МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Омский государственный университет им. Ф.М. Достоевского»
Институт среднего профессионального образования и довузовской подготовки

Учебная практика по модулю **«Осуществление интеграции программных модулей»**

ВЫПОЛНИЛ: СТУДЕНТ 1 КУРСА

ОЧНОЙ ФОРМЫ ОБУЧЕНИЯ

ГРУППЫ ДИН-111-О

ПОЛОМОШНОВ ОЛЕГ АЛЕКСЕЕВИЧ

РУКОВОДИТЕЛЬ ПРАКТИКИ:

ГОЛЬТЯПИНН ВИКТОР ВИКТОРОВИЧ

Омск 2022

Цели и задачи практики

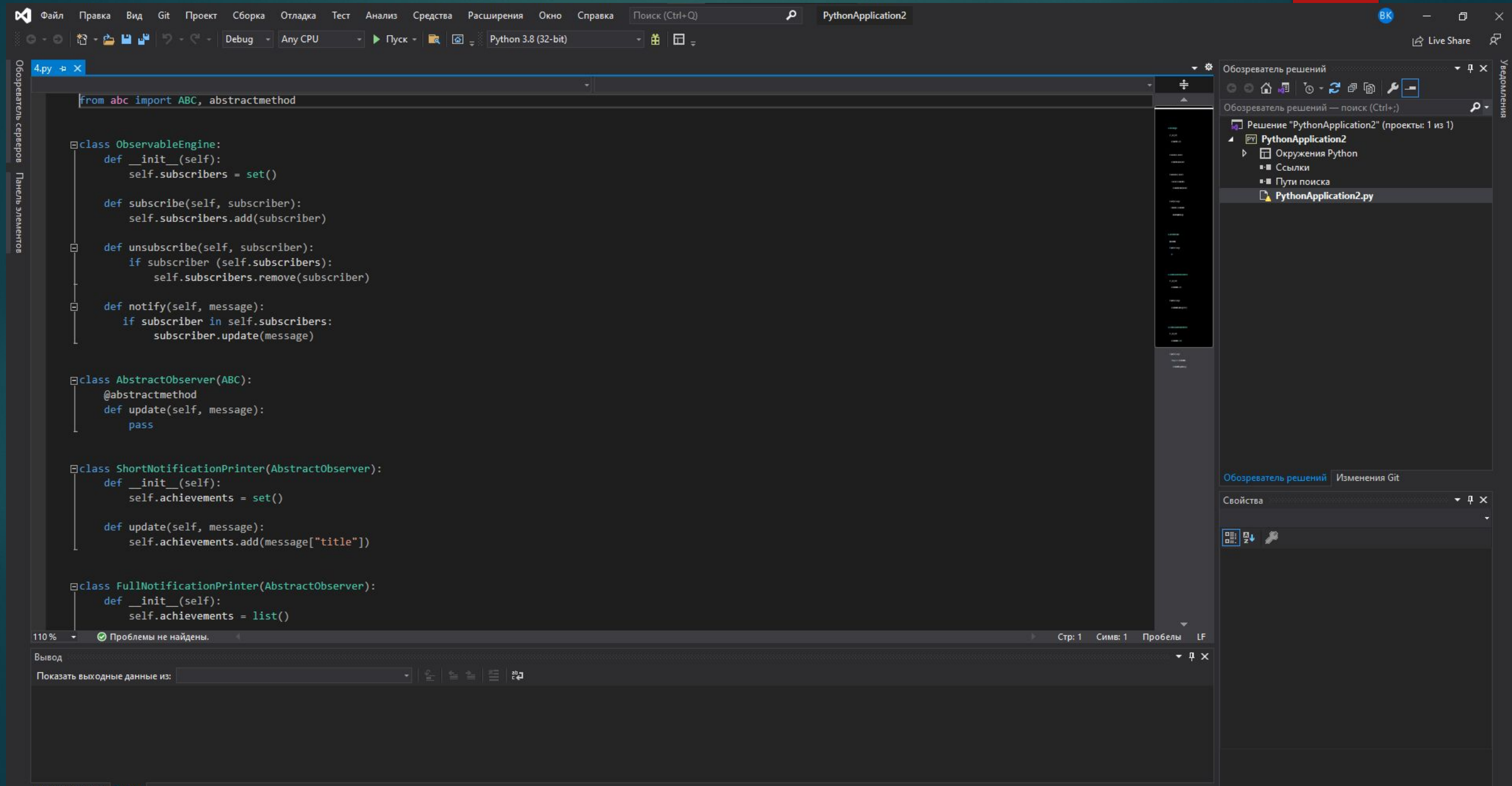
Цели:

1. Составить код для программы и его спецификацию;
2. Предоставить скриншот результатов выполнения работы кодов в среде разработки.

