**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**
**(БГТУ им. В.Г. Шухова)**

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

# Лабораторная работа №14

по дисциплине: ООП
тема: «Тестирование. Знакомство с TDD. Тесты как способ формирования архитектуры.»

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили:
пр. Черников Сергей Викторович

Белгород 2024 г.

# Лабораторная работа №14
«Тестирование. Знакомство с TDD. Тесты как способ формирования архитектуры.»
Вариант 5

**Цель работы:** знакомство с понятием тестирования. Получение практических навыков для написания модульных тестов.

Разработать ПО зоопарк, которые содержит несколько Львов. Каждый Лев умеет рычать, есть мясо и точить когти.

```python
from typing import List


class Food:
    def __init__(self):
        self.__edible = True

    def eat(self):
        self.__edible = False

    def edible(self):
        return self.__edible


class Lion:
    def __init__(self):
        self.satiety = 5

    def roar(self):
        result = "Слишком лень!" if self.is_satisfied() else "Рррры!"
        self.satiety -= 1
        return result

    def eat(self, food: Food):
        if food.edible():
            food.eat()
            self.satiety += 1

    def scratch(self):
        result = "Ккрырыры" if self.is_satisfied() else "Я слишком голоден для этого!"
        self.satiety -= 1
        return result

    @property
    def satiety(self):
        return self.__satiety

    @satiety.setter
    def satiety(self, value):
        if value > 10:
            self.__satiety = 10
        elif value < 0:
            self.__satiety = 0
        else:
            self.__satiety = value

    def is_satisfied(self):
        return self.satiety >= 5


class Zoo:
    def __init__(self, animals: List[Lion]):
        self.__animals = animals

    def add_animal(self, animal: Lion):
        self.__animals.append(animal)

    def set_animal(self, index: int, animal: Lion):
        self.__animals[index] = animal

    def remove_animal(self, index: int):
        self.__animals.pop(index)

    def get_animals(self):
        return tuple(self.__animals)


if __name__ == "__main__":
    pass
```

---

---

```
from src.lab14.ver1.core.main import Food, Lion, Zoo

import pytest


@pytest.fixture(scope='function')
def food():
    return Food()


@pytest.fixture(scope='session')
def many_plant_food():
    return (Food() for x in range(0, 999999999))


@pytest.fixture(scope='function')
def dingodog():
    return Lion()


@pytest.fixture(scope='session')
def many_lions():
    return (Lion() for x in range(0, 999999999))
```

```python
@pytest.fixture(scope='function')
def empty_zoo():
    return Zoo([])


@pytest.fixture(scope='function')
def filled_zoo(many_lions):
    return Zoo([next(many_lions) for _ in range(0, 5)])
```

---

---

```python
import src.lab14.ver1.test.conftest

import pytest


def test_food(food):
    assert food.edible()
    food.eat()
    assert not food.edible()
```

---

---

```python
import src.lab14.ver1.test.conftest

import pytest


def test_lion(dingodog, many_plant_food):
    satiety_before = dingodog.satiety
    food = next(many_plant_food)
    dingodog.eat(food)

    assert dingodog.satiety > satiety_before
    assert not food.edible()
    satiety_before = dingodog.satiety
    dingodog.eat(food)
    assert dingodog.satiety == satiety_before

    while not dingodog.is_satisfied():
        food = next(many_plant_food)
        dingodog.eat(food)

    assert dingodog.roar() == "Слишком лень!"
    while dingodog.is_satisfied():
        dingodog.roar()

    assert dingodog.roar() == "Рррры!"

    while not dingodog.is_satisfied():
        food = next(many_plant_food)
        dingodog.eat(food)

    assert dingodog.scratch() == "Ккрырырыы"
    while dingodog.is_satisfied():
        dingodog.scratch()

    assert dingodog.scratch() == "Я слишком голоден для этого!"
```

```
import src.lab14.ver1.test.conftest

import pytest


def test_zoo(empty_zoo, filled_zoo, many_plant_food):
    zoo = filled_zoo

    lion = zoo.get_animals()[0]
    satiety_before = lion.satiety
    while lion.satiety == satiety_before:
        lion.eat(next(many_plant_food))

    satiety_after = lion.satiety
    assert zoo.get_animals()[0].satiety == satiety_after

    len_before = len(zoo.get_animals())
    zoo.remove_animal(0)
    assert len(zoo.get_animals()) == len_before - 1
    zoo.add_animal(lion)
    assert zoo.get_animals()[-1] == lion
```

Результат выполнения тестов

```
============================= test session starts =============================
collecting ... collected 3 items

test_food.py::test_food PASSED                                           [ 33%]
test_lion.py::test_lion PASSED                                           [ 66%]
test_zoo.py::test_zoo PASSED                                             [100%]

============================== 3 passed in 0.02s ==============================

Process finished with exit code 0
```

В зоопарк прибыли новые животные, гиены и дикие собаки динго. Предусмотреть архитектуру так, что могут появляется новые виды животных, а также животные с несвойственным поведением. А именно Львы вегетарианцы.

```python
from abc import ABC
from io import UnsupportedOperation

from src.lab14.ver2.core.food.food import Food, Plant


class Animal(ABC):
    def __unsupported(*args, **kwargs):
        raise UnsupportedOperation('Method doesn\'t exists')

    def __init__(self, eat_func=__unsupported, scratch_func=__unsupported, roar_func=__unsupported, satiety_since=5,
    ↪   satiety_default=5):
        self.satiety = satiety_default
        self.satiety_since = satiety_since
        self.eat_func = eat_func
        self.scratch_func = scratch_func
        self.roar_func = roar_func

    def roar(self):
        return self.roar_func(self)

    def eat(self, food: Food):
        return self.eat_func(self, food)

    def scratch(self):
        return self.scratch_func(self)

    @property
    def satiety(self):
        return self.__satiety

    @satiety.setter
    def satiety(self, value):
        if value > 10:
            self.__satiety = 10
        elif value < 0:
            self.__satiety = 0
        else:
            self.__satiety = value

    def is_satisfied(self):
        return self.satiety >= 5


class Lion(Animal):
    def __init__(self):
        super().__init__(roar_func=Lion.__roar_func, eat_func=Lion.__eat_func, scratch_func=Lion.__scratch_func,
                         satiety_default=5, satiety_since=5)

    def __roar_func(self):
        result = "Слишком лень!" if self.is_satisfied() else "Ррррь!"
        self.satiety -= 1
        return result
    def __eat_func(self, food: Food):
        if food.edible():
            food.eat()
            self.satiety += 1

    def __scratch_func(self):
        result = "Ккрырыры" if self.is_satisfied() else "Я слишком голоден для этого!"
        self.satiety -= 1
        return result


class LionVegetarian(Lion):
    def __init__(self):
        super().__init__()
        self.eat_func = LionVegetarian.__eat_func

    def __eat_func(self, food: Food):
        if food.edible() and isinstance(food, Plant):
            food.eat()
            self.satiety += 1


class DingoDog(Animal):
    def __init__(self):
        super().__init__(roar_func=DingoDog.__roar_func, eat_func=DingoDog.__eat_func,
                         satiety_default=5, satiety_since=3)

    def __roar_func(self):
        result = "Тяф)" if self.is_satisfied() else "Гав гав гав!!!"
        self.satiety -= 1
        return result

    def __eat_func(self, food: Food):
        if food.edible():
            food.eat()
            self.satiety += 1


class Hyena(Animal):
    def __init__(self):
        super().__init__(scratch_func=Hyena.__scratch_func, eat_func=Hyena.__eat_func,
                         satiety_default=5, satiety_since=2)

    def __scratch_func(self):
        result = "Копаем-копаем!!!" if self.is_satisfied() else "Ой устал :("
        self.satiety -= 1
        return result

    def __eat_func(self, food: Food):
        if food.edible():
            food.eat()
            self.satiety += 1
```

```python
from abc import ABC


class Food(ABC):
    def __init__(self):
        self.__edible = True

    def eat(self):
        self.__edible = False

    def edible(self):
        return self.__edible


class Meat(Food):
    pass


class Plant(Food):
    pass
```

```python
from typing import List

from src.lab14.ver2.core.animals.animals import Animal


class Zoo:
    def __init__(self, animals: List[Animal]):
        self.__animals = animals

    def add_animal(self, animal: Animal):
        self.__animals.append(animal)

    def set_animal(self, index: int, animal: Animal):
        self.__animals[index] = animal

    def remove_animal(self, index: int):
        self.__animals.pop(index)

    def get_animals(self):
        return tuple(self.__animals)
```

```python
import animals
import food
import zoo
```

```python
if __name__ == "__main__":
    pass
```

```python
import pytest
import random

from src.lab14.ver2.core.animals.animals import Lion, LionVegetarian, DingoDog, Hyena
from src.lab14.ver2.core.food.food import Meat, Plant
from src.lab14.ver2.core.zoo.zoo import Zoo


@pytest.fixture(scope='function')
def meat_food():
    return Meat()


@pytest.fixture(scope='session')
def many_meat_food():
    return (Meat() for x in range(0, 999999999))


@pytest.fixture(scope='function')
def plant_food():
    return Plant()


@pytest.fixture(scope='session')
def many_plant_food():
    return (Plant() for x in range(0, 999999999))


@pytest.fixture(scope='session')
def many_food(many_meat_food, many_plant_food):
    return (random.randint(0, 1) == 0 and next(many_meat_food) or next(many_plant_food) for i in range(0, 999999999))


@pytest.fixture(scope='function')
def lion():
    return Lion()


@pytest.fixture(scope='function')
def dingodog():
    return DingoDog()


@pytest.fixture(scope='function')
def hyena():
    return Hyena()
```

```python
@pytest.fixture(scope='function')
def vegetarian_lion():
    return LionVegetarian()


@pytest.fixture(scope='session')
def many_lions():
    return (Lion() for x in range(0, 999999999))


@pytest.fixture(scope='function')
def empty_zoo():
    return Zoo([])


@pytest.fixture(scope='function')
def filled_zoo(many_lions):
    return Zoo([next(many_lions) for _ in range(0, 5)])
```

