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)

Кафедра программного обеспечения

вычислительной техники и автоматизированных

систем

Лабораторная работа №14

по дисциплине: ООП

тема: «**Тестирование. Знакомство с TDD. Тесты как способ формирования архитектуры**»

Выполнил: студент группы ПВ-233
Мороз Роман Алексеевич

Проверили:
Морозов Данила Александрович

Белгород 2025

Лабораторная работа № 14

Тестирование. Знакомство с TDD. Тесты как способ формирования архитектуры

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

Разработать программу на языке программирования Python и модульные тесты с использованием библиотеки pytest в соответствии с вариантом задания.

Задание на разработку ПО в лабораторной состоит из двух частей (начальные условия задачи, финальные условия задачи). В Таблице 1 приведены условия задач.

№	Начальные условия задачи	Финальные условия задачи
1.	Разработать подсистему моделирования поведения «Утки» в озере. Утка может крякать, утка может летать и плавать.	В озере теперь могут плавать не только утки, но и манок, который в отличии от утки не может летать. Предусмотреть архитектуру с возможностью расширения путем появление новых объектов с различим поведением аспектов Утки (пингвин – не крякает, страус – не крякает, но звук издает, не плавает, не летает).
2.	Разработать подсистему для стрельбы из игрового оружия. «Игровое оружие» может выполнять стрельбу, и имеет возможность отображать себя на экран.	Теперь оружие получает способность перезарядки. Появляется такой вид оружие, как нож.
3.	Разработать подсистему эмулирования процесса открытия двери. Дверь можно открыть, закрыть. Дверь соединяет два помещения, соответственно, когда она закрыты попасть в другое помещение нельзя.	Теперь появляется возможность устанавливать разные замки на двери. Механический и цифровой. При добавлении цифрового замка должна быть специальная функция для сброса пароля.
4.	Разработать систему регистрации «Специальных задач». Каждая задача имеет характеристики время создание, требуемое время на выполнение	Теперь требуется добавить возможность сортировать задачи другими видами сортировок, сортировкой вставками, и сортировкой выбором. Так же,

	и требуемые ресурсы. Контейнер этих задач умеет сортировать эти задачи «методом пузырька».	появляется возможность указания ключа сортировки.
5.	Разработать ПО зоопарк, которые содержит несколько Львов. Каждый Лев умеет рычать, есть мясо и точить когти.	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.

В качестве результатов лабораторной работы должны быть 4 программы. Тесты для первой задачи, код первой задачи, тесты для второй задачи, код второй задачи.

Контрольные вопросы:

1. Тестирование.
2. Тест-план.
3. Виды тестирования.
4. Пирамида тестирования.
5. TDD.
6. Pytest и его архитектура работы.

```
import datetime
from typing import List, Callable, Optional

class Task:
    __slots__ = ['_creation_time', '_execution_time', '_resources']

    def __init__(
        self,
        creation_time: Optional[datetime.datetime] = None,
        execution_time: Optional[datetime.timedelta] = None,
        resources: int = 0
    ):
        self._creation_time = creation_time if creation_time else
datetime.datetime.now()
        self._execution_time = execution_time if execution_time else
datetime.timedelta(0)

        if not isinstance(resources, int) or resources < 0:
            raise ValueError("Resources must be non-negative integer")
```

```
        self._resources = resources

    @property
    def creation_time(self) -> datetime.datetime:
        return self._creation_time

    @property
    def execution_time(self) -> datetime.timedelta:
        return self._execution_time

    @property
    def resources(self) -> int:
        return self._resources

    def __eq__(self, other: object) -> bool:
        if not isinstance(other, Task):
            return NotImplemented
        return (self.creation_time == other.creation_time and
                self.execution_time == other.execution_time and
                self.resources == other.resources)

    def __repr__(self) -> str:
        return f"Task({self.creation_time!r}, {self.execution_time!r},
{self.resources})"

    def __str__(self) -> str:
        return (f"Task [Created: {self.creation_time.strftime('%Y-%m-%d
%H:%M')}, "
                f"Duration: {self.execution_time}, Resources:
{self.resources}]")

class TaskContainer:
    __slots__ = ['_tasks']

    def __init__(self, tasks: Optional[List[Task]] = None):
        self._tasks = list(tasks) if tasks else []

    @property
    def tasks(self) -> List[Task]:
        return self._tasks

    def add_task(self, task: Task) -> None:
        if not isinstance(task, Task):
```

```
            raise TypeError("Only Task objects can be added")
        self._tasks.append(task)

    def __repr__(self) -> str:
        return f"TaskContainer({self._tasks!r})"

    def __str__(self) -> str:
        return "\n".join(str(task) for task in self._tasks)

    def bubble_sort(self, key: Callable[[Task], float] = lambda x:
x.creation_time.timestamp()) -> None:
        n = len(self._tasks)
        for i in range(n):
            for j in range(0, n-i-1):
                if key(self._tasks[j]) > key(self._tasks[j+1]):
                    self._tasks[j], self._tasks[j+1] = self._tasks[j+1],
self._tasks[j]

    def insertion_sort(self, key: Callable[[Task], float] = lambda x:
x.creation_time) -> None:
        for i in range(1, len(self._tasks)):
            current = self._tasks[i]
            j = i-1
            while j >= 0 and key(current) < key(self._tasks[j]):
                self._tasks[j+1] = self._tasks[j]
                j -= 1
            self._tasks[j+1] = current

    def selection_sort(self, key: Callable[[Task], float] = lambda x:
x.creation_time) -> None:
        for i in range(len(self._tasks)):
            min_idx = i
            for j in range(i+1, len(self._tasks)):
                if key(self._tasks[j]) < key(self._tasks[min_idx]):
                    min_idx = j
            self._tasks[i], self._tasks[min_idx] = self._tasks[min_idx],
self._tasks[i]
```

```
import pytest
import datetime
from reg_system import Task, TaskContainer

class TestTaskSystem:
```

```
    @pytest.fixture
    def sample_tasks(self):
        return [
            Task(datetime.datetime(2025, 1, 3)),
            Task(execution_time=datetime.timedelta(hours=2)),
            Task(resources=5),
            Task(datetime.datetime(2025, 1, 1)),
            Task(datetime.datetime(2025, 1, 2))
        ]

    def test_task_creation(self):
        t = Task()
        assert t.resources == 0
        assert isinstance(t.creation_time, datetime.datetime)

        ct = datetime.datetime(2025, 1, 1)
        et = datetime.timedelta(hours=1)
        t = Task(ct, et, 5)
        assert t.creation_time == ct
        assert t.execution_time == et

    def test_resource_validation(self):
        with pytest.raises(ValueError):
            Task(resources=-1)
        with pytest.raises(ValueError):
            Task(resources="invalid")

    def test_task_equality(self):
        t1 = Task(datetime.datetime(2025, 1, 1))
        t2 = Task(datetime.datetime(2025, 1, 1))
        assert t1 == t2

    def test_container_initialization(self):
        container = TaskContainer()
        assert len(container.tasks) == 0

        tasks = [Task(), Task()]
        container = TaskContainer(tasks)
        assert len(container.tasks) == 2

    def test_add_task_validation(self):
        container = TaskContainer()
        container.add_task(Task())
```

```
        with pytest.raises(TypeError):
            container.add_task("invalid")

        with pytest.raises(TypeError):
            container.add_task(123)

    def test_sorting_operations_1(self, sample_tasks):
        container = TaskContainer(sample_tasks.copy())
        container.bubble_sort(key=lambda x: x.creation_time.timestamp())
        assert [t.creation_time.day for t in container.tasks] == [1, 2, 3, 18,
18]

    def test_sorting_operations_2(self, sample_tasks):
        container = TaskContainer(sample_tasks.copy())
        container.insertion_sort(key=lambda x: x.creation_time.timestamp())
        assert [t.creation_time.day for t in container.tasks] == [1, 2, 3, 18,
18]

    def test_sorting_operations_3(self, sample_tasks):
        container = TaskContainer(sample_tasks.copy())
        container.selection_sort(key=lambda x: x.creation_time.timestamp())
        assert [t.creation_time.day for t in container.tasks] == [1, 2, 3, 18,
18]

    def test_sorting_operations_4(self, sample_tasks):
        container = TaskContainer(sample_tasks.copy())
        container.bubble_sort(key=lambda x: x.resources)
        assert [t.creation_time.day for t in container.tasks] == [3, 18, 1, 2,
18]

    def test_sorting_operations_5(self, sample_tasks):
        container = TaskContainer(sample_tasks.copy())
        container.insertion_sort(key=lambda x: x.resources)
        assert [t.creation_time.day for t in container.tasks] == [3, 18, 1, 2,
18]

    def test_sorting_operations_6(self, sample_tasks):
        container = TaskContainer(sample_tasks.copy())
        container.selection_sort(key=lambda x: x.resources)
        assert [t.creation_time.day for t in container.tasks] == [3, 18, 1, 2,
18]
```

```python
    def test_empty_container(self):
        container = TaskContainer()
        container.bubble_sort()
        assert len(container.tasks) == 0
        assert container.tasks == []

if __name__ == '__main__':
    pytest.main([__file__, '-v'])
```

```
❯ python3 test_reg_system.py
============================= test session starts ==============================
platform linux -- Python 3.12.7, pytest-7.4.4, pluggy-1.0.0 -- /home/crissyro/anaconda3/bin/python3
cachedir: .pytest_cache
rootdir: /home/crissyro/4-sem-university/oop/14_lab
plugins: typeguard-4.4.2, anyio-4.2.0
collected 12 items

test_reg_system.py::TestTaskSystem::test_task_creation PASSED            [  8%]
test_reg_system.py::TestTaskSystem::test_resource_validation PASSED      [ 16%]
test_reg_system.py::TestTaskSystem::test_task_equality PASSED            [ 25%]
test_reg_system.py::TestTaskSystem::test_container_initialization PASSED [ 33%]
test_reg_system.py::TestTaskSystem::test_add_task_validation PASSED      [ 41%]
test_reg_system.py::TestTaskSystem::test_sorting_operations_1 PASSED     [ 50%]
test_reg_system.py::TestTaskSystem::test_sorting_operations_2 PASSED     [ 58%]
test_reg_system.py::TestTaskSystem::test_sorting_operations_3 PASSED     [ 66%]
test_reg_system.py::TestTaskSystem::test_sorting_operations_4 PASSED     [ 75%]
test_reg_system.py::TestTaskSystem::test_sorting_operations_5 PASSED     [ 83%]
test_reg_system.py::TestTaskSystem::test_sorting_operations_6 PASSED     [ 91%]
test_reg_system.py::TestTaskSystem::test_empty_container PASSED          [100%]

============================== 12 passed in 0.03s ==============================


~/4-sem-university/oop/14_lab  main !1
```

Вывод: познакомились с понятием тестирования. Получили практические навыки для написания модульных тестов.