Задачи:

1. Создание базовых классов;
2. Декоратор класса;
3. Паттерн Адаптер;
4. Паттерн наблюдатель;
5. Абстрактная фабрика;

Инструментальное средство Visual Studio



Создание базовых классов

```python
import math
from abc import abstractmethod, ABC


class Base(ABC):
    def __init__(self, data, result):
        self.data = data
        self.result = result

    def get_answer(self):
        return [int(x >= 0.5) for x in self.data]

    def get_score(self):
        ans = self.get_answer()
        return sum([int(x == y) for x, y in zip(ans, self.result)]) / len(ans)

    @abstractmethod
    def get_loss(self):
        pass
```

Для выполнения задачи нужно импортировать абстрактные методы

```python
class A(Base):
    def get_loss(self):
        return sum(
            [(x - y) * (x - y) for (x, y) in zip(self.data, self.result)])


class B(Base):
    def get_loss(self):
        return -sum([
            y * math.log(x) + (1 - y) * math.log(1 - x)
            for (x, y) in zip(self.data, self.result)
        ])

    def get_pre(self):
        ans = self.get_answer()
        res = [int(x == 1 and y == 1) for (x, y) in zip(ans, self.result)]
        return sum(res) / sum(ans)

    def get_rec(self):
        ans = self.get_answer()
        res = [int(x == 1 and y == 1) for (x, y) in zip(ans, self.result)]
        return sum(res) / sum(self.result)

    def get_score(self):
        pre = self.get_pre()
        rec = self.get_rec()
        return 2 * pre * rec / (pre + rec)



class C(Base):
    def get_loss(self):
        return sum([abs(x - y) for (x, y) in zip(self.data, self.result)])
```

Декоратор класса

```python
from abc import ABC, abstractmethod

class Hero:
    def __init__(self):
        self.positive_effects = []
        self.negative_effects = []
        self.stats = {
            "HP": 128,
            "MP": 42,
            "SP": 100,
            "Strength": 15,
            "Perception": 4,
            "Endurance": 8,
            "Charisma": 2,
            "Intelligence": 3,
            "Agility": 8,
            "Luck": 1
        }

    def get_positive_effects(self):
        return self.positive_effects.copy()

    def get_negative_effects(self):
        return self.negative_effects.copy()

    def get_stats(self):
        return self.stats.copy()
```

```python
def get_positive_effects(self):
    return self.positive_effects.copy()

def get_negative_effects(self):
    return self.negative_effects.copy()

def get_stats(self):
    return self.stats.copy()
```

```python
class AbstractEffect(Hero, ABC):
    def __init__(self):


    @abstractmethod
    def get_positive_effects(self):
        return self.get_positive_effects()

    @abstractmethod
    def get_negative_effects(self):
        return self.get_negative_effects()

    @abstractmethod
    def get_stats(self):
        return self.get_stats()


class AbstractPositive(AbstractEffect, ABC):
    def get_negative_effects(self):
        return self.get_negative_effects()


class AbstractNegative(AbstractEffect, ABC):
    def get_positive_effects(self):
        return self.get_positive_effects()
```

Код задачи

```python
class Berserk(AbstractPositive):
    def get_stats(self):
        stats = self.get_stats()
        stats["HP"] += 50
        stats["Strength"] += 7
        stats["Endurance"] += 7
        stats["Agility"] += 7
        stats["Luck"] += 7
        stats["Perception"] -= 3
        stats["Charisma"] -= 3
        stats["Intelligence"] -= 3
        return stats

    def get_positive_effects(self):
        return self.get_positive_effects() + ["Berserk"]
```

Берсерк (Berserk) -
Увеличивает характеристики: Сила, Выносливость, Ловкость, Удача на 7;
уменьшает характеристики: Восприятие, Харизма, Интеллект на 3;
количество единиц здоровья увеличивается на 50.

```python
class Blessing(AbstractPositive):
    def get_stats(self):
        stats = self.get_stats()
        stats["Strength"] += 2
        stats["Endurance"] += 2
        stats["Agility"] += 2
        stats["Luck"] += 2
        stats["Perception"] += 2
        stats["Charisma"] += 2
        stats["Intelligence"] += 2
        return stats

    def get_positive_effects(self):
        return self.get_positive_effects() + ["Blessing"]
```

Благословение (Blessing) - увеличивает все основные характеристики на 2.

