Homework #11

01286121 Computer Programming

Software Engineering Program,

Department of Computer Engineering,

School of Engineering, KMITL

By

66011149 Phatthadon Sornplang

1.)

```
import time


class Clock:
   def __init__(self, hh=0, mm=0, ss=0):
      self.hh = hh
      self.mm = mm
      self.ss = ss

   def set_time(self, hh, mm, ss):
      self.hh = hh
      self.mm = mm
      self.ss = ss

   def run(self):
      while True:
         time.sleep(1)
         self.increment_time()
         print(self.get_formatted_time())

   def run_time(self):
      self.ss += 1
      if self.ss >= 60:
         self.ss = 0
         self.mm += 1
      if self.mm >= 60:
         self.mm = 0
         self.hh += 1
      if self.hh >= 24:
         self.hh = 0

   def get_formatted_time(self):
      return f"{self.hh:02}:{self.mm:02}:{self.ss:02}"

class AlarmClock(Clock):
   def __init__(self, hh=0, mm=0, ss=0):
      super().__init__(hh, mm, ss)
      self.alarm_hh = 0
      self.alarm_mm = 0
      self.alarm_ss = 0
      self.alarm_setting = False

   def setAlarmTime(self, hh, mm, ss):
      self.alarm_hh = hh
      self.alarm_mm = mm
      self.alarm_ss = ss

   def alarm_on(self):
      self.alarm_setting = True
```

```
    def alarm_off(self):
        self.alarm_setting = False

    def run(self):
        while True:
            time.sleep(1)
            self.run_time()
            print(self.get_formatted_time(), "Alarm : ", self.formatted_alarm_time())

            if self.alarm_setting and self.get_alarm_time():
                print("ALARM!")
                break

    def formatted_alarm_time(self):
        return f"{self.alarm_hh:02}:{self.alarm_mm:02}:{self.alarm_ss:02}"

    def get_alarm_time(self):
        return self.hh == self.alarm_hh and self.mm == self.alarm_mm and self.ss == self.alarm_ss

clock = AlarmClock(0, 1, 55)
clock.setAlarmTime(0, 2, 5)
clock.alarm_on()
clock.run()
```

