1811ICT/2807ICT/7001ICT Programming Principles Workshop 10

School of Information and Communication Technology

Griffith University

|  |  |
| --- | --- |
| Goals | This workshop focusses on everything in the course up to files. |
| When | Workshops from Friday 27 May to Thursday 2 June |
| Marks | 3 |
| Due | There are no pre-workshop tasks this week.  Workshop programming problems by 11:59pm on 5 June |

# Before your workshop class:

* Read all of this document.
* Review the lecture notes sections 1 to 25.
* **There are no pre-workshop tasks this week**.

# Workshop activities

## Class design

*Case study:* A Go Card account maintains a balance that may be spent on public transport. Users may request a statement that shows all transactions. The only transactions are to top up the account with some positive number of dollars, and to take a ride costing some dollars and cents.

The goal for this exercise is to develop a class for a Go Card Account. The class will be tested by a program that simulates transactions, like this:

|  |
| --- |
| Creating account. Input initial balance: 100  ? r 3.50  ? r 10.90  ? b  Balance = $85.60  ? t 20  ? x gghhg  Bad command.  ? t  Bad command.  ? q  Statement:  event amount ($) balance ($)  Initial balance 100.00  Ride 3.50 96.50  Ride 10.90 85.60  Top up 20.00 105.60  Final balance 105.60 |

where:

* r *number* simulates a ride costing *number* dollars;
* t *number* simulates a top up of *number* dollars;
* b requests the current balance; and
* q ends input and prints a statement.

Bad inputs are to be reported and ignored.

Let us consider the design for a class that represents a Go Card account. To design a class, we consider what services the object(s) must provide (its methods), and what data needs to be stored in the object(s) to support those services. Questions:

* What is a good name for a class that represents a Go Card account?
  + Be descriptive of what the class represents. Don’t include the word “class” in the name.
* What services should be provided?
  + A constructor (\_\_init\_\_) is required to set up the account with an initial balance.
  + It needs to record the amount each ride costs. A method that accepts the amount as a parameter is required.
  + It needs to record the amount for each top-up. A method that accepts the amount as a parameter is required.
  + It needs to be able to report the current balance at any time. A method that returns this is required.
  + A method is required print out a statement of all of the transactions.

We can see from the output of the proposed program that the class needs to store the details of every transaction in order.

* What data is required to be stored in the object to enable those services?
  + So that a method can return the current balance at any time, it would be useful have a field for the current balance.
  + So that the full statement can be printed, the object must store the amount of each transaction, in order. What data type can grow and keep multiple values in the order they are added?

## Problem 1

*Problem:* Implement the program described above, leaving out the printing of a full statement at the end.

*Answer:* Submit your code as problem1.py and insert a screenshot of your program output for the following scenario:

|  |
| --- |
| Creating account. Input initial balance: 200  ? r 2.35  ? r 1.30  ? b  ? r 9.45  ? t 10  ? q |

***Copy your code here***

# Matthew Prendergast

# 1st June, 2022 - Problem 1 (Workshop - Week 10)

class GoCard:

    """The Go Card gives a user the ability to add credit to be used on public transport"""

    def \_\_init\_\_(self, balance):

      """Creates the Go Card with an initial balance"""

      self.open\_balance = balance

      self.balance = balance

      self.transactions = [balance]

    def topUp(self, amount):

        """Tops up the balance and records the transaction"""

        self.balance += amount

        self.transactions.append(amount)

    def ride(self, amount):

        """Records a ride in the transactions and decreases the balance"""

        self.balance -= amount

        self.transactions.append(-amount)

    def balance(self):

        """Returns the current balance"""

        return self.balance

    def statement(self):

        return [self.open\_balance, self.balance, self.transactions]

def checkFloat(num):

    """This function checks to see if a string value can be converted to a float value.

