Assignment 12

Classes and objects

LILLEBAELT ACADEMY OF
PROFESSIONAL HIGHER EDUCATION

Author
Martin Grønholdt
mart80c7@edu.eal.dk

Monday 20 February 2017

Table of Contents

Introduction

The programs in this hand-in is an example of using classes in Python, the getter/setter patterns is implemented throughout.

All files for this hand in are available at:
https://github.com/deadbok/eal_programming/tree/master/ass12

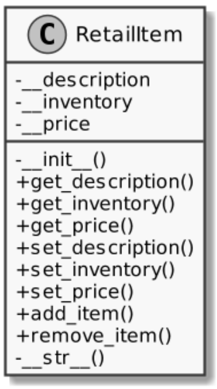
Error handling

All programs that requests user input, handle bad input by asking the user, to use only the correct data type.

Generally all error conditions lead to an error message where after the program exits or continues depending on the severity of the error.

5. RetailItem Class

This program tests a RetailItem class, that holds data for an item in a retail store.



UML representation of the RetailItem class.

prog5.py

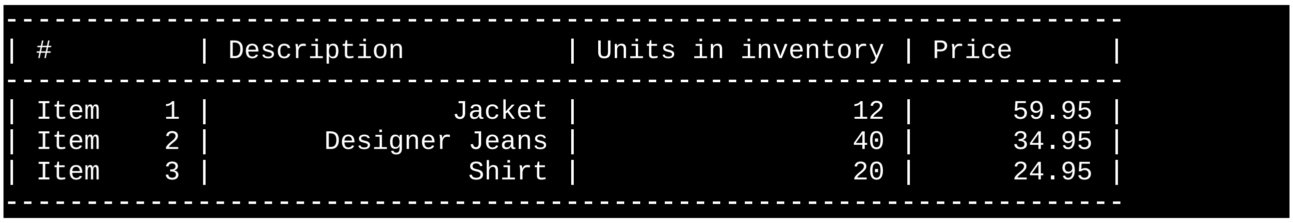
```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
"""
Name: Program  5 "RetailItem Class"
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (2017-02-19)

Program for entering details in to a ProductionWorker class.
"""
from ass12.retailitem import RetailItem


def main():
    """
    Program main entry point.
    """
    items = list()
    # Create 3 RetialItems in the list.
    items.append(RetailItem('Jacket', 12, 59.95))
    items.append(RetailItem('Designer Jeans', 40, 34.95))
    items.append(RetailItem('Shirt', 20, 24.95))

    # Print the items in a table, first the header.
    print('\n-----------------------------------------------------------' +
          '-----------')
    print('| #          | Description          | Units in inventory |' +
          ' Price      |')
    print('-----------------------------------------------------------' +
          '-----------')
    # Print the items.
    for i in range(len(items)):
        item = items[i]
        print('| Item {:4} | {:>20} |'.format(i + 1, item.get_description()),
              end='')
        print(' {:18.0f} | {:10.2f} |'.format(item.get_inventory(),
                                                item.get_price()))
    # Close the table.
    print('------------------------------------------------------------' +
          '----------')


# Run this when invoked directly
if __name__ == '__main__':
    main()
```

retailitem.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
"""
Name: Program  5 "RetailItem Class"
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (2017-02-19)

Class that holds data about an item in a retail store.
"""


class RetailItem:
    """
    RetailItem class.
    """

    def __init__(self, description='None', inventory=0, price=0):
        """
        Initialise the retail item.

        :param description: The description of the item.
        :param inventory: The number of items in the inventory-
        :param price: The price of the item.
        """
        self.__description = description
        self.__inventory = inventory
        self.__price = price

    def get_description(self):
        """
        Get the item desciption.

        :return: String with item description.
        """
        return self.__description

    def get_inventory(self):
        """
        Get the number of items in the inventory.

        :return: The number of items.
        """
        return self.__inventory

    def get_price(self):
        """
        Get the price of the item.

        :return: The price of the item.
        """
        return self.__price

    def set_description(self, description):
        """
        Set the item description.
        :param description: The item description.
        """
        self.__description = description

    def set_inventory(self, inventory):
        """
        Set the number of items in the inventory.

        :param inventory: The number of items.
        """
        self.__inventory = inventory

    def set_price(self, price):
        """
        Set the price of the item.

        :param price: The proce of the item.
        """
        self.__price = price

    def add_item(self, description=None):
        """
        Helper function to add a new item to the inventory.
        If a description is supplied, the item description must match this
        for the item to get added.

        :param description: Item description.
        :return: True if added.
        """
        ret = False
        if description is None:
            self.__inventory += 1
            ret = True
        else:
            if description == self.get_description():
                self.__inventory += 1
                ret = True

        return ret

    def remove_item(self, description=None):
        """
        Helper function to remove an item.
        If a description is supplied, the item description must match this
        for the item to get removed.

        :param description: Item description.
        :return: True if removed.
        """
        ret = False
        if description is None:
            self.__inventory -= 1
            ret = True
        else:
            if description == self.get_description():
                self.__inventory -= 1
                ret = True

        return ret
```

```python
    def __str__(self):
        """
        Return a string that is directly usable when printing the entry.

        :return: string
        """
        return ('{:20}\t'.format(self.get_description()) +
                '{:8.2f}\t'.format(self.get_price())
                )
```

Result

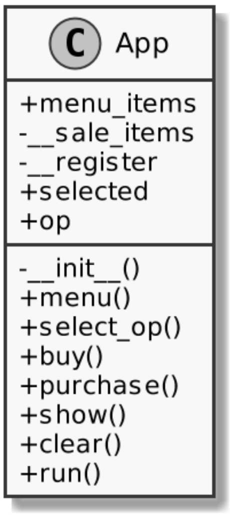