---

```python
from io import UnsupportedOperation

import src.lab14.ver2.test.conftest

import pytest


def test_dingo_dog(dingodog, many_plant_food):
    satiety_before = dingodog.satiety
    food = next(many_plant_food)
    dingodog.eat(food)

    assert dingodog.satiety > satiety_before
    assert not food.edible()
    satiety_before = dingodog.satiety
    dingodog.eat(food)
    assert dingodog.satiety == satiety_before

    while not dingodog.is_satisfied():
        food = next(many_plant_food)
        dingodog.eat(food)

    assert dingodog.roar() == "Тяф)"
    while dingodog.is_satisfied():
        dingodog.roar()

    assert dingodog.roar() == "Гав гав гав!!!"

    while not dingodog.is_satisfied():
        food = next(many_plant_food)
        dingodog.eat(food)

    with pytest.raises(UnsupportedOperation):
        dingodog.scratch()
```

```
from io import UnsupportedOperation

import src.lab14.ver2.test.conftest

import pytest


def test_hyena(hyena, many_plant_food):
    satiety_before = hyena.satiety
    food = next(many_plant_food)
    hyena.eat(food)

    assert hyena.satiety > satiety_before
    assert not food.edible()
    satiety_before = hyena.satiety
    hyena.eat(food)
    assert hyena.satiety == satiety_before

    while not hyena.is_satisfied():
        food = next(many_plant_food)
        hyena.eat(food)

    assert hyena.scratch() == "Копаем-копаем!!!"
    while hyena.is_satisfied():
        hyena.scratch()

    assert hyena.scratch() == "Ой устал :("

    with pytest.raises(UnsupportedOperation):
        hyena.roar()
```

```
import src.lab14.ver2.test.conftest

import pytest


def test_lion(lion, many_plant_food):
    satiety_before = lion.satiety
    food = next(many_plant_food)
    lion.eat(food)

    assert lion.satiety > satiety_before
    assert not food.edible()
    satiety_before = lion.satiety
    lion.eat(food)
    assert lion.satiety == satiety_before

    while not lion.is_satisfied():
        food = next(many_plant_food)
        lion.eat(food)

    assert lion.roar() == "Слишком лень!"
    while lion.is_satisfied():
        lion.roar()

    assert lion.roar() == "Рррры!"

    while not lion.is_satisfied():
        food = next(many_plant_food)
        lion.eat(food)

    assert lion.scratch() == "Ккрырыры"
    while lion.is_satisfied():
        lion.scratch()

    assert lion.scratch() == "Я слишком голоден для этого!"
```

```
import src.lab14.ver2.test.conftest

import pytest


def test_food(meat_food):
    assert meat_food.edible()
    meat_food.eat()
    assert not meat_food.edible()
```

```
import src.lab14.ver2.test.conftest

import pytest


def test_plant_food(plant_food):
    assert plant_food.edible()
    plant_food.eat()
    assert not plant_food.edible()
```

```python
import src.lab14.ver2.test.conftest

import pytest


def test_lion(vegetarian_lion, many_plant_food, meat_food):
    satiety_before = vegetarian_lion.satiety
    food = next(many_plant_food)
    vegetarian_lion.eat(food)

    assert vegetarian_lion.satiety > satiety_before
    assert not food.edible()
    satiety_before = vegetarian_lion.satiety
    vegetarian_lion.eat(food)
    assert vegetarian_lion.satiety == satiety_before

    while not vegetarian_lion.is_satisfied():
        food = next(many_plant_food)
        vegetarian_lion.eat(food)

    assert vegetarian_lion.roar() == "Слишком лень!"
    while vegetarian_lion.is_satisfied():
        vegetarian_lion.roar()

    assert vegetarian_lion.roar() == "Рррры!"

    while not vegetarian_lion.is_satisfied():
        food = next(many_plant_food)
        vegetarian_lion.eat(food)

    assert vegetarian_lion.scratch() == "Ккрырырыы"
    while vegetarian_lion.is_satisfied():
        vegetarian_lion.scratch()

    assert vegetarian_lion.scratch() == "Я слишком голоден для этого!"

    while vegetarian_lion.satiety != 0:
        vegetarian_lion.roar()

    vegetarian_lion.eat(meat_food)
    assert vegetarian_lion.satiety == 0
    assert meat_food.edible()
```

```python
import src.lab14.ver2.test.conftest

import pytest


def test_zoo(empty_zoo, filled_zoo, many_plant_food):
    zoo = filled_zoo
    lion = zoo.get_animals()[0]
    satiety_before = lion.satiety
    while lion.satiety == satiety_before:
        lion.eat(next(many_plant_food))

    satiety_after = lion.satiety
    assert zoo.get_animals()[0].satiety == satiety_after

    len_before = len(zoo.get_animals())
    zoo.remove_animal(0)
    assert len(zoo.get_animals()) == len_before - 1
    zoo.add_animal(lion)
    assert zoo.get_animals()[-1] == lion
```

Результат выполнения тестов

```
============================= test session starts =============================
collecting ... collected 7 items

test_dingo_dog.py::test_dingo_dog PASSED                                 [ 14%]
test_hyena.py::test_hyena PASSED                                         [ 28%]
test_lion.py::test_lion PASSED                                           [ 42%]
test_meat_food.py::test_food PASSED                                      [ 57%]
test_plant_food.py::test_plant_food PASSED                               [ 71%]
test_vegetarian_lion.py::test_lion PASSED                                [ 85%]
test_zoo.py::test_zoo PASSED                                             [100%]

============================== 7 passed in 0.02s ==============================

Process finished with exit code 0
```

Ссылка на репозиторий

**Вывод:** в ходе лабораторной работы познакомились с понятием тестирования. Получили практические навыки для написания модульных тестов.