2.)

```
import turtle
import math



def RobotBattle():
    robotList = []
    while True:
        turtle.clear()
        for robot in robotList:
            robot.draw()


        print("==== Robots ====")
        i = 0
        for robot in robotList:
            print(i, ": ", end="")
            robot.displayStatus()
            i += 1
        print()

        choice = input("Enter which robot to order, 'c' to create new robot, 'q' to quit: ")

        if choice == "q":
            break
        elif choice == "c":
            print("Enter which type of robots to create")
            robotType = input("'r' for Robot, 'm' for MedicBot, 's' for StrikerBot: ")

            if robotType == "r":
                newRobot = Robot()
            elif robotType == "m":
                newRobot = MedicBot()
            elif robotType == "s":
                newRobot = StrikerBot()

            robotList.append(newRobot)
        else:
            n = int(choice)
            robotList[n].command(robotList)

        robotList = [robot for robot in robotList if robot.health > 0]

class Robot:
    def __init__(self):
        self.energy = 100
        self.health = 100
        self.x = 100
        self.y = 100

    def move(self, newX, newY):
        if self.energy > 0:
            self.x = newX
            self.y = newY
            self.energy -= 10
        if self.energy < 0:
            self.energy = 0

    def draw(self):
        turtle.penup()
        turtle.goto(self.x, self.y - 15)
        turtle.pendown()
        turtle.circle(15)
        turtle.penup()

    def displayStatus(self):
        print(f"x={self.x}, y={self.y}, energy={self.energy}, health={self.health}")

    def command(self, robotList):
        newX = int(input("Enter new x-coordinate for the robot: "))
        newY = int(input("Enter new y-coordinate for the robot: "))
        self.move(newX, newY)



class MedicBot(Robot):
    def draw(self):
        super().draw()

        turtle.color("red")
        crossSize = 10

        turtle.penup()
        turtle.goto(self.x, self.y + crossSize/2)
        turtle.pendown()
        turtle.goto(self.x, self.y - crossSize/2)
        turtle.penup()
        turtle.goto(self.x - crossSize/2, self.y)
        turtle.pendown()
        turtle.goto(self.x + crossSize/2, self.y)

        turtle.penup()
        turtle.color("black")

    def heal(self, r):
        distance = math.sqrt((self.x - r.x)**2 + (self.y - r.y)**2)
        if self.energy >= 20 and distance <= 10:
            self.energy -= 20
            r.health += 10

    def command(self, robotList):
        action = input("Do you want to move the MedicBot or heal another robot? Enter 'move' or 'heal': ").lower()

        if action == "move":
            newX = int(input("Enter new x-coordinate for the MedicBot: "))
            newY = int(input("Enter new y-coordinate for the MedicBot: "))
            self.move(newX, newY)
        elif action == "heal":
            for i, robot in enumerate(robotList):
                print(f"{i}: Robot at x={robot.x}, y={robot.y}")

            target = int(input("Which robot do you want to heal? Enter the robot's number: "))
            if 0 <= target < len(robotList):
                self.heal(robotList[target])
            else:
                print("Invalid choice!")
class StrikerBot(Robot):
    def __init__(self):
        super().__init__()
        self.missile = 5

    def draw(self):
        super().draw()

        squareSide = 20
        diagonal = squareSide * (2**0.5)

        turtle.penup()
        turtle.goto(self.x, self.y + diagonal / 2)
        turtle.setheading(-45)
        turtle.pendown()

        for _ in range(4):
            turtle.forward(squareSide)
            turtle.right(90)

        turtle.penup()
        turtle.setheading(0)
        turtle.color("black")

    def strike(self, r):
        distance = math.sqrt((self.x - r.x)**2 + (self.y - r.y)**2)
        if self.energy >= 20 and self.missile > 0 and distance <= 10:
            self.energy -= 20
            self.missile -= 1
            r.health -= 50

    def displayStatus(self):
        super().displayStatus()
        print(f"missile={self.missile}")

    def command(self, robotList):
        action = input("Do you want to move the StrikerBot or strike another robot? Enter 'move' or 'strike': ").lower()
```

```
        if action == "move":
            newX = int(input("Enter new x-coordinate for the StrikerBot: "))
            newY = int(input("Enter new y-coordinate for the StrikerBot: "))
            self.move(newX, newY)
        elif action == "strike":
            for i, robot in enumerate(robotList):
                print(f"{i}: Robot at x={robot.x}, y={robot.y}")

            target = int(input("Which robot do you want to strike? Enter the robot's number: "))
            if 0 <= target < len(robotList):
                self.strike(robotList[target])
            else:
                print("Invalid choice!")

RobotBattle()
```

3.)

```
 import turtle as t

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

class Rectangle2D:
    def __init__(self, x=0, y=0, width=0, height=0):
        self.x = x
        self.y = y
        self.width = width
        self.height = height

    def get_center(self):
        return Point(self.x, self.y)
    def __str__(self):
        return f"Centered at ({self.x}, {self.y}) with width {self.width} and height {self.height}"

def get_x(point):
    return point.x

def get_y(point):
    return point.y

def getRectangle(points):
    min_x = min(points, key=get_x).x
    max_x = max(points, key=get_x).x
    min_y = min(points, key=get_y).y
    max_y = max(points, key=get_y).y

    center_x = (min_x + max_x) / 2
    center_y = (min_y + max_y) / 2
    width = max_x - min_x
    height = max_y - min_y

    return Rectangle2D(center_x, center_y, width, height)


def drawRectangle(rectangle):
    topLeftX = rectangle.x - rectangle.width / 2
    topLeftY = rectangle.y + rectangle.height / 2
    t.penup()
    t.setpos(topLeftX, topLeftY)
    t.pendown()
    for _ in range(2):
        t.forward(rectangle.width)
        t.right(90)
        t.forward(rectangle.height)
        t.right(90)


def drawPoint(point):
    t.penup()
    t.setpos(point.x, point.y)
    t.pendown()
    t.dot(5)


def main():
    points = []


    try:
        coords = list(map(float, input("Enter the points as x1 y1 x2 y2 ... : ").split()))
        points = [Point(coords[i], coords[i+1]) for i in range(0, len(coords), 2)]
    except ValueError:
        print("Invalid input. Please enter coordinates as pairs of x y.")



    screen = t.Screen()
    t.speed(0)


    if points:
        bounding_rectangle = getRectangle(points)
        print("Bounding Rectangle:")
        print(f"X: {bounding_rectangle.x}, Y: {bounding_rectangle.y}")
        print(f"Width: {bounding_rectangle.width}, Height: {bounding_rectangle.height}")


        drawRectangle(bounding_rectangle)
        for point in points:
            drawPoint(point)
        t.done()
    else:
        print("No points input.")
```

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

4.)

```
 import turtle
import abc



class Char(metaclass = abc.ABCMeta):


    @abc.abstractmethod
    def draw(self, x, y):
        pass


    @abc.abstractmethod
    def getWidth(self):
        pass


class Char0(Char):
    def draw(self, x, y):
        turtle.penup()
        turtle.goto(x, y)
        turtle.write('0')


    def getWidth(self):
        return 14


class Char1(Char):
    def draw(self, x, y):
        turtle.penup()
        turtle.goto(x, y)
        turtle.write('1')


    def getWidth(self):
        return 10
```

```
class Char2(Char):
    def draw(self, x, y):
        turtle.penup()
        turtle.goto(x, y)
        turtle.write('2')

    def getWidth(self):
        return 14

class Char3(Char):
    def draw(self, x, y):
        turtle.penup()
        turtle.goto(x, y)
        turtle.write('3')

    def getWidth(self):
        return 14

class Char4(Char):
    def draw(self, x, y):
        turtle.penup()
        turtle.goto(x, y)
        turtle.write('4')

    def getWidth(self):
        return 14

class Char5(Char):
    def draw(self, x, y):
        turtle.penup()
        turtle.goto(x, y)
        turtle.write('5')
    def getWidth(self):
        return 14

class Char6(Char):
    def draw(self, x, y):
        turtle.penup()
        turtle.goto(x, y)
        turtle.write('6')

    def getWidth(self):
        return 14

class Char7(Char):
    def draw(self, x, y):
        turtle.penup()
        turtle.goto(x, y)
        turtle.write('7')

    def getWidth(self):
        return 14

class Char8(Char):
    def draw(self, x, y):
        turtle.penup()
        turtle.goto(x, y)
        turtle.write('8')

    def getWidth(self):
        return 14

class Char9(Char):
    def draw(self, x, y):
        turtle.penup()
        turtle.goto(x, y)
        turtle.write('9')

    def getWidth(self):
        return 14



num_char_map = {
    '0': Char0(),
    '1': Char1(),
    '2': Char2(),
    '3': Char3(),
    '4': Char4(),
    '5': Char5(),
    '6': Char6(),
    '7': Char7(),
    '8': Char8(),
    '9': Char9()
}

def drawNum(x):
    x_str = str(x)
    x_position = 0
    for digit in x_str:
        char_obj = num_char_map[digit]
        char_obj.draw(x_position, 0)
        x_position += char_obj.getWidth()

drawNum(1234567890)
turtle.done()
```

5.)

```
import abc

class stationarygood(metaclass = abc.ABCMeta):
    def __init__(self, name, price):
        self.name = name
        self.price = price
    @abc.abstractmethod
    def get_cost(self):
        pass


class Magazine(stationarygood):
    def __init__(self, name, price):
        super().__init__(name, price)
    def get_cost(self):
        return self.price

class Book(stationarygood):
    def __init__(self, name, price):
        super().__init__(name, price)
    def get_cost(self):
        return self.price * 0.9

class Ribbon(stationarygood):
    def __init__(self, name,price, length):
        self.length = length
        super().__init__(name,price)
    def get_cost(self):
        return self.price * self.length

def TotalCost(basket):
    total_cost = 0
    for item in basket:
        total_cost += item.get_cost()
    return total_cost
```

```
magazine = Magazine("Computer World", 70)

book = Book("Windows 7 for Beginners", 200)

ribbon = Ribbon("Blue Ribbon",5, 10)



print(f"Total cost:{TotalCost([magazine] * 3 + [book] * 2 + [ribbon])} Bahts")
```