```
----------------------------------------------------------------------
| #         | Description         | Units in inventory | Price      |
----------------------------------------------------------------------
| Item    1 |               Jacket |                 12 |      59.95 |
| Item    2 |       Designer Jeans |                 40 |      34.95 |
| Item    3 |                Shirt |                 20 |      24.95 |
----------------------------------------------------------------------
```

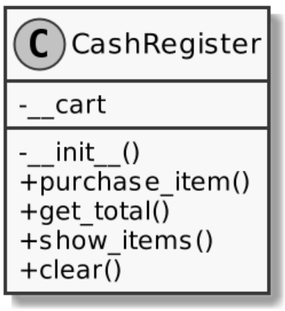
The output of program 5.

7. Cash Register

This programs builds on the RetailItem class and add a CashRegister class and a TUI to purchase items.



UML representation of the App class that has the overall control.



UML representation of the Cash register class that uses the RetailItem class to manage purchasing these.

prog7.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
"""
Name: Program 7 "Cash Register"
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (2017-02-19)

Program that simulates a cash register with built in inventory keeping.
"""
from ass12.retailitem import RetailItem
from ass12.cashregister import CashRegister


class App:
    """
    This class takes care of the overall control and ui of the application.
    """
    # The key assignments for the menu.
    QUIT = 0
    BUY = 1
    PURCHASE = 2
    SHOW = 3
    CLEAR = 4

    def __init__(self, items=[]):
        """
        Create the application class.

        ;param items: List of RetailItems for sale.
        """
        # List of menu items, their numbers, and methods.
        self.menu_items = [
            (self.QUIT, ' {}: {}'.format(self.QUIT, 'Quit'), None),
            (self.BUY, ' {}: {}'.format(self.BUY, 'Add items to the cart'),
             self.buy),
            (self.PURCHASE, ' {}: {}'.format(self.PURCHASE,
                                             'Purchase items in the cart'),
             self.purchase),
            (self.SHOW, ' {}: {}'.format(self.SHOW,
                                         'Show all items in the cart'),
             self.show),
            (self.CLEAR, ' {}: {}'.format(self.CLEAR, 'Clear the cart'),
             self.clear)
        ]

        # Items for sale
        self.__sale_items = items

        # Cash register
        self.__register = CashRegister()

        # Currently no entry is selected.
        self.selected = None
        self.op = -1
    def menu(self):
        """
        Print the main menu.
        """
        # Print header and selection name.
        print('\nMain menu', end='')
        if self.selected is None:
            print(' (no entry selected):')
        else:
            print(' ({} selected): '.format(self.selected))

        # Print menu entries.
        for entry in self.menu_items:
            print(entry[1])

        # Print the data of the selected entry.
        if self.selected is not None:
            print('\nSelected: ')
            self.printEntry(self.selected, False)

    def select_op(self):
        """
        Present a menu and validate the user selection.
        """
        # Show the menu.
        self.menu()
        # Get the user choice.
        choice = input('\nSelect operation: ')
        # Blank line.
        print('')

        # Validate the user input.
        try:
            self.op = int(choice)
            # Check if the choice is within range.
            if (self.op < 0) or (self.op > len(self.menu_items)):
                raise ValueError
        except ValueError:
            print('Wrong input, please try again')
            # Call recursively, program dies if you do enough wrong choices,
            # because of this.
            self.select_op()

    def buy(self):
        """
        Add user selected items to the cart..
        """
        # Variable for the selected item.
        selected = 1
        # Keep running the menu until 0 is selected
        while selected:
            print('Select an item to add it to the cart.\n')

            # Print a list of the items
            i = 0
            for i in range(len(self.__sale_items)):
                print('{:2.0f}: {}'.format(i + 1, self.__sale_items[i]))

            try:
                # Get the item selection from the user
                selected = int(input('\nInput item to add (0 to stop): '))
                # Handle, quit, wrong item, and adding.
                if selected == 0:
                    print('\n*Leaving item selection.*\n')
                elif len(self.__sale_items) < (selected):
                    print('\n*Non existing item selected.*\n')
                else:
                    self.__register.purchase_item(
                        self.__sale_items[selected - 1])

            except ValueError:
                print('\n*Please use only numbers.*\n')

    def purchase(self):
        """
        Edit an entry.
        """
        print('\nThank you for your purchase.\n')
        self.__register.clear()

    def show(self):
        """
        Show all entries in the cart.
        """
        self.__register.show_items()

    def clear(self):
        """
        Clear the cart.
        """
        self.__register.clear()

    def run(self):
        """
        Run the application.

        """
        # Let the user select an operation.
        self.select_op()

        # Run until quit is selected.
        while self.op != 0:
            # Run the selected operation.
            self.menu_items[self.op][2]()
            # Let the user select an operation.
            self.select_op()

        print('Bye.')


def main():
    """
    Program main entry point.
    """
    # These are the items for sale.
    items = list()
    items.append(RetailItem('BC547 npn transistor', 1, 1.05))
    items.append(RetailItem('BC557 pnp transistor', 1, 1.15))
    items.append(RetailItem('10kOhm 1/4W resistor', 1, 0.25))
    items.append(RetailItem('1kOhm 1/4W resistor', 1, 0.25))
    items.append(RetailItem('10uF 16V capacitor', 1, 0.50))
    items.append(RetailItem('5mm red LED', 1, 1.00))

    # Create an application instance using the items.
    app = App(items)
    # Run it.
    app.run()


# Run this when invoked directly
if __name__ == '__main__':
    main()
```

cashregister.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
"""
Name: Program 7 "Cash Register"
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (2017-02-19)

Class that simulates a cash register.
"""


class CashRegister:
    """
    CashRegister class.
    """

    def __init__(self):
        """
        Constructor.
        """
        # List holding the items selected for purchase.
        self.__cart = list()

    def purchase_item(self, new_item):
        """
        Purchase an item.

        :param item: A RetailItem.
        """
        # Add the item to the cart.
        # Run through all items and ask each to add it, if it is the owner.
        i = 1
        found = False
        if len(self.__cart):
            found = self.__cart[0].add_item(new_item.get_description())
            while not found and (len(self.__cart) > i):
                found = self.__cart[i].add_item(new_item.get_description())
                i += 1

        # No similar RetailItem was in the cart, add this one.
        if not found:
            self.__cart.append(new_item)

    def get_total(self):
        """
        Get the total price of the items in the cart.

        :return: Total price of items in the cart.
        """
        ret = 0
        for item in self.__cart:
            ret += item.get_price() * item.get_inventory()

        return (ret)
    def show_items(self):
        """
        Print all items in the cart.
        """
        # Print the items in a table, first the header.
        print('\n------------------------------------------------------' +
              '-----------')
        print('| #         | Description          | Units in cart |' +
              ' Price      |')
        print('------------------------------------------------------' +
              '-----------')
        # Print the items.
        for i in range(len(self.__cart)):
            item = self.__cart[i]
            print(
                '| Item {:4} | {:>20} |'.format(i + 1, item.get_description()),
                end='')
            print(' {:13.0f} | {:10.2f} |'.format(item.get_inventory(),
                                                  item.get_price() *
                                                  item.get_inventory()))

        # Handle empty table.
        if not len(self.__cart):
            print(
                '| No items                                        ' +
                '              |')
        # Close the table.
        print('-------------------------------------------------------' +
              '----------')
        print('Total price: {:10.2f}'.format(self.get_total()))

    def clear(self):
        """
        Clear the cart.
        """
        self.__cart = list()
```

retailitem.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
"""
Name: Program  5 "RetailItem Class"
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (2017-02-19)

Class that holds data about an item in a retail store.
"""


class RetailItem:
    """
    RetailItem class.
    """

    def __init__(self, description='None', inventory=0, price=0):
        """
        Initialise the retail item.

        :param description: The description of the item.
        :param inventory: The number of items in the inventory-
        :param price: The price of the item.
        """
        self.__description = description
        self.__inventory = inventory
        self.__price = price

    def get_description(self):
        """
        Get the item desciption.

        :return: String with item description.
        """
        return self.__description

    def get_inventory(self):
        """
        Get the number of items in the inventory.

        :return: The number of items.
        """
        return self.__inventory

    def get_price(self):
        """
        Get the price of the item.

        :return: The price of the item.
        """
        return self.__price

    def set_description(self, description):
        """
        Set the item description.
        :param description: The item description.
        """
        self.__description = description

    def set_inventory(self, inventory):
        """
        Set the number of items in the inventory.

        :param inventory: The number of items.
        """
        self.__inventory = inventory

    def set_price(self, price):
        """
        Set the price of the item.

        :param price: The proce of the item.
        """
        self.__price = price

    def add_item(self, description=None):
        """
        Helper function to add a new item to the inventory.
        If a description is supplied, the item description must match this
        for the item to get added.

        :param description: Item description.
        :return: True if added.
        """
        ret = False
        if description is None:
            self.__inventory += 1
            ret = True
        else:
            if description == self.get_description():
                self.__inventory += 1
                ret = True

        return ret

    def remove_item(self, description=None):
        """
        Helper function to remove an item.
        If a description is supplied, the item description must match this
        for the item to get removed.

        :param description: Item description.
        :return: True if removed.
        """
        ret = False
        if description is None:
            self.__inventory -= 1
            ret = True
        else:
            if description == self.get_description():
                self.__inventory -= 1
                ret = True

        return ret
    def __str__(self):
        """
        Return a string that is directly usable when printing the entry.

        :return: string
        """
        return ('{:20}\t'.format(self.get_description()) +
                '{:8.2f}\t'.format(self.get_price())
                )
```

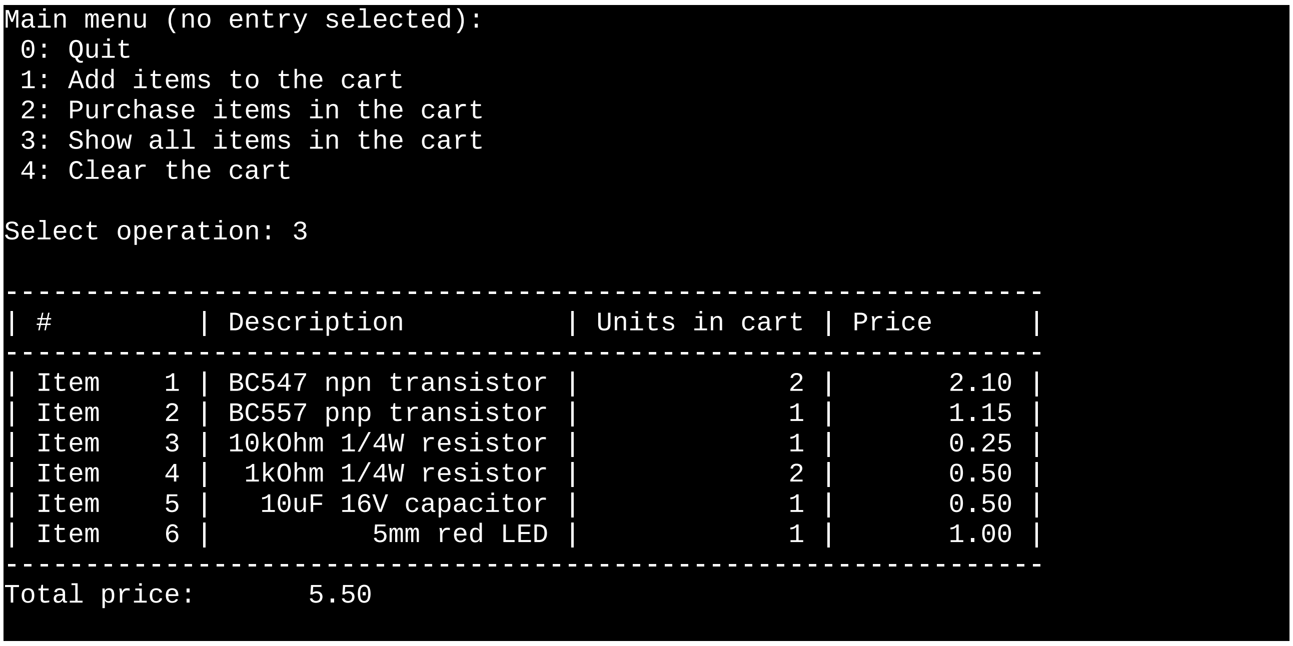
Result

```
Main menu (no entry selected):
 0: Quit
 1: Add items to the cart
 2: Purchase items in the cart
 3: Show all items in the cart
 4: Clear the cart

Select operation:
```

Main menu.

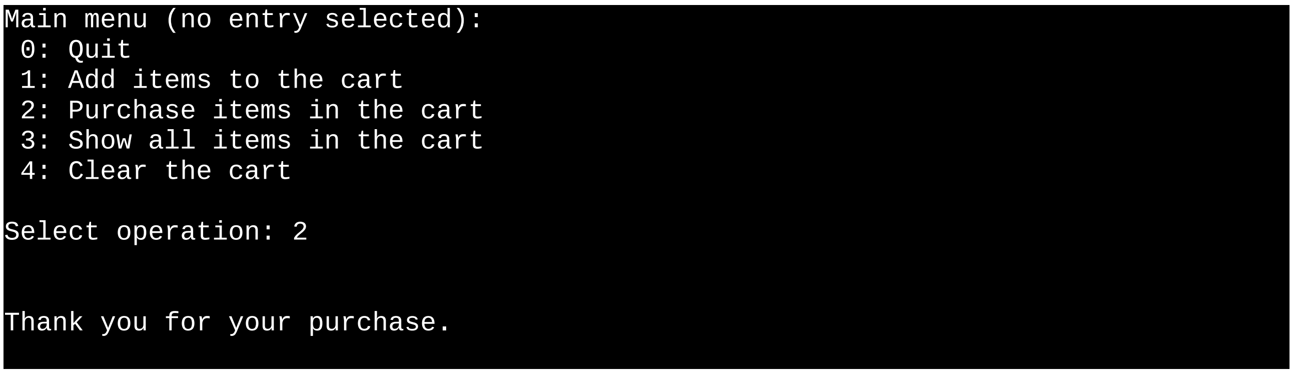
```
Select an item to add it to the cart.

 1: BC547 npn transistor          1.05
 2: BC557 pnp transistor          1.15
 3: 10kOhm 1/4W resistor          0.25
 4: 1kOhm 1/4W resistor           0.25
 5: 10uF 16V capacitor            0.50
 6: 5mm red LED                   1.00

Input item to add (0 to stop): 1

Select an item to add it to the cart.

 1: BC547 npn transistor          1.05
 2: BC557 pnp transistor          1.15
 3: 10kOhm 1/4W resistor          0.25
 4: 1kOhm 1/4W resistor           0.25
 5: 10uF 16V capacitor            0.50
 6: 5mm red LED                   1.00

Input item to add (0 to stop): 0

*Leaving item selection.*
```

Adding items to the cart.



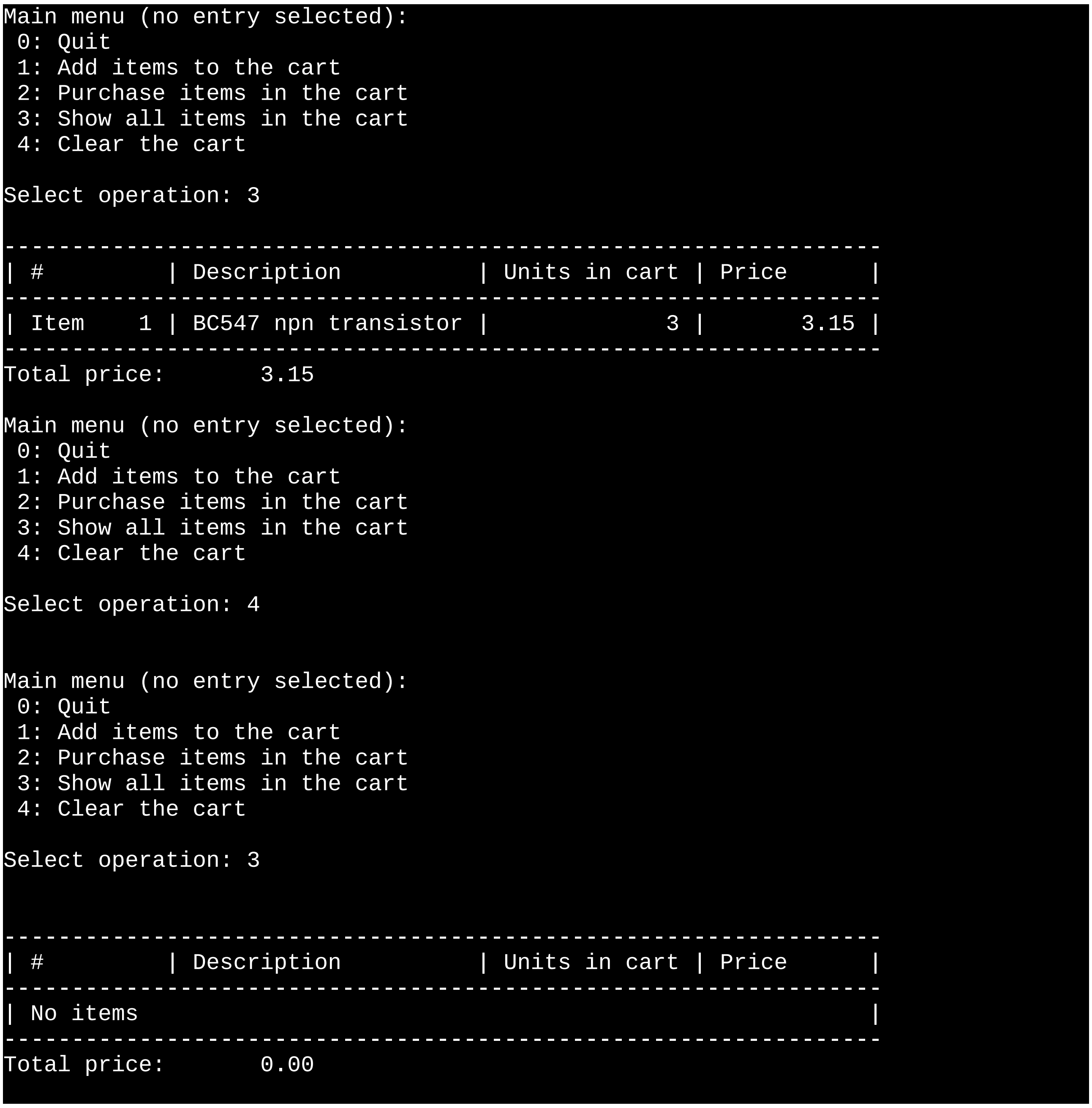
```
Main menu (no entry selected):
 0: Quit
 1: Add items to the cart
 2: Purchase items in the cart
 3: Show all items in the cart
 4: Clear the cart

Select operation: 3

-----------------------------------------------------------------
| #         | Description          | Units in cart | Price      |
-----------------------------------------------------------------
| Item    1 | BC547 npn transistor |             2 |       2.10 |
| Item    2 | BC557 pnp transistor |             1 |       1.15 |
| Item    3 | 10kOhm 1/4W resistor |             1 |       0.25 |
| Item    4 |  1kOhm 1/4W resistor |             2 |       0.50 |
| Item    5 |   10uF 16V capacitor |             1 |       0.50 |
| Item    6 |          5mm red LED |             1 |       1.00 |
-----------------------------------------------------------------
Total price:       5.50
```

Showing items in the cart.



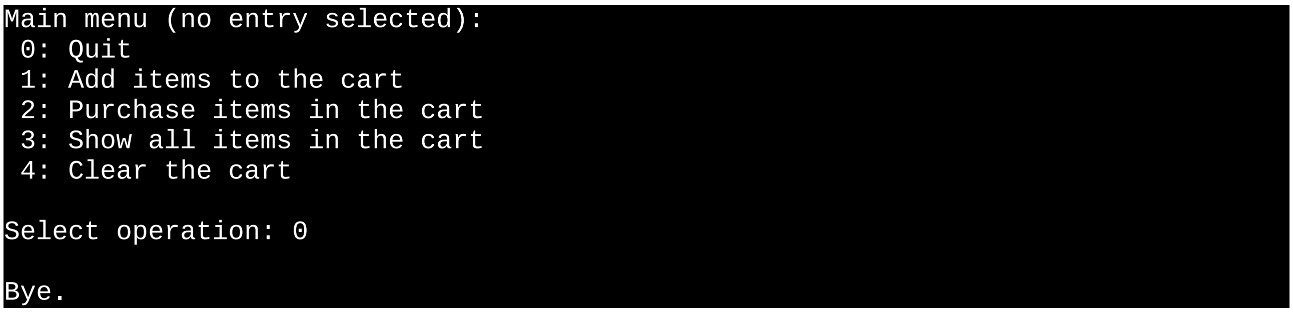
```
Main menu (no entry selected):
 0: Quit
 1: Add items to the cart
 2: Purchase items in the cart
 3: Show all items in the cart
 4: Clear the cart

Select operation: 2


Thank you for your purchase.
```

Buying stuff.



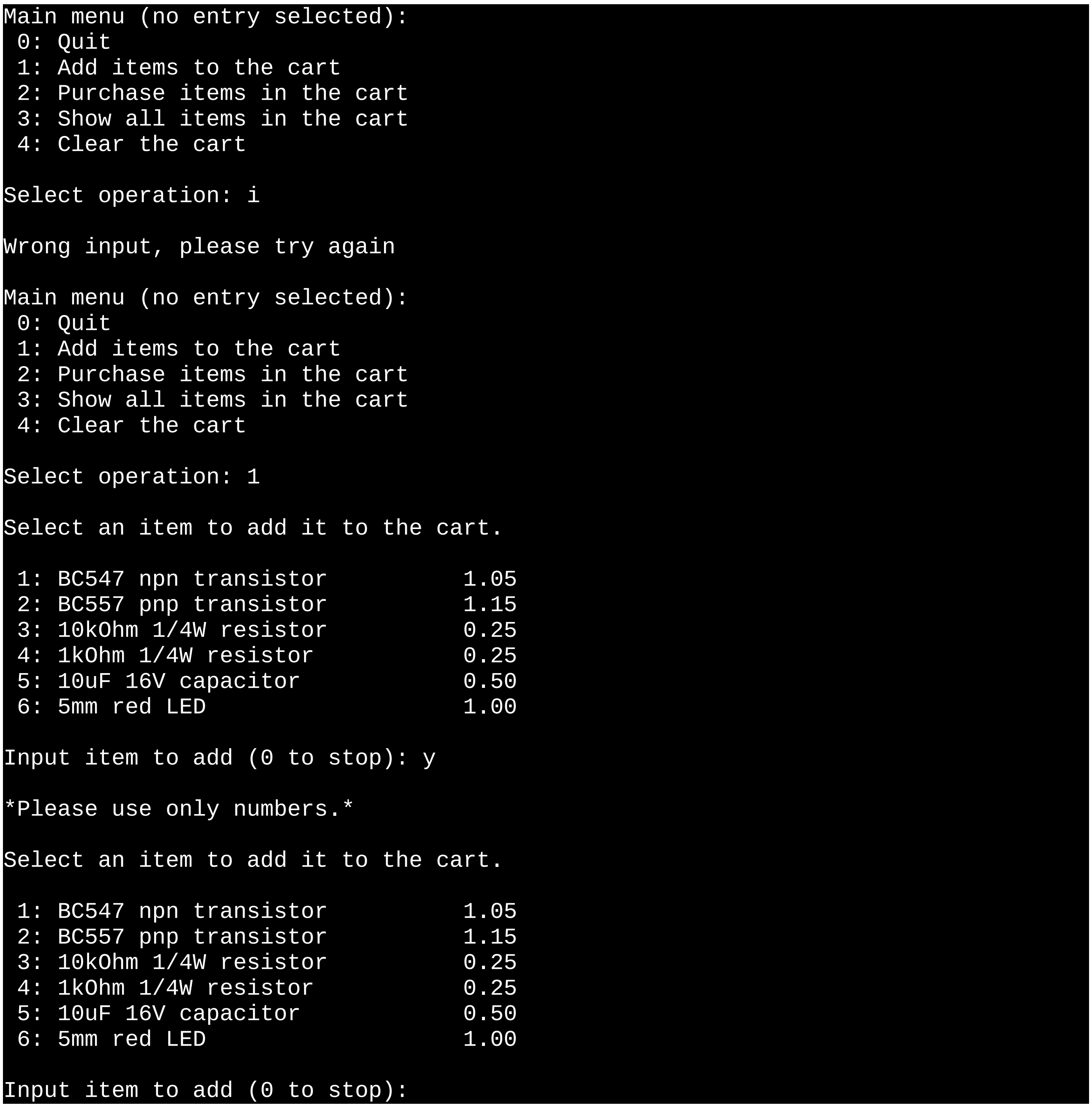
```
Main menu (no entry selected):
 0: Quit
 1: Add items to the cart
 2: Purchase items in the cart
 3: Show all items in the cart
 4: Clear the cart

Select operation: 3

-----------------------------------------------------------------
| #         | Description          | Units in cart | Price      |
-----------------------------------------------------------------
| Item    1 | BC547 npn transistor |             3 |       3.15 |
-----------------------------------------------------------------
Total price:       3.15

Main menu (no entry selected):
 0: Quit
 1: Add items to the cart
 2: Purchase items in the cart
 3: Show all items in the cart
 4: Clear the cart

Select operation: 4


Main menu (no entry selected):
 0: Quit
 1: Add items to the cart
 2: Purchase items in the cart
 3: Show all items in the cart
 4: Clear the cart

Select operation: 3


-----------------------------------------------------------------
| #         | Description          | Units in cart | Price      |
-----------------------------------------------------------------
| No items                                                      |
-----------------------------------------------------------------
Total price:       0.00
```

Clearing the cart.



```
Main menu (no entry selected):
 0: Quit
 1: Add items to the cart
 2: Purchase items in the cart
 3: Show all items in the cart
 4: Clear the cart

Select operation: 0

Bye.
```

Quitting.

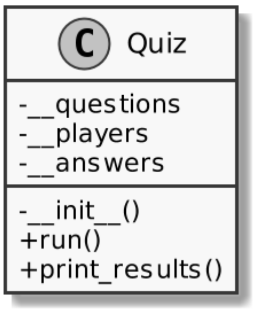


```
Main menu (no entry selected):
 0: Quit
 1: Add items to the cart
 2: Purchase items in the cart
 3: Show all items in the cart
 4: Clear the cart

Select operation: i

Wrong input, please try again

Main menu (no entry selected):
 0: Quit
 1: Add items to the cart
 2: Purchase items in the cart
 3: Show all items in the cart
 4: Clear the cart

Select operation: 1

Select an item to add it to the cart.

 1: BC547 npn transistor          1.05
 2: BC557 pnp transistor          1.15
 3: 10kOhm 1/4W resistor          0.25
 4: 1kOhm 1/4W resistor           0.25
 5: 10uF 16V capacitor            0.50
 6: 5mm red LED                   1.00

Input item to add (0 to stop): y

*Please use only numbers.*

Select an item to add it to the cart.

 1: BC547 npn transistor          1.05
 2: BC557 pnp transistor          1.15
 3: 10kOhm 1/4W resistor          0.25
 4: 1kOhm 1/4W resistor           0.25
 5: 10uF 16V capacitor            0.50
 6: 5mm red LED                   1.00

Input item to add (0 to stop):
```

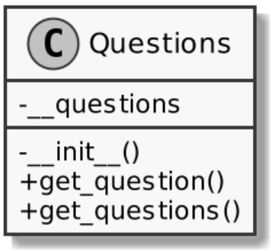
Handling of wrong input.

8. Trivia Game

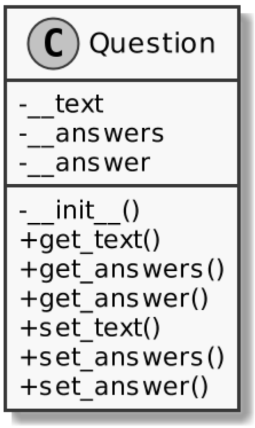
This program uses a Question class, holding questions and answers, class as the basis for a Trivia Game. A Questions class keeps track of all the questions and loads them form a text file. The Quiz class manages the players, ask the questions, gets the answers, and outputs the final game results.



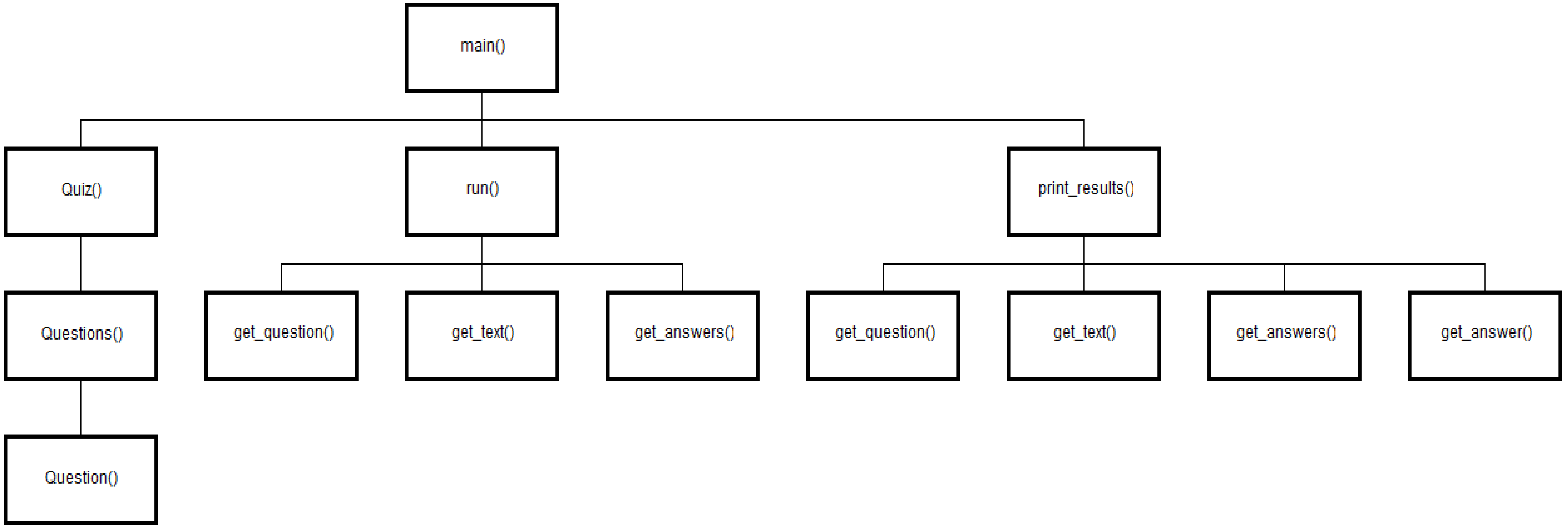
UML representation of the Quiz class that runs the quiz



UML representation of the Questions that loads the questions from a text file.



UML representation of the Question class that holds a question and answers.



Partial hierarchy diagram for program 8.

The hierarchy diagram above is not complete, it shows only calls to the internal functions that form the main logic.
To the left we see the string of constructors starting from the Quiz object being instantiated in the main function. The run function is where the questions are asked, so it fetches one using get_question. Run then calls get_text and get_answers to get the question and the possible answers for displaying. When the run function is done asking questions, the print_result method takes over. It runs through all questions and displays them getting the text using get_text. It uses get answers to get the text of the correct answer and the guessed answer. Lastly print_result uses get_answer to get the correct answer for checking.

prog8.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
"""
Name: Program 8 "Trivia Game"
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (2017-02-12)

A Trivia game.
"""
from ass12.quiz import Quiz


def main():
    """
    Program main entry point.
    """
    # Create a game for 2 players.
    prog_quiz = Quiz(2, 'triviaQuestionsChap11.txt')
    # Run the game.
    prog_quiz.run()
    # Print the results.
    prog_quiz.print_results()


# Run this when invoked directly
if __name__ == '__main__':
    main()
```

quiz.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
"""
Name: Program 8 "Trivia Game"
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (2017-02-19)

Class that handles the general quiz logic.
"""
import random

from ass12.questions import Questions

class Quiz:
    """
    Quiz class.
    """
    def __init__(self, players, data_file_path):
        """
        Constructor.

        :param players: Number of players.
        :param data_file_path: Text file to read the questions and answers from.
        """
        self.__questions = Questions(data_file_path)
        self.__players = players
        self.__answers = list()

    def run(self):
        """
        Run a the quiz.
        """
        n_questions = len(self.__questions.get_questions())

        for player in range(self.__players):
            print('\nQuestions for player {}'.format(player + 1))
            print('----------------------\n\n')

            n_answered = list()
            i = 0
            for i in range(5):
                print('Question {}:'.format(i + 1))
                print('-----------\n')
                number = random.randrange(0, n_questions)
                # If question is answered find a new one.
                # If question choices are exhausted this will go into an
                # endless loop.
                while number in [a[0] for a in n_answered]:
                    number = random.randrange(0, n_questions)

                # Print question.
                current_question = self.__questions.get_question(number)
                print(current_question.get_text())
                # Print answers.
                for j in range(4):
                    print(' {}: {}'.format(j + 1,
                                           current_question.get_answers(j + 1)))

                # Get the user answer.
                u_answer = 0
                while not u_answer:
                    try:
                        u_answer = int(
                            input('\nEnter the number of the correct answer: '))
                        if (u_answer < 1) or (u_answer > 4):
                            raise ValueError
                    except ValueError:
                        print('\n*Please use only numbers 1, 2, 3, and 4.*\n')
                        u_answer = 0

                # Add to answered questions.
                n_answered.append((number, u_answer))

                print()

            # Save the players answers.
            self.__answers.append(n_answered)

    def print_results(self):
        """
        Print the results of a game.
        """
        # For all players
        for player in range(self.__players):
            print('\nResults for player {}'.format(player + 1))
            print('----------------------\n\n')

            # All answers.
            i = 1
            correct = 0
            for answer in self.__answers[player]:
                print('Question {}:'.format(i))
                print('------------\n')
                # ¿Que?
                que = self.__questions.get_question(answer[0])
                # Print question, answer, and correct answer.
                print(que.get_text())
                print('Answer: ' +
                      format(que.get_answers(answer[1])))
                print('Correct answer: ' +
                      format(que.get_answers(que.get_answer())))
                if que.get_answer() == answer[1]:
                    print('\nAnswer is correct.')
                    correct += 1
                else:
                    print('\nAnswer is wrong.')

                # Next question.
                i += 1

                print()

        # Print final scores for all players.
        for player in range(self.__players):
            print('Player {}: {} answer(s) of 5 correct.'.format(player + 1,
                                                                 correct))
```

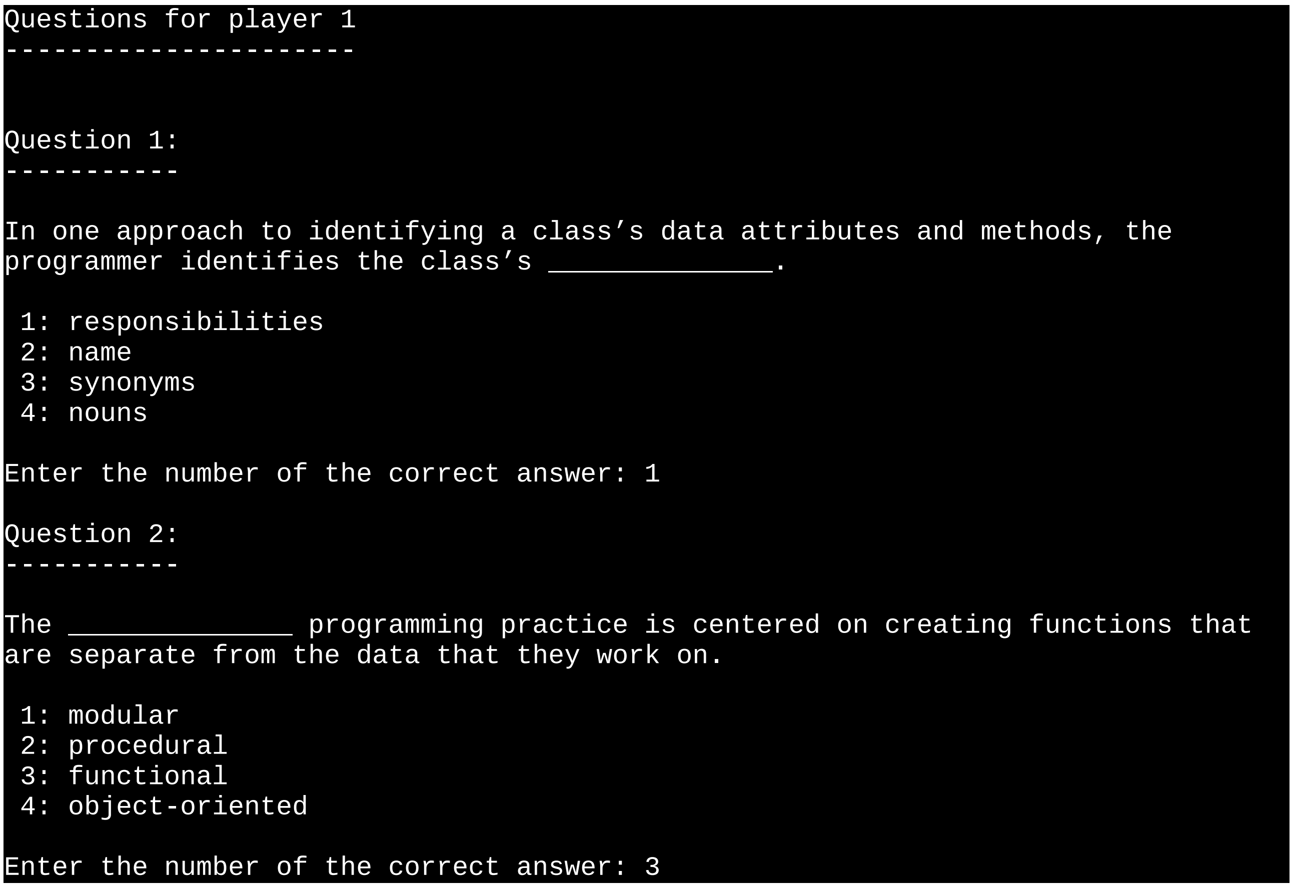
questions.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
"""
Name: Program 8 "Trivia Game"
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (2017-02-19)

Class that encapsulates trivia questions.
"""
from ass12.question import Question


class Questions:
    """
    Class that holds trivia questions.
    """

    def __init__(self, file_name=None):
        """
        Constructor.

        :param file_name: File name to read the questions from.
        """
        self.__questions = list()

        # Read the questions from the input file.
        # Handle IO errors and conversion errors using excaptions.
        try:
            with open(file_name, 'r') as data_file:
                # Read every line until the end of the file.
                # This actually ends up reading a chunk of 6 lines during each
                # iteration.
                for line in data_file:
                    # First line is the question.
                    q = Question()
                    q.set_text(line)

                    # Next 4 line are the answers.
                    q_answers = list()
                    line = data_file.readline().strip()
                    if line != '':
                        q_answers.append(line.strip())
                    line = data_file.readline().strip()
                    if line != '':
                        q_answers.append(line.strip())
                    line = data_file.readline().strip()
                    if line != '':
                        q_answers.append(line.strip())
                    line = data_file.readline().strip()
                    if line != '':
                        q_answers.append(line.strip())
                        q.set_answers(q_answers)

                    # Last line is the answer.
                    line = data_file.readline().strip()
                if line != '':
                    q.set_answer(int(line.strip()))
                    # Add the question, now that we've read it without
                    # failure.
                    self.__questions.append(q)

        except IOError as ex:
            # Complain when something goes wrong with the file access.
            print('Exception: {}'.format(str(ex)))
            print('Error loading quiz questions.')
            exit(1)
        except ValueError:
            # Something wrong with the contents if the file.
            print('Wrong format of input file.')
            exit(1)

    def get_question(self, number):
        """
        Get a specific question.

