|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | | | |
|  | Институт информационных технологий (ИТ) |
|  | Кафедра инструментального и прикладного программного обеспечения (ИиППО) |

|  |  |  |  |
| --- | --- | --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №6** | | | |
| **по дисциплине** | | | |
| **«Тестирование и верификация программного обеспечения» на тему**  **«Разработка через тестирование»** | | | |
| Выполнил студент группы ИКБО-20-19 | | Московка А.А. | |
|  | |  | |
| Принял | | Мельников Д.А. | |
| Практические работы выполнены | «\_10\_»\_декабря\_2021 г. | | Московка А.А.  (подпись студента) | |
| «Зачтено» | «\_\_»\_\_2021 г. | | Мельников Д.А.  (подпись руководителя) | |
|  |  | |  | |

Москва 2021

# Контракты на программный код (контрактное программирование)

На основе изучения материала лекций по дисциплине «Тестирование и верификация программного обеспечения» требуется выполнить следующее.

1. Реализовать взаимодействующую объектно-ориентированную систему (несколько объектов). Можно использовать темы заданий по объектно-ориентированному программированию.
2. Логика должна быть реализована внутри объектов, также должен быть объект приложения, который создает все объекты, запускает взаимодействие и печатает состояние.
3. Выделить ограничения на свойства и результаты поведения. Описать контракты на объекты.
4. Реализовать взаимодействующую систему с помощью принципа SCOOP. Для этого каждый объект должен работать в бесконечном цикле и взаимодействовать с другими объектами. Использовать пример со спящим парикмахером.
5. Добавить контракты в уже имеющуюся программную систему на

.NET.

1. Всю обработку ошибок произвести с помощью описания контрактов

и их проверки.

1. Реализовать взаимодействующую объектно-ориентированную систему (несколько объектов). Можно использовать темы заданий по объектно-ориентированному программированию.

В данной практической работе был реализован объектно-ориентированный калькулятор:

Листинг 1 – Код калькулятора

from contracts import contract  
  
  
class Add:  
 @contract  
 def \_\_init\_\_(self, x: '(int|float)', y: '(int|float)'):  
 self.number1 = x  
 self.number2 = y  
  
 @contract  
 def add(self) -> '(int|float)':  
 return self.number1 + self.number2  
  
  
class Sub:  
 @contract  
 def \_\_init\_\_(self, x: '(int|float)', y: '(int|float)'):  
 self.number1 = x  
 self.number2 = y  
  
 @contract  
 def sub(self) -> '(int|float)':  
 return self.number1 - self.number2  
  
  
class Mul:  
 @contract  
 def \_\_init\_\_(self, x: '(int|float)', y: '(int|float)'):  
 self.number1 = x  
 self.number2 = y  
  
 @contract  
 def mul(self) -> '(int|float)':  
 return self.number1 \* self.number2  
  
  
class Div:  
 @contract  
 def \_\_init\_\_(self, x: '(int|float)', y: '(int|float)'):  
 self.number1 = x  
 self.number2 = y  
  
 @contract  
 def div(self) -> '(int|float)':  
 return self.number1 / self.number2  
  
  
class Calculator(Add, Sub, Mul, Div):  
 @contract  
 def \_\_init\_\_(self, x: '(int|float)', y: '(int|float)', op: 'str'):  
 self.number1 = x  
 self.number2 = y  
 self.operation = op  
  
 @contract  
 def performOperation(self) -> '(int|float)':  
 if self.operation == '+':  
 a = Add(self.number1, self.number2)  
 return a.add()  
 elif self.operation == '-':  
 a = Sub(self.number1, self.number2)  
 return a.sub()  
 if self.operation == '\*':  
 a = Mul(self.number1, self.number2)  
 return a.mul()  
 if self.operation == '/':  
 a = Div(self.number1, self.number2)  
 return a.div()  
  
  
class CalculatorHandler:  
 def \_\_init\_\_(self) -> None:  
 print(f"1 + 10 = {Calculator(1, 10, '+').performOperation()}")  
 print(f"15 - 6 = {Calculator(15, 6, '-').performOperation()}")  
 print(f"2 \* 40 = {Calculator(2, 40, '\*').performOperation()}")  
 print(f"30 / 15 = {Calculator(30, 15, '/').performOperation()}")  
  
  
def main():  
 CalculatorHandler()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

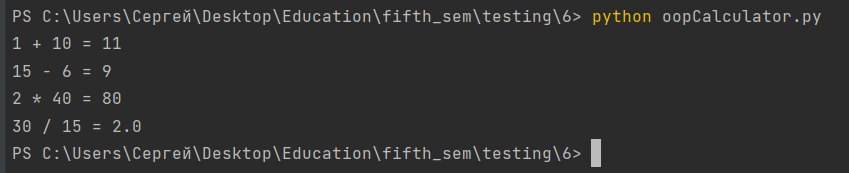


Рисунок 1 - Результат работы ООП Калькулятора

1. Логика должна быть реализована внутри объектов, также должен быть объект приложения, который создает все объекты, запускает взаимодействие и печатает состояние.

Логика представляет собой математические операции внутри каждого объекта.

1. Выделить ограничения на свойства и результаты поведения. Описать контракты на объекты.

Объект add складывает два экземпляра класса. Объект sub вычитает из первого экземпляра класса второй. Объект mul перемножает между собой два экземпляра класса. Объект calculator делает все вышеперечисленные операции.

1. Реализовать взаимодействующую систему с помощью принципа SCOOP. Для этого каждый объект должен работать в бесконечном цикле и взаимодействовать с другими объектами. Использовать пример со спящим парикмахером.

Листинг 2 – Реализация Спящего парикмахера.

from threading import Thread, Lock, Event  
from contracts import contract  
import time  
import random  
import names  
  
  
mutex = Lock()  
  
customerIntervalMin = 5  
customerIntervalMax = 15  
haircutDurationMin = 3  
haircutDurationMax = 15  
  
  
class BarberShop:  
 waitingCustomers = []  
  
 @contract  
 def \_\_init\_\_(self, barber, numberOfSeats: 'int, >0'):  
 self.barber = barber  
 self.numberOfSeats = numberOfSeats  
 print(f"BarberShop initilized with {numberOfSeats} seats")  
 print(f"Customer min interval {customerIntervalMin}")  
 print(f"Customer max interval {customerIntervalMax}")  
 print(f"Haircut min duration {haircutDurationMin}")  
 print(f"Haircut max duration {customerIntervalMax}")  
 print('---------------------------------------')  
  
 def openShop(self):  
 print('Barber shop is opening')  
 workingThread = Thread(target=self.barberGoToWork)  
 workingThread.start()  
  
 def barberGoToWork(self):  
 while True:  
 mutex.acquire()  
  
 if len(self.waitingCustomers) > 0:  
 c = self.waitingCustomers[0]  
 del self.waitingCustomers[0]  
 mutex.release()  
 self.barber.cutHair(c)  
 else:  
 mutex.release()  
 print('Aaah, all done, going to sleep')  
 barber.sleep()  
 print('Barber woke up')  
  
 def enterBarberShop(self, customer):  
 mutex.acquire()  
 print(f'>> {customer.name} entered the shop and is looking for a seat')  
  
 if len(self.waitingCustomers) == self.numberOfSeats:  
 print(f'Waiting room is full, {customer.name} is leaving.')  
 mutex.release()  
 else:  
 print(f'{customer.name} sat down in the waiting room')  
 self.waitingCustomers.append(c)  
 mutex.release()  
 barber.wakeUp()  
  
  
class Customer:  
 @contract  
 def \_\_init\_\_(self, name: 'str'):  
 self.name = name  
  
  
class Barber:  
 barberWorkingEvent = Event()  
  
 def sleep(self):  
 self.barberWorkingEvent.wait()  
  
 def wakeUp(self):  
 self.barberWorkingEvent.set()  
  
 def cutHair(self, customer):  
 self.barberWorkingEvent.clear()  
  
 print(f'{customer.name} is having a haircut')  
  
 randomHairCuttingTime = random.randrange(  
 haircutDurationMin, haircutDurationMax+1)  
 time.sleep(randomHairCuttingTime)  
 print(f'{customer.name} is done')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 customers = []  
 for \_ in range(15):  
 customers.append(Customer(names.get\_first\_name()))  
  
 barber = Barber()  
  
 barberShop = BarberShop(barber, numberOfSeats=5)  
 barberShop.openShop()  
  
 while len(customers) > 0:  
 c = customers.pop()  
 barberShop.enterBarberShop(c)  
 customerInterval = random.randrange(  
 customerIntervalMin, customerIntervalMax+1)  
 time.sleep(customerInterval)

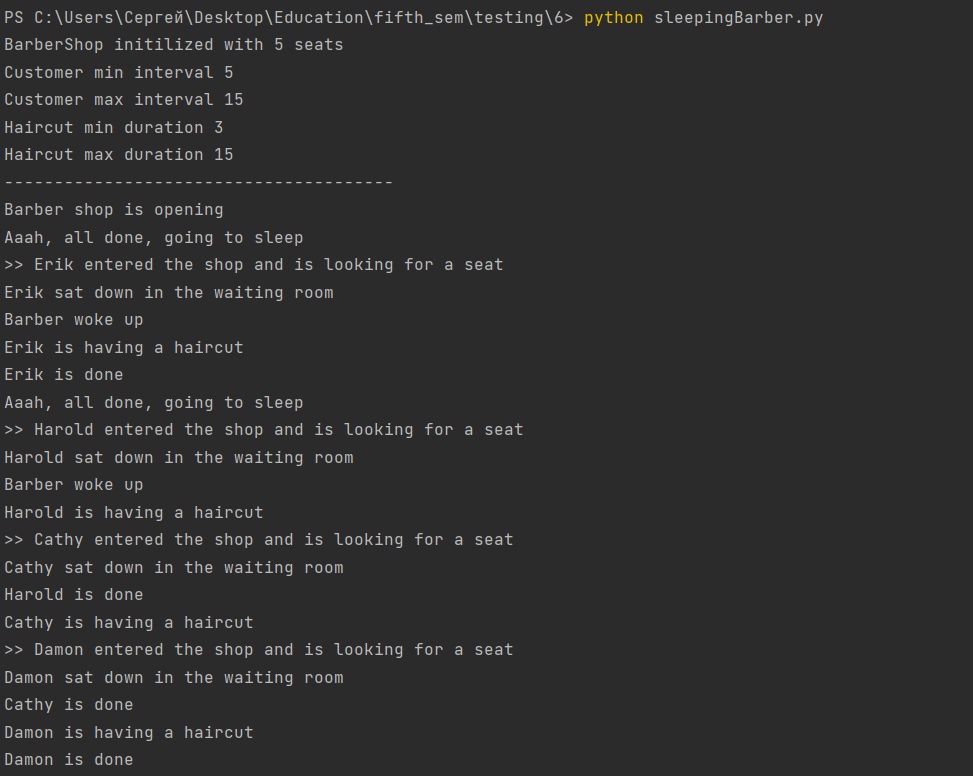


Рисунок 2 - Результат работы Спящего парикмахера

1. Всю обработку ошибок произвести с помощью описания контрактов

и их проверки.

Пример работы программы без учета контрактов (без установленной заранее библиотеки).



Рисунок 3 - Пример работы программы

**Вывод:**

На основе выполнения данной практической работы и изучения материала лекций, были выполнены следующие задачи:

1. Реализовать взаимодействующую объектно-ориентированную систему (несколько объектов). Можно использовать темы заданий по объектно-ориентированному программированию.
2. Логика должна быть реализована внутри объектов, также должен быть объект приложения, который создает все объекты, запускает взаимодействие и печатает состояние.
3. Выделить ограничения на свойства и результаты поведения. Описать контракты на объекты.
4. Реализовать взаимодействующую систему с помощью принципа SCOOP. Для этого каждый объект должен работать в бесконечном цикле и взаимодействовать с другими объектами. Использовать пример со спящим парикмахером.
5. Добавить контракты в уже имеющуюся программную систему на

.NET.

1. Всю обработку ошибок произвести с помощью описания контрактов

и их проверки.

А также были получены теоретические и практические навыки разработки и реализации взаимодействующих систем при помощи принципа SCOOP и использования контрактов.

## Список литературы

1. Алпатов, А. Н. Тестирование и отладка программного обеспечения : методические указания / А. Н. Алпатов. — Москва : РТУ МИРЭА, 2020. — 40 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/167578 (дата обращения: 24.09.2021). — Режим доступа: для авториз. пользователей.
2. Карпович, Е. Е. Методы тестирования и отладки программного обеспечения : учебник / Е. Е. Карпович. — Москва : МИСИС, 2020. — 136 с. — ISBN 978-5-907226-64-7. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/147965 (дата обращения: 24.09.2021). — Режим доступа: для авториз. пользователей.