Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Отчет по лабораторной работе №2 по курсу «Разработка интернет-приложений»

Выполнила:
студент группы ИУ5-54Б
Подопригорова Н. С.

Проверил:
Доцент кафедры ИУ5
Гапанюк Ю. Э.

Москва, 2020 г.

1. Описание задания

1. Необходимо для произвольной предметной области реализовать три шаблона проектирования: один порождающий, один структурный и один поведенческий.
2. Для каждой реализации шаблона необходимо написать модульный тест. В модульных тестах необходимо применить следующие технологии:
   - TDD - фреймворк.
   - BDD - фреймворк.
   - Создание Mock-объектов.

В данном случае я реализую шаблоны строитель, состояния и фасад.

2. Текст программы

cell.py
```
from abc import ABC, abstractmethod
import random

class State(ABC):

    @property
    def context(self):
        return self._context

    @context.setter
    def context(self, context) -> None:
        self._context = context

    def execute(self) -> bool:
        """ Main body of the state. """
        pass

class Cell():

    _state = None

    def __init__(self, state = State(), name = "", energy = None)
-> None:
        self.name = name
        self.nucleus = False
        self.membrane = False
        self.wall = False
        self.mitochondrion = False
        self.cytoplasm = False
        self.centriole = False

        if energy is None:
            self.energy = random.uniform(0, 1)
        else:
            self.energy = energy
        self.transition_to(state)

    def change_energy(self, diff: float) -> None:
        self.energy += diff
        if self.energy > 1:
            self.energy = 1

    def transition_to(self, state) -> str:
        results = []
        results.append("{} cell: Transition to 
{}".format(self.name, type(state).__name__))
#        if not suppress_output:
#            print("{} cell: Transition to {}".format(self.name, 
type(state).__name__))
        self._state = state
        self._state.context = self
        return "\n".join(results)

    def live(self) -> str:
        result = []
        while(True):
            indicator, answer = self._state.execute()
            result.append(answer)
            if indicator == False:
                break
        return "\n".join(result)

    def __str__(self) -> None:
        dict_parts = vars(self).copy()
        dict_parts.pop('name', None)
        dict_parts.pop('_state', None)
        dict_parts.pop('energy', None)
        list_parts = list(filter(lambda key: dict_parts.get(key), 
list(dict_parts.keys())))
        if not list_parts:
            return "{} cell contains nothing and has {} 
energy".format(self.name, self.energy)
        return "{} cell contains {} and has {} 
energy".format(self.name, ', '.join(list_parts), self.energy)
```

cell_builder.py

```
# Реализация шаблона Строитель
from cell import Cell
from cell_states import CellStateA

class CellBuilder():
    def __init__(self) -> None:
        self.reset()

    def reset(self) -> None:
        self._cell = Cell(CellStateA())

    @property
    def cell(self) -> Cell:
        cell = self._cell
        self.reset()
        return cell

    def set_name(self, name) -> None:
        self._cell.name = name

    def add_nucleus(self) -> None:
        self._cell.nucleus = True

    def add_membrane(self) -> None:
        self._cell.membrane = True

    def add_wall(self) -> None:
        self._cell.wall = True

    def add_mitochondrion(self) -> None:
        self._cell.mitochondrion = True

    def add_citoplasm(self) -> None:
        self._cell.cytoplasm = True

    def add_centriole(self) -> None:
        self._cell.centriole = True

class Nature:

    def __init__(self) -> None:
        self._builder = None
```

```python
    @property
    def builder(self) -> CellBuilder:
        return self._builder

    @builder.setter
    def builder(self, builder: CellBuilder) -> None:
        self._builder = builder

    def build_animal_cell(self) -> None:
        self.builder.set_name("Animal")
        self.builder.add_nucleus()
        self.builder.add_membrane()
        self.builder.add_mitochondrion()
        self.builder.add_citoplasm()
        self.builder.add_centriole()

    def build_bacteria_cell(self) -> None:
        self.builder.set_name("Bacterial")
        self.builder.add_membrane()
        self.builder.add_wall()
        self.builder.add_citoplasm()

    def build_plant_cell(self) -> None:
        self.builder.set_name("Plant")
        self.builder.add_nucleus()
        self.builder.add_membrane()
        self.builder.add_wall()
        self.builder.add_mitochondrion()
        self.builder.add_citoplasm()
        self.builder.add_centriole()

    def build_fungal_cell(self) -> None:
        self.builder.set_name("Fungal")
        self.builder.add_nucleus()
        self.builder.add_membrane()
        self.builder.add_wall()
        self.builder.add_mitochondrion()
        self.builder.add_citoplasm()

def main():
    nature = Nature()
    builder = CellBuilder()
    nature.builder = builder
    nature.build_animal_cell()
    print(builder.cell)

    nature.build_plant_cell()
    print(builder.cell)

    nature.build_bacteria_cell()
    print(builder.cell)

    nature.build_fungal_cell()
    print(builder.cell)

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

cell_states.py
# Реализация шаблона Состояния

```
# A -> A = 0.6
# A -> B = 0.3
# A -> C = 0.1
# B -> A = 0.9
# B -> C = 0.1
# C -> C = 1

from abc import ABC, abstractmethod
from cell import Cell, State
import random

class CellStateA(State):

    """ add energy (eat, breathes)"""
    def execute(self):
#        print("A")
        results = []
        self.context.change_energy(random.normalvariate(0.1, 
0.05))

#        print("\n{} cell eat and 
breaths".format(self.context.name))
#        print("{} cell has {} energy".format(self.context.name, 
self.context.energy))
        results.append("\n{} cell eat and 
breaths".format(self.context.name))
```

```
        results.append("{} cell has {}
energy".format(self.context.name, self.context.energy))
        results.append(self.change_state())

        return True, "\n".join(results)

    def change_state(self):
        results = []
        next_state = random.choices(['A', 'B', 'C'], weights = [1
- self.context.energy ** 2, self.context.energy ** (1/2),
(self.context.energy - 1) ** 2])
        if next_state[0] == 'B':

results.append(self.context.transition_to(CellStateB()))
        elif next_state[0] == 'C':

results.append(self.context.transition_to(CellStateC()))
        return "\n".join(results)


class CellStateB(State):
    """ duplicates """

    def execute(self):
#        print("B")
        results = []
        self.context.change_energy(-
self.context.energy*random.normalvariate(0.5, 0.1))

#        print("\n{} cell duplicates".format(self.context.name))
#        print("{} cell has {} energy".format(self.context.name,
self.context.energy))
        results.append("\n{} cell
duplicates".format(self.context.name))
        results.append("{} cell has {}
energy".format(self.context.name, self.context.energy))
        results.append(self.change_state())

        return True, "\n".join(results)

    def change_state(self):
        results = []
        next_state = random.choices(['A', 'C'], weights = [1 -
self.context.energy ** 2, (self.context.energy - 1) ** 2])
        if next_state[0] == 'A':
```

```
results.append(self.context.transition_to(CellStateA()))
        elif next_state[0] == 'C':

results.append(self.context.transition_to(CellStateC()))
        return "\n".join(results)


class CellStateC(State):
    """ dies """

    def execute(self) -> bool:
#        print("C")
        results = []
#        print("\n{} cell dies".format(self.context.name))
        results.append("\n{} cell dies".format(self.context.name))
        return False, "\n".join(results)

def main():
    cell = Cell(CellStateA(), "Fungal")
    print(cell.live())

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

world.py
# Реализация шаблона Фасад

```
from cell_builder import Nature, CellBuilder
from cell import Cell
from cell_states import CellStateA, CellStateB, CellStateC

class World:

    def __init__(self, nature: Nature, builder: CellBuilder) ->
None:
        self._cells = []
        self._nature = nature or Nature()
        self._builder = builder or CellBuilder()

    def add_four_cells(self) -> None:
        # строители создают клетки и помещают их в список cells
        self._nature.builder = self._builder

        self._nature.build_animal_cell()
        self._cells.append(self._builder.cell)

        self._nature.build_plant_cell()
        self._cells.append(self._builder.cell)

        self._nature.build_bacteria_cell()
        self._cells.append(self._builder.cell)

        self._nature.build_fungal_cell()
        self._cells.append(self._builder.cell)

    def operation(self) -> str:
        results = []
        self.add_four_cells()
        for cell in self._cells:
            results.append(cell.__str__())
            results.append(cell.live())
            results.append("\n
-------------------------------------- \n")
        return "\n".join(results)

def main(world):
    print(world.operation())


if __name__ == "__main__":
    nature = Nature()
    builder = CellBuilder()

    world = World(nature, builder)
    main(world)
```

test_builder.py

```
import unittest
from cell_builder import CellBuilder, Nature
from cell import Cell, State
from cell_states import CellStateA

class TestNature(unittest.TestCase):
    def setUp(self):
        self._nature = Nature()
        self._builder = CellBuilder()
        self._nature.builder = self._builder

    def test_build_animal_cell(self):
        self._nature.build_animal_cell()
        self.assertIn("Animal cell contains nucleus, membrane,
mitochondrion, cytoplasm, centriole",
self._builder.cell.__str__())
        self.assertIsInstance(self._builder.cell, Cell)

    def test_build_plant_cell(self):
        self._nature.build_plant_cell()
        self.assertIn("Plant cell contains nucleus, membrane,
wall, mitochondrion, cytoplasm, centriole",
self._builder.cell.__str__())

    def test_build_bacteria_cell(self):
        self._nature.build_bacteria_cell()
        self.assertIn("Bacterial cell contains membrane, wall,
cytoplas", self._builder.cell.__str__())

    def test_build_fungal_cell(self):
        self._nature.build_fungal_cell()
        self.assertIn("Fungal cell contains nucleus, membrane,
wall, mitochondrion, cytoplas", self._builder.cell.__str__())

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

test_cell_states.py

```
import unittest
from unittest.mock import Mock

from cell_states import CellStateA
from cell import Cell

class TestCellStateA(unittest.TestCase):

    def setUp(self):
        self.mockCell = Mock()
        self.mockCell.energy = 1.
        self.mockCell.name = "Mushroom"
        self.mockCell.change_energy = Mock(return_value = None)
        self.mockCell.transition_to = Mock(return_value =
"Transition to CellStateE")
        self.mockCell.live = Mock(return_value = "I'm alive")

        self.state = CellStateA()
        self.state.context = self.mockCell
    def test_execute(self):
        self.assertIn("Mushroom cell has 1.0 energy",
self.state.execute()[1])
        self.assertEqual(True, self.state.execute()[0])

    def test_change_state(self):
        self.assertEqual("Transition to CellStateE",
self.state.change_state())

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

test_world.py

```
import unittest
from unittest.mock import Mock

from cell_builder import Nature, CellBuilder
from world import World
from cell import Cell

class TestWorld(unittest.TestCase):
    def setUp(self):
        self.nature = Nature()
        self.builder = CellBuilder()
        self.world = World(self.nature, self.builder)

    def test_add_four_cells(self):
        self.world.add_four_cells()
        self.assertIsInstance(self.world._cells[0], Cell)
        self.assertEqual(len(self.world._cells), 4)

    def test_operation(self):
        result = self.world.operation()
        self.assertIn("Fungal cell dies", result)

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

3. Примеры выполнения программы.

cell_builder.py

```
Animal cell contains nucleus, membrane, mitochondrion, cytoplasm, centriole and
has 0.84484951328121 energy
```

```
Plant cell contains nucleus, membrane, wall, mitochondrion, cytoplasm, centriole and has 0.6367726002790847 energy
Bacterial cell contains membrane, wall, cytoplasm and has 0.23451426148273735 energy
Fungal cell contains nucleus, membrane, wall, mitochondrion, cytoplasm and has 0.08374290397401551 energy
```

cell_states.py

```
Fungal cell eat and breaths
Fungal cell has 0.26379176478694144 energy
Fungal cell: Transition to CellStateB

Fungal cell duplicates
Fungal cell has 0.1015037482825828 energy
Fungal cell: Transition to CellStateA

Fungal cell eat and breaths
Fungal cell has 0.2735669907485787 energy
Fungal cell: Transition to CellStateB

Fungal cell duplicates
Fungal cell has 0.12463163530402863 energy
Fungal cell: Transition to CellStateC

Fungal cell dies
```

world.py

```
Animal cell contains nucleus, membrane, mitochondrion, cytoplasm, centriole and has 0.5118478325290264 energy

Animal cell eat and breaths
Animal cell has 0.6344813691607085 energy
Animal cell: Transition to CellStateB

Animal cell duplicates
Animal cell has 0.342233539192723 energy
Animal cell: Transition to CellStateA

Animal cell eat and breaths
Animal cell has 0.481339251661587 energy


Animal cell eat and breaths
Animal cell has 0.569518802076691 energy
Animal cell eat and breaths
Animal cell has 0.6558694847589099 energy
Animal cell: Transition to CellStateB

Animal cell duplicates
Animal cell has 0.2469648683675032 energy
Animal cell: Transition to CellStateC