        :param number: The question number.
        :return: Question object.
        """
        return self.__questions[number]

    def get_questions(self):
        """
        Get a list of all questions.

        :return: List of Question ojects.
        """
        return self.__questions
```

question.py

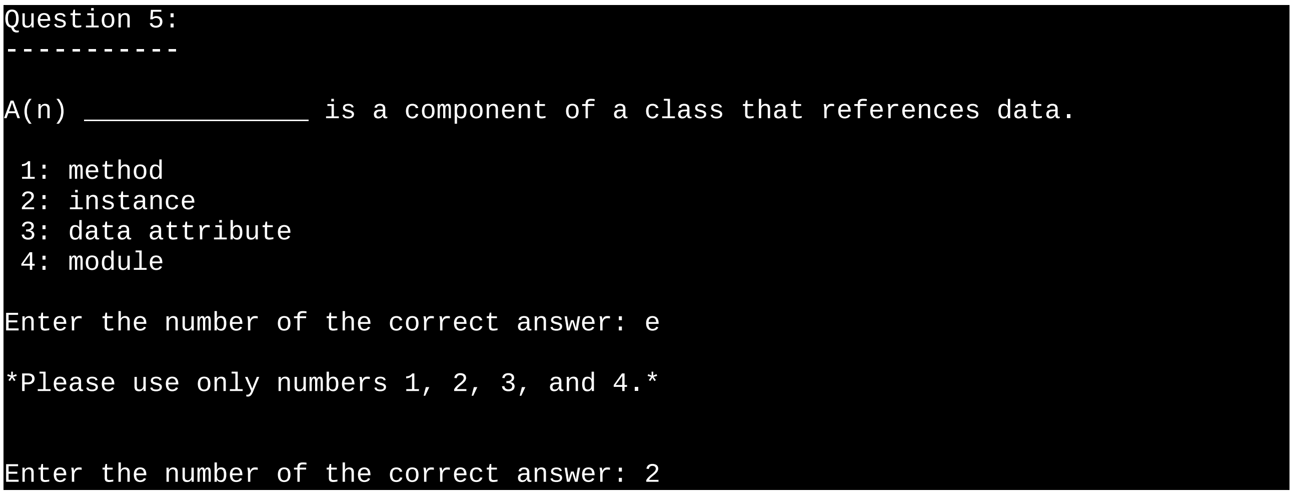
```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
"""
Name: Program 8 "Trivia Game"
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (2017-02-19)

Class that encapsulates q trivia question.
"""


class Question:
    """
    Class that holds personal information.
    """

    def __init__(self, text='None', answers=[], answer=0):
        """
        Constructor.

        :param text: The question.
        :param answers: A list of answer strings.
        :param answer: Number of the correct answer.
        """
        self.__text = text
        self.__answers = answers
        self.__answer = 1

    def get_text(self):
        """
        Get the question.

        :return: The question.
        """
        return self.__text

    def get_answers(self, number = None):
        """
        Get the answers.

        :param number: If set only, return the specific question.
        :return: List of answers.
        """
        ret = self.__answers

        if (number > 0) and (number < 5):
            ret = self.__answers[number - 1]

        return ret

    def get_answer(self):
        """
        Get the number of the correct answer.

        :return: The number of the correct answer.
        """
        return self.__answer

    def set_text(self, text):
        """
        Set the question.

        :param text: String with the question.
        """
        self.__text = text

    def set_answers(self, answers):
        """
        Set the possible answers.

        :param answers: List of answers.
        """
        if len(answers) == 4:
            self.__answers = answers
        else:
            print('Error: Exactly 4 answers are required')

    def set_answer(self, answer):
        """
        Set the number of the correct answer.

        :param answer: The number of the correct answer.
        """
        if (answer > 0) and (answer < 5):
            self.__answer = answer
        else:
            print('Answer is out of range.')
```

Result



```
Questions for player 1
----------------------


Question 1:
-----------

In one approach to identifying a class’s data attributes and methods, the
programmer identifies the class’s ______________.

 1: responsibilities
 2: name
 3: synonyms
 4: nouns

Enter the number of the correct answer: 1

Question 2:
-----------

The ______________ programming practice is centered on creating functions that
are separate from the data that they work on.

 1: modular
 2: procedural
 3: functional
 4: object-oriented

Enter the number of the correct answer: 3
```

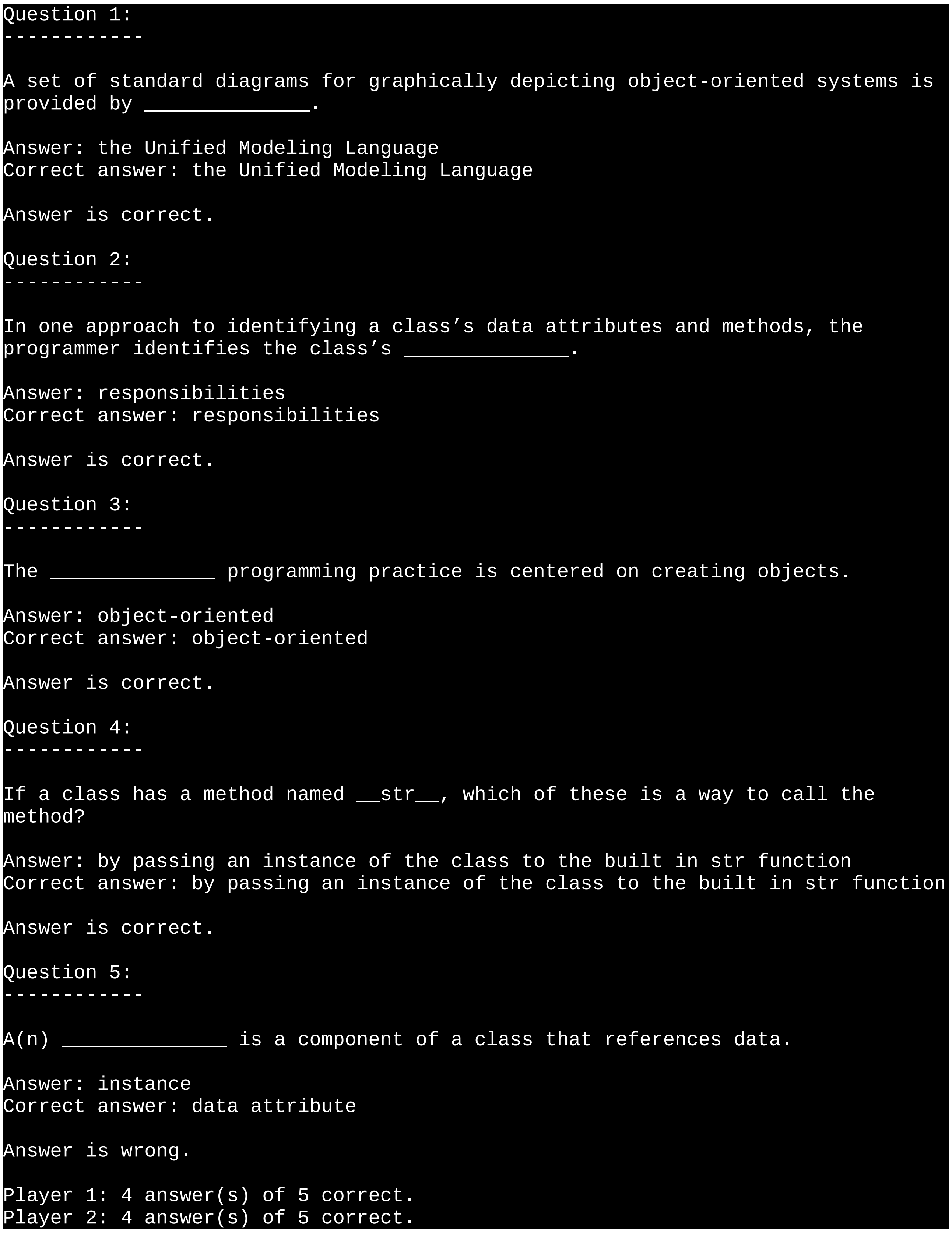
Player 1 answering the first questions.



```
Question 5:
-----------

A(n) ______________ is a component of a class that references data.

 1: method
 2: instance
 3: data attribute
 4: module

Enter the number of the correct answer: e

*Please use only numbers 1, 2, 3, and 4.*


Enter the number of the correct answer: 2
```

Handling of incorrect input.

```
Results for player 1
----------------------

Question 1:
------------

In one approach to identifying a class’s data attributes and methods, the
programmer identifies the class’s ______________.

Answer: responsibilities
Correct answer: responsibilities

Answer is correct.

Question 2:
------------

The ______________ programming practice is centered on creating functions that
are separate from the data that they work on.

Answer: functional
Correct answer: procedural

Answer is wrong.

Question 3:
------------

The ______________ programming practice is centered on creating objects.

Answer: object-oriented
Correct answer: object-oriented

Answer is correct.

Question 4:
------------

A(n) ______________ method stores a value in a data attribute or changes its
value in some other way.

Answer: mutator
Correct answer: mutator

Answer is correct.

Question 5:
------------

In one approach to identifying the classes in a problem, the programmer
identifies the ______________ in a description of the problem domain.

Answer: verbs
Correct answer: nouns

Answer is wrong.


Results for player 2
----------------------
```



```
Question 1:
------------

A set of standard diagrams for graphically depicting object-oriented systems is
provided by ______________.

Answer: the Unified Modeling Language
Correct answer: the Unified Modeling Language

Answer is correct.

Question 2:
------------

In one approach to identifying a class’s data attributes and methods, the
programmer identifies the class’s ______________.

Answer: responsibilities
Correct answer: responsibilities

Answer is correct.

Question 3:
------------

The ______________ programming practice is centered on creating objects.

Answer: object-oriented
Correct answer: object-oriented

Answer is correct.

Question 4:
------------

If a class has a method named __str__, which of these is a way to call the
method?

Answer: by passing an instance of the class to the built in str function
Correct answer: by passing an instance of the class to the built in str function

Answer is correct.

Question 5:
------------

A(n) ______________ is a component of a class that references data.

Answer: instance
Correct answer: data attribute

Answer is wrong.

Player 1: 4 answer(s) of 5 correct.
Player 2: 4 answer(s) of 5 correct.
```

Output of the final results when the game is over.

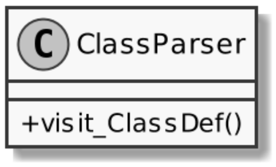
Python to PlantUML

I am a lazy man, Python helps me a lot in this regard. An example of how Python helps me being lazy is generating the class diagrams in this report, I use the Python AST to parse the source files for classes, methods and members. The program outputs a file in PlantUML[1] which is an open source program and a language to create graphs. PlantUML renders the output from my program into the graphics in this document, using GraphViz/dot.

Running py2uml on itself produces the following output:

```
@startuml
skinparam monochrome true
skinparam classAttributeIconSize 0
scale 2
class ClassParser{
    +visit_ClassDef()
}
@enduml
```

Example PlantUML source



Example rendering of the PlantUML file

1 http://plantuml.com/

py2puml.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
#-------------------------------------------------------------------------------
# "THE BEER-WARE LICENSE" (Revision 42):
# <martin.groenholdt@gmail.com> wrote this file. As long as you retain this
# notice you can do whatever you want with this stuff. If we meet some day,
# and you think this stuff is worth it, you can buy me a beer in return.
# Martin B. K. Grønholdt
#-------------------------------------------------------------------------------
# Program to parse Python classes and write their info to PlantUML files
# (see http://plantuml.com/) that can be used to generate UML files and GraphViz
# renderings of classes.
#
# Missing:
#   * Inheritance parsing
#   * Does not like end='' in print.
#
# History:
#
# Version 0.1.2
#   * Exception handling.
#
# Version 0.1.1
#   * Comments.
#
# Version 0.1.0
#   * First working version.

import argparse
import ast

__version__ = '0.1.2'

class ClassParser(ast.NodeVisitor):
    """
    Class to parse the stuff we're interested in from a class.

     * Methods and their visibility.
     * Members created in __init__ and their visibility.
    """
    # List to put the class data.
    puml_classes = list()

    def visit_ClassDef(self, node):
        """
        Class visitor that parses the info we want, when encountering a class
        definition.

        :param node: The node of the class.
        """
        #Dictionary containing the interesting parts of the classes structure
        puml_class = dict()
        puml_class['name'] = node.name
        puml_class['members'] = list()
        puml_class['methods'] = list()

        #Run through all children of the class definition.
        for child in node.body:
            # If we have a function definition, store it.
            if isinstance(child, ast.FunctionDef):
                # Check if it is "private".
                if child.name.startswith('__'):
                    puml_class['methods'].append('-' + child.name)
                else:
                    puml_class['methods'].append('+' + child.name)

                # Check if this s the constructor.
                if child.name == '__init__':
                    # Find all assignment expressions in the constructor.
                    for code in child.body:
                        if isinstance(code, ast.Assign):
                            # Find attributes since we want "self." +
                            # "something"
                            for target in code.targets:
                                if isinstance(target, ast.Attribute):
                                    # If the value is a name and its id is self.
                                    if isinstance(target.value, ast.Name):
                                        if target.value.id == 'self':
                                            # Check if it is "private".
                                            members = puml_class['members']
                                            if target.attr.startswith('__'):
                                                members.append('-' +
                                                               target.attr)
                                            else:
                                                members.append('+' +
                                                               target.attr)

        # Save the class.
        self.puml_classes.append(puml_class)


if __name__ == '__main__':
    # Takes a python file as a parameter.
    parser = argparse.ArgumentParser(prog='py2puml')
    parser.add_argument('py_file', type=argparse.FileType('r'))
    args = parser.parse_args()

    # Output file name is input file +'.puml'
    puml_file_name = args.py_file.name + '.puml'

    try:
        with open(puml_file_name, 'w') as puml_file:
            # Write the beginnings of the PlantUML file.
            puml_file.write('@startuml\nskinparam monochrome true\n' +
                            'skinparam classAttributeIconSize 0\nscale 2\n')

            # Use AST to parse the file.
            tree = ast.parse(args.py_file.read())
            class_writer = ClassParser()
            class_writer.visit(tree)

            # Write the resulting classes in PlantUML format.
            for puml_class in class_writer.puml_classes:
                puml_file.write('class ' + puml_class['name'] + '{\n')

                for member in puml_class['members']:
                    puml_file.write('    ' + member + '\n')

                for method in puml_class['methods']:
                    puml_file.write('    ' + method + '()\n')

                puml_file.write('}\n')

            # End the PlantUML files.
            puml_file.write('@enduml')
    except IOError:
        print('I/O error.')
    except SyntaxError as see:
        print('Syntax error in ', end='')
        print( args.py_file.name + ':' + str(see.lineno) + ':' + str(see.offset)
               + ': ' + see.text)
```

The source is a bit ugly the way `visit_ClassDef` parses the class, but it works ^{TM}

Conclusion

The programs in this assignment were more complete, which makes them more complex. A lot of this is because the user might do stuff that you do not expect. Therefore the user should have clear concise output from the program, and input from the user should be treated like it was entered by a three year old supah-haXOR with a hammer. Thus more code goes into validation and output.