        If the string value is a float: RETURNS TRUE

        If the string value is not a float: RETURNS FALSE"""

    try:

        float(num)

        return True

    except:

        return False

# Prompt the user to enter the new balance for the Go Card.

new\_balance = float(input("Creating account. Input initial balance: "))

# Create the new Go Card object.

new\_card = GoCard(new\_balance)

# Prompt the user for their next command.

command = input("? ")

# Process the command

while command != "q":

    # Split the command into a list and change all number values to floats.

    command\_list = command.split()

    for i in range(len(command\_list)):

        if checkFloat(command\_list[i]):

            command\_list[i] = float(command\_list[i])

    # Top up the Go Card functionality

    if len(command\_list) == 2 and command\_list[0] == "t" and command\_list[1] > 0:

        new\_card.topUp(command\_list[1])

    # Record a ride

    elif len(command\_list) == 2 and command\_list[0] == "r" and command\_list[1] > 0:

        new\_card.ride(command\_list[1])

    # Print the balance

    elif len(command\_list) == 1 and command\_list[0] == "b":

        print(f"Balance = ${new\_card.balance:.2f}")

    # Inform the user of their bad command.

    else:

        print("Bad command.")

    command = input("? ")

***Insert your screenshots here***

**Text

Description automatically generated**

## Problem 2

*Problem:* Implement the program described above, including the printing of a full statement at the end.

*Answer:* Submit your code as problem2.py and insert a screenshot of your program output for the scenario described in Problem 1.

***Copy your code here***

# Matthew Prendergast

# 1st June, 2022 - Problem 2 (Workshop - Week 10)

class GoCard:

    """The Go Card gives a user the ability to add credit to be used on public transport"""

    def \_\_init\_\_(self, balance):

      """Creates the Go Card with an initial balance"""

      self.open\_balance = balance

      self.balance = balance

      self.transactions = [balance]

    def topUp(self, amount):

        """Tops up the balance and records the transaction"""

        self.balance += amount

        self.transactions.append(amount)

    def ride(self, amount):

        """Records a ride in the transactions and decreases the balance"""

        self.balance -= amount

        self.transactions.append(-amount)

    def balance(self):

        """Returns the current balance"""

        return self.balance

    def statement(self):

        return [self.open\_balance, self.balance, self.transactions]

def checkFloat(num):

    """This function checks to see if a string value can be converted to a float value.

        If the string value is a float: RETURNS TRUE

        If the string value is not a float: RETURNS FALSE"""

    try:

        float(num)

        return True

    except:

        return False

# Prompt the user to enter the new balance for the Go Card.

new\_balance = float(input("Creating account. Input initial balance: "))

# Create the new Go Card object.

new\_card = GoCard(new\_balance)

# Prompt the user for their next command.

command = input("? ")

# Process the command

while command != "q":

    # Split the command into a list and change all number values to floats.

    command\_list = command.split()

    for i in range(len(command\_list)):

        if checkFloat(command\_list[i]):

            command\_list[i] = float(command\_list[i])

    # Top up the Go Card functionality

    if len(command\_list) == 2 and command\_list[0] == "t" and command\_list[1] > 0:

        new\_card.topUp(command\_list[1])

    # Record a ride

    elif len(command\_list) == 2 and command\_list[0] == "r" and command\_list[1] > 0:

        new\_card.ride(command\_list[1])

    # Print the balance

    elif len(command\_list) == 1 and command\_list[0] == "b":

        print(f"Balance = ${new\_card.balance:.2f}")

    # Inform the user of their bad command.

    else:

        print("Bad command.")

    command = input("? ")

# Print the statement after quitting.

statement = new\_card.statement()

print("Statement:")

print("{:20} {:>20} {:>20}".format("event", "amount ($)", "balance ($)"))

print("{:20} {:>20} {:>20.2f}".format("Initial balance", "", statement[0]))

print\_balance = statement[0]

for i in range(1, len(statement[2])):

    if statement[2][i] < 0:

        print\_balance += statement[2][i]

        print("{:20} {:>20.2f} {:>20.2f}".format("Ride", abs(statement[2][i]), print\_balance))

    if statement[2][i] > 0:

        print\_balance += statement[2][i]

        print("{:20} {:>20.2f} {:>20.2f}".format("Top up", statement[2][i], print\_balance))

print("{:20} {:>20} {:>20.2f}".format("Final balance", "", statement[1]))

***Insert your screenshots here***

*A screenshot of a computer

Description automatically generated with medium confidence*

# Submission and marking

There are no pre-workshop tasks this week.

For workshop tasks, please submit this document with copied codes and inserted screenshots using the provided submission link in the course website. Students get 3 marks if they complete both problems correctly, or 2 marks if they complete problem 1 correctly, or 1 mark if they partially complete problem 1, or 0 marks without any attempt.