**РК №2**

**Парадигмы и конструкции языков программирования**

**19 Вариант**

**ИБМ3-34Б**

**Примакова Александра**

import unittest

class Detail:

"""Деталь"""

def \_\_init\_\_(self, id, name, description, manufacturer\_id, стоимость):

self.id = id

self.name = name

self.description = description

self.manufacturer\_id = manufacturer\_id

self.стоимость = стоимость

class Manufacturer:

"""Производитель"""

def \_\_init\_\_(self, id, name, country):

self.id = id

self.name = name

self.country = country

class DetailManufacturer:

"""

'Детали производителя' для реализации

связи многие-ко-многим

"""

def \_\_init\_\_(self, manufacturer\_id, detail\_id):

self.manufacturer\_id = manufacturer\_id

self.detail\_id = detail\_id

def get\_one\_to\_many(manufacturers, details):

"""Задание А1: Соединение данных один-ко-многим"""

return [(d.name, d.стоимость, m.name)

for m in manufacturers

for d in details

if d.manufacturer\_id == m.id]

def get\_total\_cost(details, manufacturers):

"""Задание А2: Суммарная стоимость деталей каждого производителя"""

total\_cost = {}

for d in details:

manufacturer\_id = d.manufacturer\_id

total\_cost[manufacturer\_id] = total\_cost.get(manufacturer\_id, 0) + d.стоимость

return [(m.name, total\_cost.get(m.id, 0)) for m in manufacturers]

def get\_many\_to\_many(manufacturers, details\_manufacturers, details):

"""Задание А3: Список производителей с деталями"""

many\_to\_many\_temp = [(m.name, dm.manufacturer\_id, dm.detail\_id)

for m in manufacturers

for dm in details\_manufacturers

if m.id == dm.manufacturer\_id]

many\_to\_many = {}

for manufacturer\_name, \_, detail\_id in many\_to\_many\_temp:

detail\_name = next((d.name for d in details if d.id == detail\_id), None) #Handle missing detail

if manufacturer\_name not in many\_to\_many:

many\_to\_many[manufacturer\_name] = []

many\_to\_many[manufacturer\_name].append(detail\_name)

return many\_to\_many

#Data - separated for easier testing

manufacturers = [

Manufacturer(1, 'Компания A', 'Россия'),

Manufacturer(2, 'Компания B', 'Германия'),

Manufacturer(3, 'Компания C', 'Китай'),

]

details = [

Detail(1, 'Гайка', 'Гайка M8', 1, 10),

Detail(2, 'Шайба', 'Шайба под гайку M8', 1, 5),

Detail(3, 'Болт', 'Болт с шестигранной головкой M8', 1, 15),

Detail(4, 'Подшипник', 'Шариковый подшипник 6203', 2, 25),

Detail(5, 'Шестерня', 'Шестерня 10 зубьев', 3, 30),

]

details\_manufacturers = [

DetailManufacturer(1, 1),

DetailManufacturer(1, 2),

DetailManufacturer(1, 3),

DetailManufacturer(2, 4),

DetailManufacturer(3, 5),

]

def main():

"""Основная функция"""

print("Задание А1")

print(sorted(get\_one\_to\_many(manufacturers, details)))

print("Задание А2")

print(get\_total\_cost(details, manufacturers))

print("Задание А3")

print(get\_many\_to\_many(manufacturers, details\_manufacturers, details))

if \_\_name\_\_ == "\_\_main\_\_":

main()

class TestDataProcessing(unittest.TestCase):

def test\_get\_one\_to\_many(self):

self.assertEqual(len(get\_one\_to\_many(manufacturers, details)), 5)

def test\_get\_total\_cost(self):

self.assertEqual(get\_total\_cost(details, manufacturers), [('Компания A', 30), ('Компания B', 25), ('Компания C', 30)])

def test\_get\_many\_to\_many(self):

expected = {'Компания A': ['Гайка', 'Шайба', 'Болт'], 'Компания B': ['Подшипник'], 'Компания C': ['Шестерня']}

self.assertEqual(get\_many\_to\_many(manufacturers, details\_manufacturers, details), expected)

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main(argv=['first-arg-is-ignored'], exit=False)

Результат выполнения:

