COMP 1023 Introduction to Python Programming
Exercises on Introduction to Object-Oriented Programming

COMP 1023 Teaching Team

Department of Computer Science & Engineering
The Hong Kong University of Science and Technology, Hong Kong SAR, China

Exercise 1

Implement a class named `Square` to represent a square. The class contains:

- A private instance variable `side` with a default value 1.
- An accessor method `getSide()` for `side`.
- A method named `getArea()` that returns the area of this square.
- A method named `getPerimeter()` that returns the perimeter.

Write a test program that creates two `Square` objects, one with side of 5.3 and the other with side of 9. Display the side, area, and perimeter of each square in this order.

```
Square 1 - Side: 5.3
Area: 28.09
Perimeter: 21.2
Square 2 - Side: 9
Area: 81
Perimeter: 36
```

Exercise 1 Solutions

```
class Square:
    def __init__(self, side=1):
        self.__side = side

    def getSide(self):
        return self.__side

    def getArea(self):
        return self.__side * self.__side

    def getPerimeter(self):
        return 4 * self.__side

def main():
    square1 = Square(5.3)
    square2 = Square(9)

    print(f"Square 1 - Side: {square1.getSide()}")
    print(f"Area: {square1.getArea()}");  print(f"Perimeter: {square1.getPerimeter()}")

    print(f"Square 2 - Side: {square2.getSide()}")
    print(f"Area: {square2.getArea()}");  print(f"Perimeter: {square2.getPerimeter()}")

if __name__ == "__main__": main()
```

Exercise 2

Implement a class named `Account` that contains:

- A private int instance variable `id` with a default value 0.
- A private float instance variable `balance` with a default value 100.
- A private float instance variable `monthlyInterestRate` with a default value 0.
- The accessor and mutator methods for `id`, `balance`, and `monthlyInterestRate`.
- A method named `getAnnualInterestRate()` that returns the annual interest rate.
- A method named `getAnnualInterest()` that returns the annual 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.

(Hint: The method `getAnnualInterest()` is to return the annual interest amount, not the interest rate. Use this formula to calculate the annual interest: `balance * annualInterestRate`. `annualInterestRate` is `monthlyInterestRate * 12`. Note that monthlyInterestRate is a percent (like 0.375%). 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 0.375%. Use the withdraw method to withdraw $2,500, use the deposit method to deposit $3,000, and print the id, balance, annual interest rate, and annual interest.

```
Account ID: 1122
Balance: $20500.0
Annual Interest Rate: 0.045(4.5%)
Annual Interest: $922.5
```

Exercise 2 Solutions

```
class Account:
    def __init__(self, id=0, balance=100.0, monthlyInterestRate=0.0):
        self.__id = id
        self.__balance = balance
        self.__monthlyInterestRate = monthlyInterestRate

    def getId(self):
        return self.__id
    def getBalance(self):
        return self.__balance
    def getMonthlyInterestRate(self):
        return self.__monthlyInterestRate

    def setId(self, id):
        self.__id = id
    def setBalance(self, balance):
        self.__balance = balance
    def setMonthlyInterestRate(self, monthlyInterestRate):
        self.__monthlyInterestRate = monthlyInterestRate

    def getAnnualInterestRate(self):
        return (self.__monthlyInterestRate / 100) * 12

    def getAnnualInterest(self):
        annualInterestRate = self.getAnnualInterestRate()
        return self.__balance * annualInterestRate
```

```
    def withdraw(self, amount):
        if amount <= self.__balance:
            self.__balance -= amount
            return True
        else:
            print("Insufficient funds for withdrawal")
            return False

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
            return True
        else:
            print("Deposit amount must be positive")
            return False

def main():
    account = Account(1122, 20000.0, 0.375)
    account.withdraw(2500)
    account.deposit(3000)
    print(f"Account ID: {account.getId()}")
    print(f"Balance: ${account.getBalance()}")
    print(f"Annual Interest Rate: {account.getAnnualInterestRate()}\
 ({account.getAnnualInterestRate()*100}%)")
    print(f"Annual Interest: ${account.getAnnualInterest()}")

if __name__ == "__main__": main()
```

Exercise 3

Implement a class named `QuadraticEquation` for a quadratic equation $ax^2 + bx + c = 0$. The class contains:

- The private instance variables `a`, `b`, `c` that represent three coefficients.
- The initializer with the arguments for `a`, `b`, and `c`.
- The three accessors 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 formula:

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \text{and} r_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

  These methods are useful only if the discriminant is non-negative. Let these methods return None if the discriminant is negative.

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".

```
Enter a: 13.5
Enter b: 45.2
Enter c: 12.4
The roots are -0.3014932803673518 and -3.046664867780797
```

Exercise 3 Solutions

```
import math

class QuadraticEquation:
    def __init__(self, a, b, c):
        self.__a = a
        self.__b = b
        self.__c = c

    def getA(self):
        return self.__a
    def getB(self):
        return self.__b
    def getC(self):
        return self.__c

    def getDiscriminant(self):
        return self.__b**2 - 4 * self.__a * self.__c

    def getRoot1(self):
        discriminant = self.getDiscriminant()
        if discriminant >= 0:
            return (-self.__b + math.sqrt(discriminant)) / (2 * self.__a)
        else:
            return None
```

```
    def getRoot2(self):
        discriminant = self.getDiscriminant()
        if discriminant >= 0:
            return (-self.__b - math.sqrt(discriminant)) / (2 * self.__a)
        else:
            return None

def main():
    a = float(input("Enter a: "))
    b = float(input("Enter b: "))
    c = float(input("Enter c: "))

    equation = QuadraticEquation(a, b, c)
    discriminant = equation.getDiscriminant()

    if discriminant > 0:
        root1 = equation.getRoot1()
        root2 = equation.getRoot2()
        print(f"The roots are {root1} and {root2}")
    elif discriminant == 0:
        root = equation.getRoot1()
        print(f"The root is {root}")
    else:
        print("The equation has no real roots")

if __name__ == "__main__": main()
```

Exercise 4

Implement a class named `LinearEquation` for a 2×2 system of linear equations:

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

- The private instance variables `a`, `b`, `c`, `d`, `e`, and `f` with accessor methods.
- The initializer 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 named `getX()` and `getY()` that returns the solution for the equation.

Write a test program that prompts the user to enter values for `a`, `b`, `c`, `d`, `e`, and `f` and displays the result. If $ad - bc$ is 0, report that "The equation has no solution".

```
Enter a, b, c, d, e, f: 9.0 4.0 3.0 -5.0 -6.0 -21.0
x is -2.0 and y is 3.0
```

Exercise 4 Solutions

```
class LinearEquation:
    def __init__(self, a, b, c, d, e, f):
        self.__a = a; self.__b = b; self.__c = c
        self.__d = d; self.__e = e; self.__f = f

    def getA(self): return self.__a
    def getB(self): return self.__b
    def getC(self): return self.__c
    def getD(self): return self.__d
    def getE(self): return self.__e
    def getF(self): return self.__f

    def isSolvable(self):
        return (self.__a * self.__d - self.__b * self.__c) != 0

    def getX(self):
        if self.isSolvable():
            numerator = self.__e * self.__d - self.__b * self.__f
            denominator = self.__a * self.__d - self.__b * self.__c
            return numerator / denominator
        else:
            return None
```

```
    def getY(self):
        if self.isSolvable():
            numerator = self.__a * self.__f - self.__e * self.__c
            denominator = self.__a * self.__d - self.__b * self.__c
            return numerator / denominator
        else:
            return None

def main():
    input_values = input("Enter a, b, c, d, e, f: ").split()

    a = float(input_values[0]); b = float(input_values[1]); c = float(input_values[2])
    d = float(input_values[3]); e = float(input_values[4]); f = float(input_values[5])

    equation = LinearEquation(a, b, c, d, e, f)

    if equation.isSolvable():
        x = equation.getX()
        y = equation.getY()
        print(f"x is {x} and y is {y}")
    else:
        print("The equation has no solution")

if __name__ == "__main__": main()
```

Exercise 4

Implement a class named `Point` to represent a point with x- and y-coordinates. The class contains:

- Two private instance variables `x` and `y` that represent the coordinates with accessor methods.
- The initializer with the arguments for `y` and `y`, with default values 0, 0.
- A method named `distance` that returns the distance from this point to an other point of the Point type.
- A method named `isNearBy(p)` that returns true if point p is close to this point. Two points are close if their distance is less than 5.

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.

```
Enter the x-coordinate of point1: 9.0
Enter the y-coordinate of point1: 4.0
Enter the x-coordinate of point2: 3.0
Enter the y-coordinate of point2: -5.0
The distance between the two points is 10.816653826391969
The two points are not near to each other
```

Exercise 5 Solutions

```
import math

class Point:
    def __init__(self, x=0, y=0):
        self.__x = x
        self.__y = y

    def getX(self):
        return self.__x

    def getY(self):
        return self.__y

    def distance(self, other_point):
        dx = self.__x - other_point.getX()
        dy = self.__y - other_point.getY()
        return math.sqrt(dx**2 + dy**2)

    def isNearBy(self, p):
        return self.distance(p) < 5
```

```
def main():
    x1 = float(input("Enter the x-coordinate of point1: "))
    y1 = float(input("Enter the y-coordinate of point1: "))

    x2 = float(input("Enter the x-coordinate of point2: "))
    y2 = float(input("Enter the y-coordinate of point2: "))

    point1 = Point(x1, y1)
    point2 = Point(x2, y2)

    distance = point1.distance(point2)

    print(f"The distance between the two points is {distance}")

    if point1.isNearBy(point2):
        print("The two points are near to each other")
    else:
        print("The two points are not near to each other")

if __name__ == "__main__": main()
```