Код задачи

```python
class Weakness(AbstractNegative):
    def get_stats(self):
        stats = self.get_stats()
        stats["Strength"] -= 4
        stats["Endurance"] -= 4
        stats["Agility"] -= 4
        return stats

    def get_negative_effects(self):
        return self.get_negative_effects() + ["Weakness"]
```

Слабость (Weakness) - уменьшает характеристики : Сила, Выносливость, Ловкость на 4.

```python
class EvilEye(AbstractNegative):
    def get_stats(self):
        stats = self.get_stats()
        stats["Luck"] -= 10
        return stats

    def get_negative_effects(self):
        return self.get_negative_effects() + ["EvilEye"]
```

Сглаз (EvilEye) - уменьшает характеристику Удача на 10.

```python
class Curse(AbstractNegative):
    def get_stats(self):
        stats = self.get_stats()
        stats["Strength"] -= 2
        stats["Endurance"] -= 2
        stats["Agility"] -= 2
        stats["Luck"] -= 2
        stats["Perception"] -= 2
        stats["Charisma"] -= 2
        stats["Intelligence"] -= 2
        return stats

    def get_negative_effects(self):
        return self.get_negative_effects() + ["Curse"]
```

Проклятье (Curse) - уменьшает все основные характеристики на 2.

Паттерн Адаптер

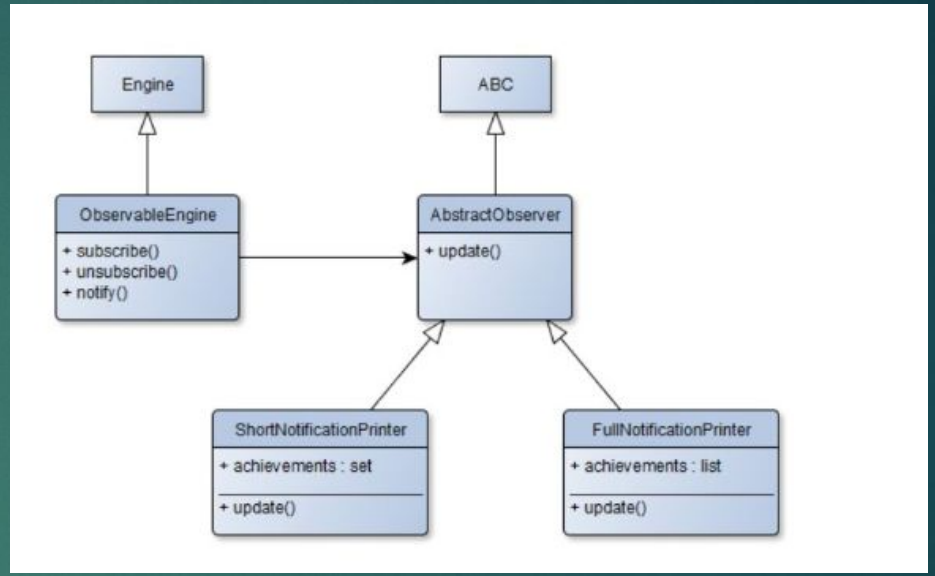
```python
class MappingAdapter:
    def __init__(self, adaptee):
        self.adaptee = adaptee

    def _get_objects_by_grid(self, descriptor, grid):
        result = []
        for i in range(dim[0]):
            for j in range(dim[1]):
                if grid[i][j] == descriptor:
                    result.append((j, i))
        return result

    def lighten(self, grid):
        dim = (len(grid[0]), len(grid))
        self.adaptee.set_dim(dim)
        lights = self._get_objects_by_grid(1, grid)
        obstacles = self._get_objects_by_grid(-1, grid)
        self.adaptee.set_lights(lights)
        self.adaptee.set_obstacles(obstacles)
        return self.adaptee.grid
```

```python
class Light:
    def __init__(self, dim):
        self.dim = dim
        self.grid = [[0 for i in range(dim[0])] for _ in range(dim[1])]
        self.lights = []
        self.obstacles = []

    def set_dim(self, dim):
        self.dim = dim
        self.grid = [[0 for i in range(dim[0])] for _ in range(dim[1])]

    def set_lights(self, lights):
        self.lights = lights
        self.generate_lights()

    def set_obstacles(self, obstacles):
        self.obstacles = obstacles
        self.generate_lights()

    def generate_lights(self):
        return self.grid.copy()


class System:
    def __init__(self):
        self.map = self.grid = [[0 for i in range(30)] for _ in range(20)]
        self.map[5][7] = 1
        self.map[5][2] = -1

    def get_lightening(self, light_mapper):
        self.lightmap = light_mapper.lighten(self.map)
```