Animal cell dies

 -------------------------------------

Plant cell contains nucleus, membrane, wall, mitochondrion, cytoplasm, centriole and
has 0.8460731595599891 energy

Plant cell eat and breaths
Plant cell has 0.9460547755670118 energy
Plant cell: Transition to CellStateB

Plant cell duplicates
Plant cell has 0.48903281897456125 energy
Plant cell: Transition to CellStateA

Plant cell eat and breaths
Plant cell has 0.5807197287100576 energy


Plant cell eat and breaths
Plant cell has 0.6923454582535086 energy
Plant cell: Transition to CellStateB

Plant cell duplicates
Plant cell has 0.17719431298729982 energy
Plant cell: Transition to CellStateC

Plant cell dies

 -------------------------------------

Bacterial cell contains membrane, wall, cytoplasm and has 0.17024904519126227
energy

Bacterial cell eat and breaths
Bacterial cell has 0.2692291763740476 energy
```

Bacterial cell: Transition to CellStateC

Bacterial cell dies

Fungal cell contains nucleus, membrane, wall, mitochondrion, cytoplasm and has 0.9987973249182323 energy

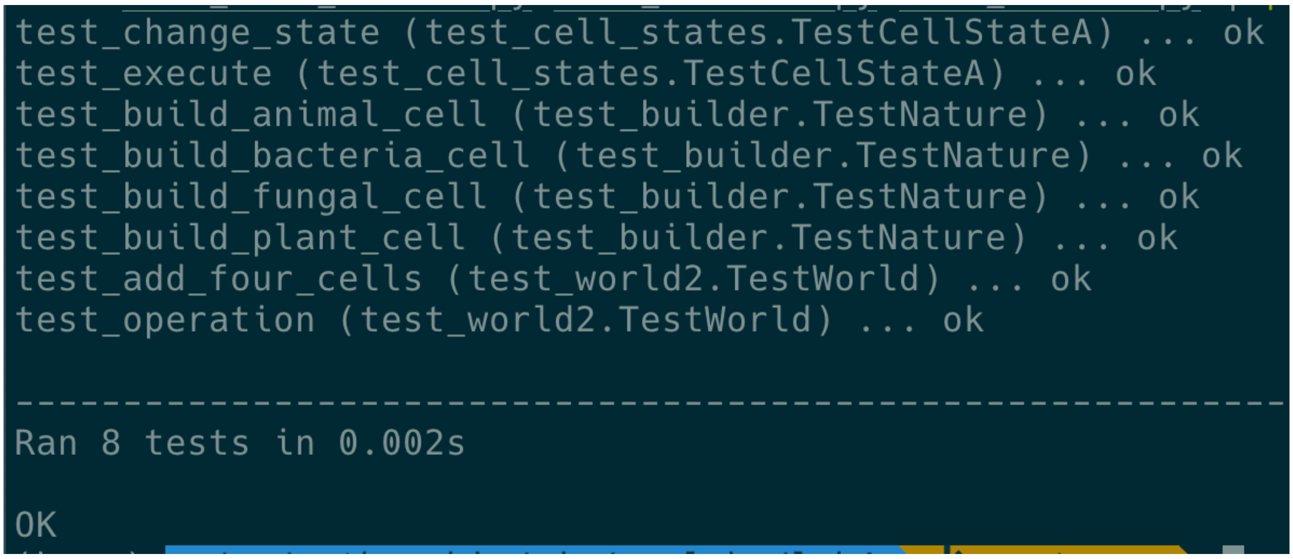
Fungal cell eat and breaths
Fungal cell has 1 energy
Fungal cell: Transition to CellStateB

Fungal cell duplicates
Fungal cell has 0.6007878067188772 energy
Fungal cell: Transition to CellStateA

Fungal cell eat and breaths
Fungal cell has 0.7152061409840661 energy
Fungal cell: Transition to CellStateC

Fungal cell dies

Тесты:



```
test_change_state (test_cell_states.TestCellStateA) ... ok
test_execute (test_cell_states.TestCellStateA) ... ok
test_build_animal_cell (test_builder.TestNature) ... ok
test_build_bacteria_cell (test_builder.TestNature) ... ok
test_build_fungal_cell (test_builder.TestNature) ... ok
test_build_plant_cell (test_builder.TestNature) ... ok
test_add_four_cells (test_world2.TestWorld) ... ok
test_operation (test_world2.TestWorld) ... ok

----------------------------------------------------------------------
Ran 8 tests in 0.002s

OK
```