Московский государственный технический университет им. Н.Э. Баумана

Факультет ИУ
Кафедра ИУ5

Курс «Основы информатики»
Отчет по Рубежному контролю №2

Выполнил студент группы ИУ5-33Б: Уфимцев Е.Е. Подпись и дата:

Проверил преподаватель каф.: Гапанюк Ю. Е. Подпись и дата:

Москва, 2024 г

Code Realisation

```
import unittest
from typing import List, Tuple


class Conductor:
    def __init__(self, conductorID: int, name: str,
work_experience: int, orchestraID: int):
        self._conductorID = conductorID
        self._name = name
        self._work_experience = work_experience
        self._orchestraID = orchestraID

    def __str__(self) -> str:
        return f"Conductor: {self._name}"

    @property
    def id(self) -> int:
        return self._conductorID

    @property
    def name(self) -> str:
        return self._name

    @property
    def work_experience(self) -> int:
        return self._work_experience

    @property
    def orchestraID(self) -> int:
        return self._orchestraID


class Orchestra:
    def __init__(self, orchestraID: int, name: str,
date: int):
        self._ID = orchestraID
        self._name = name
        self._establishment = date
```

```
    def __str__(self) -> str:
        return f"Orchestra: {self._name}"

    @property
    def id(self) -> int:
        return self._ID

    @property
    def name(self) -> str:
        return self._name

    @property
    def establishment(self) -> int:
        return self._establishment


class Ensemble:
    def __init__(self, conductorID: int, orchestraID:
int):
        self._conductor_ID = conductorID
        self._orchestra_ID = orchestraID

    @property
    def conductorID(self) -> int:
        return self._conductor_ID

    @property
    def orchestraID(self) -> int:
        return self._orchestra_ID


def get_one_to_many(conductors: List[Conductor],
orchestras: List[Orchestra]) -> List[Tuple[str, int,
str, int]]:
    return [
        (conductor.name, conductor.work_experience,
orchestra.name, orchestra.establishment)
        for conductor in conductors
        for orchestra in orchestras
        if conductor.orchestraID == orchestra.id
```

```
    ]


def get_many_to_many(conductors: List[Conductor],
orchestras: List[Orchestra], ensembles: List[Ensemble])
-> List[Tuple[str, int, str]]:
    many_to_many_temp = [
        (orchestra.name, ensemble.conductorID,
ensemble.orchestraID)
        for orchestra in orchestras
        for ensemble in ensembles
        if orchestra.id == ensemble.orchestraID
    ]
    return [
        (conductor.name, conductor.work_experience,
orchestra_name)
        for orchestra_name, conductorID, _ in
many_to_many_temp
        for conductor in conductors if conductor.id ==
conductorID
    ]


def task1(one_to_many: List[Tuple[str, int, str, int]])
-> List[Tuple[str, str]]:
    result = [(conductor_name, orchestra_name) for
conductor_name, _, orchestra_name, _ in one_to_many]
    return sorted(result, key=lambda x: x[0])


def task2(one_to_many: List[Tuple[str, int, str, int]])
-> List[Tuple[str, str, int]]:
    result = [
        (orchestra_name, conductor_name,
work_experience)
        for conductor_name, work_experience,
orchestra_name, _ in one_to_many
        if work_experience <= 10
    ]
    return sorted(result, key=lambda x: x[2])
```

```
def task3(many_to_many: List[Tuple[str, int, str]]) ->
List[Tuple[str, str]]:
    result = [(orchestra_name, conductor_name) for
conductor_name, _, orchestra_name in many_to_many]
    return sorted(set(result), key=lambda x: x[0])


# Unit Tests
class TestOrchestraTasks(unittest.TestCase):
    def setUp(self):
        self.orchestras = [
            Orchestra(1, "Gewandhausorchester Leipzig",
1781),
            Orchestra(2, "New York Philharmonic",
1842),
        ]
        self.conductors = [
            Conductor(1, "Mario", 11, 1),
            Conductor(2, "Zuck", 7, 2),
        ]
        self.ensembles = [
            Ensemble(1, 1),
            Ensemble(2, 2),
        ]
        self.one_to_many =
get_one_to_many(self.conductors, self.orchestras)
        self.many_to_many =
get_many_to_many(self.conductors, self.orchestras,
self.ensembles)

    def test_task1(self):
        expected = [("Mario", "Gewandhausorchester
Leipzig"), ("Zuck", "New York Philharmonic")]
        self.assertEqual(task1(self.one_to_many),
expected)

    def test_task2(self):
```

```
        expected = [("New York Philharmonic", "Zuck",
7)]
        self.assertEqual(task2(self.one_to_many),
expected)

    def test_task3(self):
        expected = [("Gewandhausorchester Leipzig",
"Mario"), ("New York Philharmonic", "Zuck")]
        self.assertEqual(task3(self.many_to_many),
expected)


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

Testing: