**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по РК №2

Вариант запросов: B

Вариант предметной области: 2

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-33Б |  | преподаватель каф. ИУ5 |
| Абрамов Александр |  | Гапанюк Ю. Е. |
|  |  |  |

Москва, 2023 г.

**Вариант запросов B.**

# «Школьник» и «Класс» связаны соотношением один-ко-многим. Выведите список всех школьников, фамилия которых начинается на букву «А».

# «Школьник» и «Класс» связаны соотношением один-ко-многим. Выведите список классов с минимальным средним баллом в каждом классе, отсортированный по минимальному среднему баллу.

# «Школьник» и «Класс» связаны соотношением многие-ко-многим. Выведите список всех связанных школьников и классов, отсортированный по школьникам, сортировка классов произвольная.

# **Задание**

1) Проведите рефакторинг текста программы рубежного контроля №1 таким образом, чтобы он был пригоден для модульного тестирования.

2) Для текста программы рубежного контроля №1 создайте модульные тесты с применением TDD - фреймворка (3 теста).

**Текст программы**

**Файл main.py**

class Student:  
 def \_\_init\_\_(self, student\_id: int, name: str, class\_id: int, avg\_rating: float):  
 self.\_student\_id = student\_id  
 self.\_name = name  
 self.\_class\_id = class\_id  
 self.\_avg\_rating = avg\_rating  
  
 @property  
 def student\_id(self) -> int:  
 return self.\_student\_id  
  
 @property  
 def name(self) -> str:  
 return self.\_name  
  
 @property  
 def class\_id(self) -> int:  
 return self.\_class\_id  
  
 @property  
 def avg\_rating(self) -> float:  
 return self.\_avg\_rating  
  
  
class Class:  
 def \_\_init\_\_(self, class\_id: int, name: str):  
 self.\_class\_id = class\_id  
 self.\_name = name  
  
 @property  
 def class\_id(self) -> int:  
 return self.\_class\_id  
  
 @property  
 def name(self) -> str:  
 return self.\_name  
  
  
def query1(student\_class: list):  
 # Задание В1  
 data = []  
 for student, cls in student\_class:  
 if student.name.startswith('А'):  
 data.append((student.name, cls.name))  
 return data  
  
  
def query2(student\_class: list):  
 # Задание В2  
 data = []  
 class\_min\_ratings = {}  
 for student, cls in student\_class:  
 if cls.name in class\_min\_ratings:  
 if student.avg\_rating < class\_min\_ratings[cls.name]:  
 class\_min\_ratings[cls.name] = student.avg\_rating  
 else:  
 class\_min\_ratings[cls.name] = student.avg\_rating  
  
 sorted\_classes = sorted(class\_min\_ratings.items(), key=lambda x: x[1])  
 for group, min\_rating in sorted\_classes:  
 data.append((group, min\_rating))  
 return data  
  
  
def query3(student\_class: list):  
 # Задание В3  
 data = []  
 student\_class.sort(key=lambda x: x[0].name)  
 for student, cls in student\_class:  
 data.append((student.name, cls.name))  
 return data  
  
  
def generate\_data():  
 # Создаем объекты класса Class  
 classes = [  
 Class(1, "2Б"),  
 Class(2, "10А"),  
 Class(3, "5В")  
 ]  
  
 # Создаем объекты класса Student  
 students = [  
 Student(1, "Лупарев", 1, 4.9),  
 Student(2, "Гукасян", 1, 3.8),  
 Student(3, "Абрамов", 2, 4.5),  
 Student(4, "Иноземцев", 2, 2.7),  
 Student(5, "Барсукова", 3, 5.0)  
 ]  
  
 # Создаем список "Школьники и классы" для связи один-ко-многим  
 student\_class = [  
 (students[0], classes[0]),  
 (students[1], classes[0]),  
 (students[2], classes[1]),  
 (students[3], classes[1]),  
 (students[4], classes[2])  
 ]  
 return classes, students, student\_class  
  
  
def execute\_queries(student\_class: list):  
 print("Задание В1")  
 for stud, cls in query1(student\_class):  
 print(f"{stud} - {cls}")  
 print()  
  
 print("Задание В2")  
 for group, rating in query2(student\_class):  
 print(f"{group} - Минимальный рейтинг: {rating}")  
 print()  
  
 print("Задание В3")  
 for stud, cls in query3(student\_class):  
 print(f"{stud} - {cls}")  
 print()  
  
  
def main():  
 # Генерация данных  
 classes, students, student\_class = generate\_data()  
  
 # Запуск запросов  
 execute\_queries(student\_class)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

**Файл TDDtests.py**

import unittest  
from main import \*  
  
  
# Тестирование класса "Студент"  
class TestStudent(unittest.TestCase):  
  
 def test\_student\_creation(self):  
 student = Student(1, "Беляев", 2, 4.5)  
 self.assertEqual(student.student\_id, 1)  
 self.assertEqual(student.name, "Беляев")  
 self.assertEqual(student.class\_id, 2)  
 self.assertEqual(student.avg\_rating, 4.5)  
  
  
# Тестирование класса "Класс"  
class TestClass(unittest.TestCase):  
  
 def test\_computer\_classroom\_creation(self):  
 cls = Class(1, "1Б")  
 self.assertEqual(cls.class\_id, 1)  
 self.assertEqual(cls.name, "1Б")  
  
  
# Тестирование выполнения запросов  
class TestQueryExecution(unittest.TestCase):  
 def setUp(self):  
 self.classes, self.students, self.student\_class = generate\_data()  
  
 # Тестирование запроса №1  
 def test\_query1(self):  
 result = query1(self.student\_class)  
 self.assertEqual(result, [("Абрамов", "10А")])  
  
 # Тестирование запроса №2  
 def test\_query2(self):  
 result = query2(self.student\_class)  
 self.assertEqual(result, [("10А", 2.7), ("2Б", 3.8), ("5В", 5.0)])  
  
 # Тестирование запроса №3  
 def test\_query3(self):  
 result = query3(self.student\_class)  
 self.assertEqual(result,  
 [("Абрамов", "10А"), ("Барсукова", "5В"), ("Гукасян", "2Б"),  
 ("Иноземцев", "10А"), ("Лупарев", "2Б")])  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

**Результат выполнения программы**