Написать обертку над движком, которая будет иметь возможность подписывать наблюдателей и рассылать им уведомления.

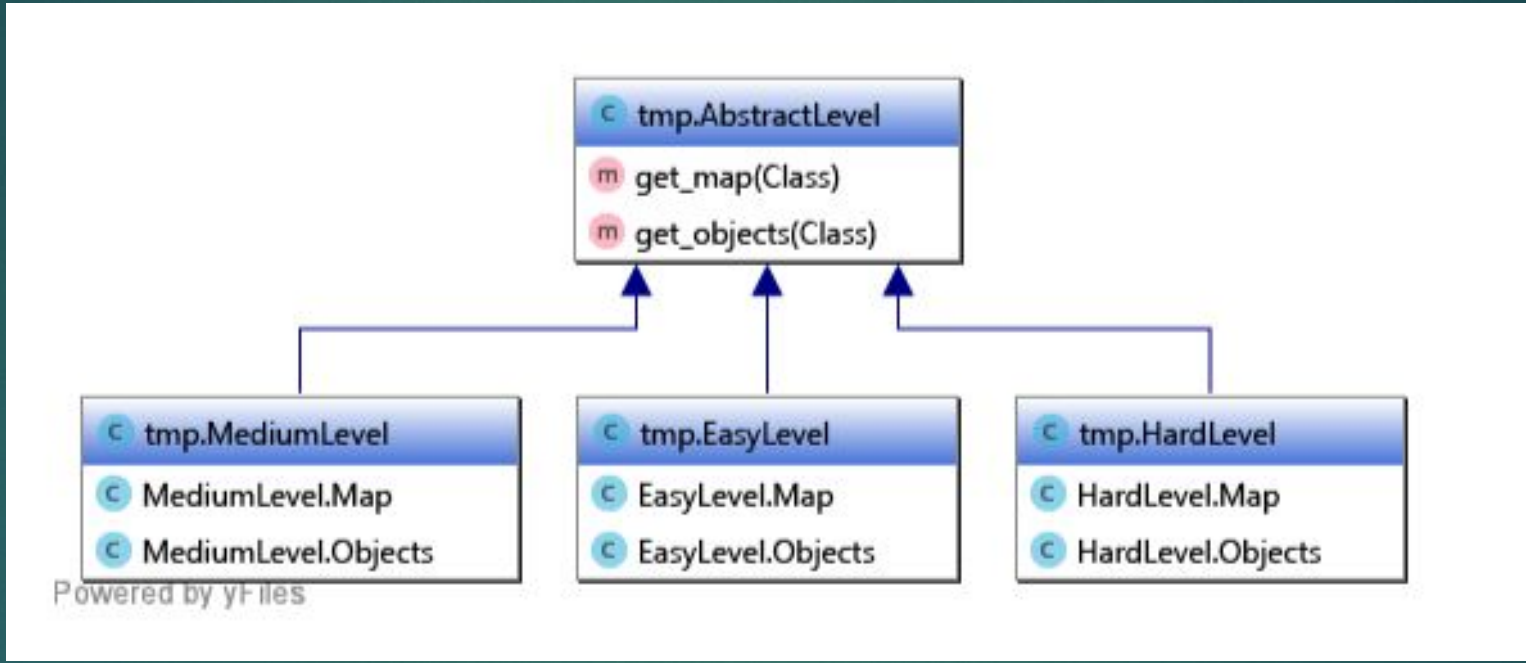
Паттерн Наблюдатель

```python
from abc import ABC, abstractmethod

class ObservableEngine:
    def __init__(self):
        self.subscribers = set()

    def subscribe(self, subscriber):
        self.subscribers.add(subscriber)

    def unsubscribe(self, subscriber):
        if subscriber (self.subscribers):
            self.subscribers.remove(subscriber)

    def notify(self, message):
       if subscriber in self.subscribers:
            subscriber.update(message)


class AbstractObserver(ABC):
    @abstractmethod
    def update(self, message):
        pass


class ShortNotificationPrinter(AbstractObserver):
    def __init__(self):
        self.achievements = set()

    def update(self, message):
        self.achievements.add(message["title"])


class FullNotificationPrinter(AbstractObserver):
    def __init__(self):
        self.achievements = list()

    def update(self, message):
        if message not in self.achievements:
            self.achievements.append(message)
```



