts the user to enter two circles with x- and y-coordinates and the radius, creates two `Circle2D` objects `c1` and `c2`, displays their areas and perimeters, and displays the result of `c1.containsPoint(c2.getX(), c2.getY())`, `c1.contains(c2)`, `c1.overlaps(c2)`, and the result of `c1 < c2`.

***9.13** (*Geometry: The `Rectangle2D` class*) Define the `Rectangle2D` class that contains:

- Two float data fields named `x` and `y` that specify the center of the rectangle with getter/setter methods. (Assume that the rectangle sides are parallel to *x*- or *y*- axes.)
- The data fields `width` and `height` with getter/setter methods.
- A constructor that creates a rectangle with the specified `x`, `y`, `width`, and `height` with default values 0.

- A method `getArea()` that returns the area of the rectangle.
- A method `getPerimeter()` that returns the perimeter of the rectangle.
- A method `containsPoint(x, y)` that returns `True` if the specified point (`x`, `y`) is inside this rectangle (see Figure 9.15a).
- A method `contains(Rectangle2D)` that returns `True` if the specified rectangle is inside this rectangle (see Figure 9.15b).
- A method `overlaps(Rectangle2D)` that returns `True` if the specified rectangle overlaps with this rectangle (see Figure 9.15c).
- Implement the `__contains__(another)` method that returns `True` if this rectangle is contained in another rectangle.
- Implement the `__cmp__`, `__lt__`, `__le__`, `__eq__`, `__ne__`, `__gt__`, `__ge__` methods that compare two circles based on their areas.

Figure 9.15



(a) A point is inside the rectangle. (b) A rectangle is inside another rectangle. (c) A rectangle overlaps another rectangle.

Draw the UML diagram for the class, and then implement the class. Write a test program that prompts the user to enter two rectangles with center *x*-, *y*-coordinates, width, and height, creates two `Rectangle2D` objects `r1` and `r2`, displays their areas and perimeters, and displays the result of `r1.containsPoint(r2.getX(), r2.getY())`, `r1.contains(r2)`, and `r1.overlaps(r2)`, and the result of `c1 < c2`.

9.14 (*Use the* `Rational` *class*) Write a program that computes the following summation series using the `Rational` class:

$$\frac{1}{2} + \frac{2}{3} + \frac{3}{4} + \ldots + \frac{8}{9} + \frac{9}{10}$$

***9.15** (*Math: The `Complex` class*) Python has the `complex` class for performing complex number arithmetic. In this exercise, you will design and implement your own `Complex` class. Note that the `complex` class in Python is named in lowercase, but our custom `Complex` class is named with C in uppercase.

A complex number is a number of the form $a + bi$, where a and b are real numbers and i is $\sqrt{-1}$. The numbers a and b are known as the real part and the imaginary part of the complex number, respectively. You can perform addition, subtraction, multiplication, and division for complex numbers using the following formulas:

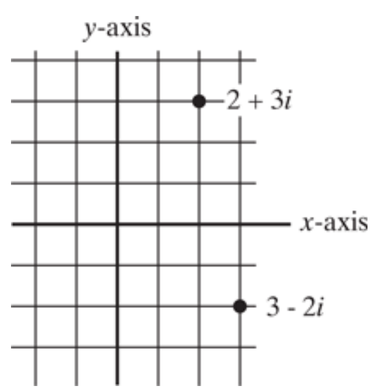
$$(a + bi) + (c + di) = (a + c) + (b + d)i$$
$$(a + bi) - (c + di) = (a - c) + (b - d)i$$
$$(a + bi) * (c + di) = (ac - bd) + (bc + ad)i$$
$$(a + bi) / (c + di) = (ac + bd) / \left(c^2 + d^2\right) + (bc - ad)i / \left(c^2 + d^2\right)$$

You can also obtain the absolute value for a complex number using the following formula:

$$\left|a + bi\right| = \sqrt{a^2 + b^2}$$

(A complex number can be interpreted as a point on a plane by identifying the (a,b) values as the coordinates of the point. The absolute value of the complex number corresponds to the distance of the point to the origin, as shown in Figure 9.16.)

Figure 9.16



Point (2, 3) can be written as a complex number $(2 + 3i)$ and $(3, \ -2)$ as $(3 - 2i)$.

Design a class named `Complex` for representing complex numbers and the methods `__add__`, `__sub__`, `__mul__`, `__truediv__`, and `__abs__` for performing complex-number operations, and override the `__str__` method by returning a string representation for a complex number. The `__str__` method returns `(a + bi)` as a string. If `b` is `0`, it simply returns `a`.

Provide a constructor `Complex(a, b)` to create a complex number $a + bi$ with the default value of `0` for a and b. Also provide the `getRealPart()` and `getImaginaryPart()` methods for returning the real and imaginary parts of the complex number, respectively.

Write a test program that prompts the user to enter two complex numbers and displays the result of their addition, subtraction, multiplication, division, and absolute value.

***9.16** (*Convert decimals to fractions*) Write a program that prompts the user to enter a decimal number and display the number in a fraction. Hint: Read the decimal number as a string, extract the integer part and fractional part from the string, and use the `Rational` class to obtain a rational number for the decimal number.

***9.17** (*Algebra: vertex form equations*) The equation of a parabola can be expressed in either standard form

$\left(y = ax^2 + bx + c = 0\right)$ or vertex form

$\left(y = a(x - h)^2 + k\right)$. Write a program that prompts the user to enter a, b, and c as integers in standard form and displays

$h\left(= \frac{-b}{2a}\right)$ and

$k\left(= \frac{4ac - b^2}{2a}\right)$ in the vertex form.

***9.18** (*Algebra: solve quadratic equations*) Rewrite Programming Exercise 3.1 to obtain imaginary roots if the determinant is less than 0 using the `Complex` class in Programming Exercise 9.15.

***9.19** (*Parse complex numbers*) Add the following nonmember function in the `Complex` class defined in Programming Exercise 9.15.

```
def parseComplexNumber(s):
```

The function returns a `Complex` object from a string that represents a complex number. Here are some examples of parsing complex numbers:

```
c1 = parseComplexNumber("3.5 + 2.23i")
c2 = parseComplexNumber("3.5") # Imaginary part is 0
c3 = parseComplexNumber("-2.23i") # Real part is 0
c4 = parseComplexNumber("3.5—2.23i") # This is K
```

Write a test program that prompts the user to enter two complex numbers as strings and displays their addition. Note that if the real part or imaginary part is 0, it is not displayed. If the imaginary part is 1, the number 1 is not displayed.

***9.20** (*Parse rational numbers*) Write the following function that returns a `Rational` object from a string that represents a rational number.

```
def parseRationalNumber(s)
```

The `Rational` class is defined in Listing 9.13. Here are some examples of parsing rational numbers:

```
r1 = parseRationalNumber("3 / 15")
r2 = parseRationalNumber("-3/15") # This is OK
r3 = parseRationalNumber("34") # Denominator is 1
```

Write a test program that prompts the user to enter two rational numbers as strings and displays their addition.

****9.21** (*Bin packing with largest object first*) The bin packing problem is to pack the objects of various weights into containers. Assume that each container can hold a maximum of 10 pounds. The program uses an algorithm that places an object with the largest weight into the first bin in which it would fit. Your program should prompt the user to enter the weight of each object. The program displays the total number of containers needed to pack the objects and the contents of each container.

Hint: Define a `Bin` class for creating `Bin` objects. Each bin holds some items. You may define the `Bin` class as shown in https://liangpy.pearsoncmg.com/test/Bin.txt.
Does this program produce an optimal solution, that is, finding the minimum number of containers to pack the objects?