Написать обертку над движком, которая будет иметь возможность подписывать наблюдателей и рассылать им уведомления.
Написать реализацию классов иерархии наблюдателей.
Иерархия классов приведена на следующей UML диаграмме:

Абстрактная фабрика



Код задачи

```python
import random
from abc import ABC, abstractmethod


class AbstractLevel(ABC):
    @abstractmethod
    def get_map(self):
        pass

    @abstractmethod
    def get_objects(self):
        pass
```

Необходимо создать абстрактную фабрику AbstractLevel с классовыми методами get_map() и get_objects()

```python
class EasyLevel(AbstractLevel):
    class Map:
        def __init__(self):
            self.map = [[0 for j in range(5)] for i in range(5)]
            for i in range(5):
                for j in range(5):
                    if i == 0 or j == 0 or i == 4 or j == 4:
                        self.map[j][i] = -1
                    else:

                        self.map[j][i] = random.randint(0, 2)

        def get_map(self):
            return self.map

    class Objects:
        def __init__(self):

            self.objects = [('next_lvl', (2, 2))]

        def get_objects(self, map):

            for obj_name in ['rat']:
                coord = (random.randint(1, 3), random.randint(1, 3))

                intersect = True
                while intersect:
                    intersect = False
                    for obj in self.objects:
                        if coord == obj[1]:
                            intersect = True
                            coord = (random.randint(1, 3), random.randint(1, 3))
                self.objects.append((obj_name, coord))
            return self.objects
```

Код задачи

```python
class MediumLevel(AbstractLevel):
    class Map:
        def __init__(self):
            self.map = [[0 for j in range(8)] for i in range(8)]
            for i in range(8):
                for j in range(8):
                    if i == 0 or j == 0 or i == 7 or j == 7:
                        self.map[j][i] = -1
                    else:

                        self.map[j][i] = random.randint(0, 2)

        def get_map(self):
            return self.map

    class Objects:
        def __init__(self):

            self.objects = [('next_lvl', (4, 4))]

        def get_objects(self, map):

            for obj_name in ['rat', 'snake']:
                coord = (random.randint(1, 6), random.randint(1, 6))

                intersect = True
                while intersect:
                    intersect = False
                    for obj in self.objects:
                        if coord == obj[1]:
                            intersect = True
                            coord = (random.randint(1, 6), random.randint(1, 6))
                self.objects.append((obj_name, coord))
            return self.objects
```

```python
class HardLevel(AbstractLevel):
    class Map:
        def __init__(self):
            self.map = [[0 for j in range(10)] for i in range(10)]
            for i in range(10):
                for j in range(10):
                    if i == 0 or j == 0 or i == 9 or j == 9:

                        self.map[j][i] = -1
                    else:

                        self.map[j][i] = random.randint(-1, 8)
        def get_map(self):
            return self.map


    class Objects:
        def __init__(self):

            self.objects = [('next_lvl', (5, 5))]

        def get_objects(self, map):

            for obj_name in ['rat', 'snake']:
                coord = (random.randint(1, 8), random.randint(1, 8))

                intersect = True
                while intersect:
                    intersect = False
                    if map[coord[0]][coord[1]] == -1:
                        intersect = True
                        coord = (random.randint(1, 8), random.randint(1, 8))
                        continue
                    for obj in self.objects:
                        if coord == obj[1]:
                            intersect = True
                            coord = (random.randint(1, 8), random.randint(1, 8))
                self.objects.append((obj_name, coord))
            return self.objects
```

ЗАКЛЮЧЕНИЕ

При прохождении учебной практики освоены следующие практические навыки и знания:
теоретические и практические знания ООП, и паттернов.
Создание базовых классов, паттерн адаптер и наблюдатель, декоратор класса, и абстрактной фабрики

Спасибо за внимание