«Министерство образования и науки Российской Федерации Федеральное государственное бюджетное образовательное

учреждение высшего образования «Алтайский государственный технический университет им. И.И. Ползунова»

Факультет информационных технологий Кафедра прикладной математики

Отчёт защищён с оценкой Преподаватель Савченко В.В.

« » 2022 г.

Отчёт по лабораторной работе №9

**Тема: Порождающие паттерны 1 - проектирование**

**(Singleton, Object Pool)**

Студент группы ПИ 92 В.М. Шульпов

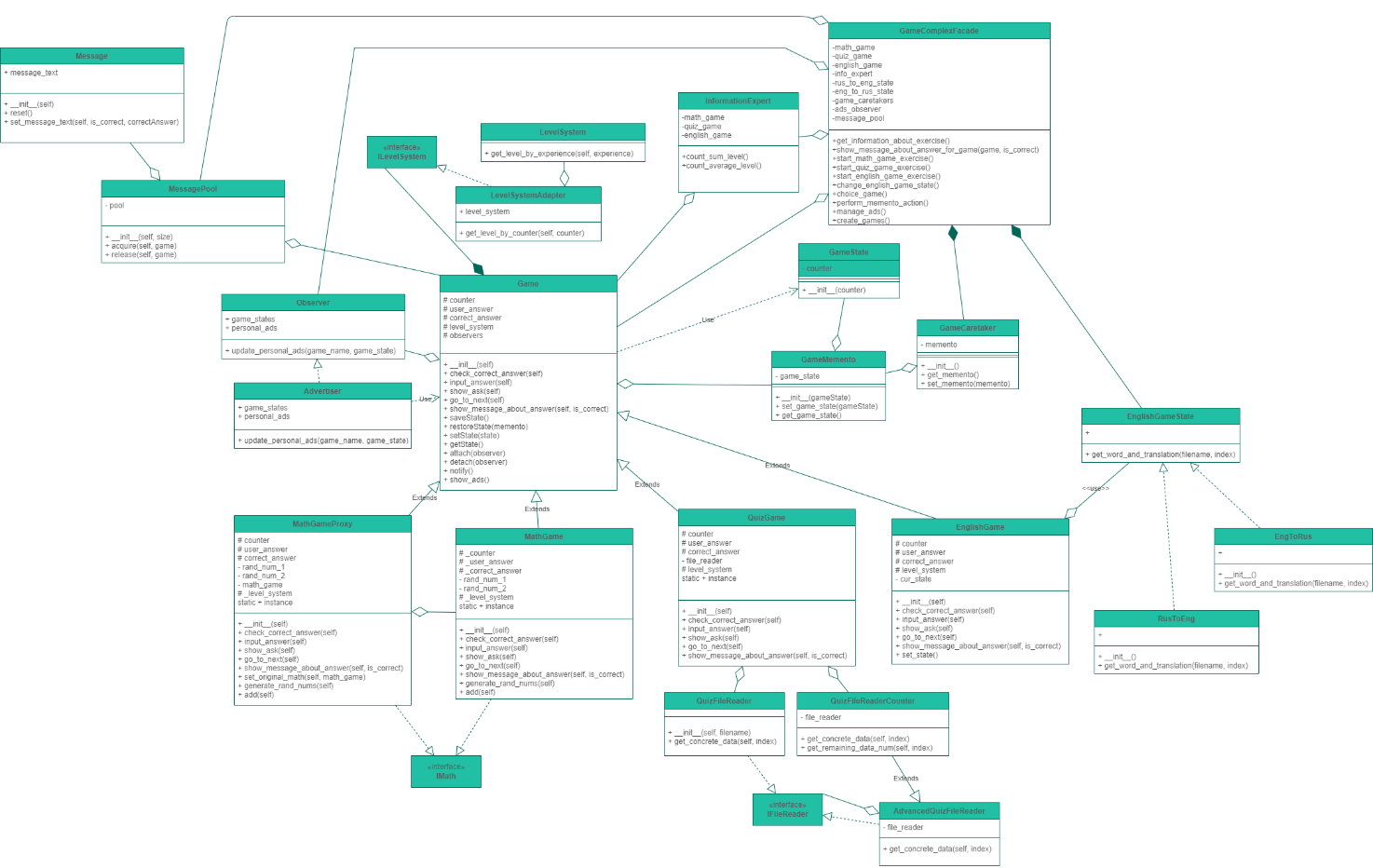
Преподаватель Савченко В.В.

Барнаул 2022

Задание

|  |  |  |  |
| --- | --- | --- | --- |
| № | ФИО студента  группы ПИ-92 | Прикладная область | Задание |
| 219 | Шульпов Виктор Максимович | Игровые системы | Проектирование обучающего игрового комплекса для  младших школьников |

**Диаграмма классов**

****

Краткое описание архитектуры паттерна ObjectPool (Пул объектов) в программе

Проблема:

Создание объекта требует больших затрат или может быть создано только

ограниченное количество объектов некоторого класса.

Решение:

Желательно, чтобы все многократно используемые объекты, свободные в некоторый

момент времени, хранились в одном и том же пуле объектов. Тогда ими можно управлять на

основе единой политики. Для этого класс Object Pool проектируется с помощью паттерна

Singleton. Если у нас много однотипных объектов, создание и уничтожение которых занимает

много ресурсов, удобно использовать этот паттерн.

Пулы объектов (известны также как пулы ресурсов) используются для управления

кэшированием объектов. Клиент, имеющий доступ к пулу объектов, может избежать создания

новых объектов, просто запрашивая в пуле уже созданный экземпляр. Пул объектов может

быть растущим, когда при отсутствии свободных создаются новые объекты или c

ограничением количества создаваемых объектов. Уничтожение объекта заменяется на его

возврат в пул для дальнейшего использования.

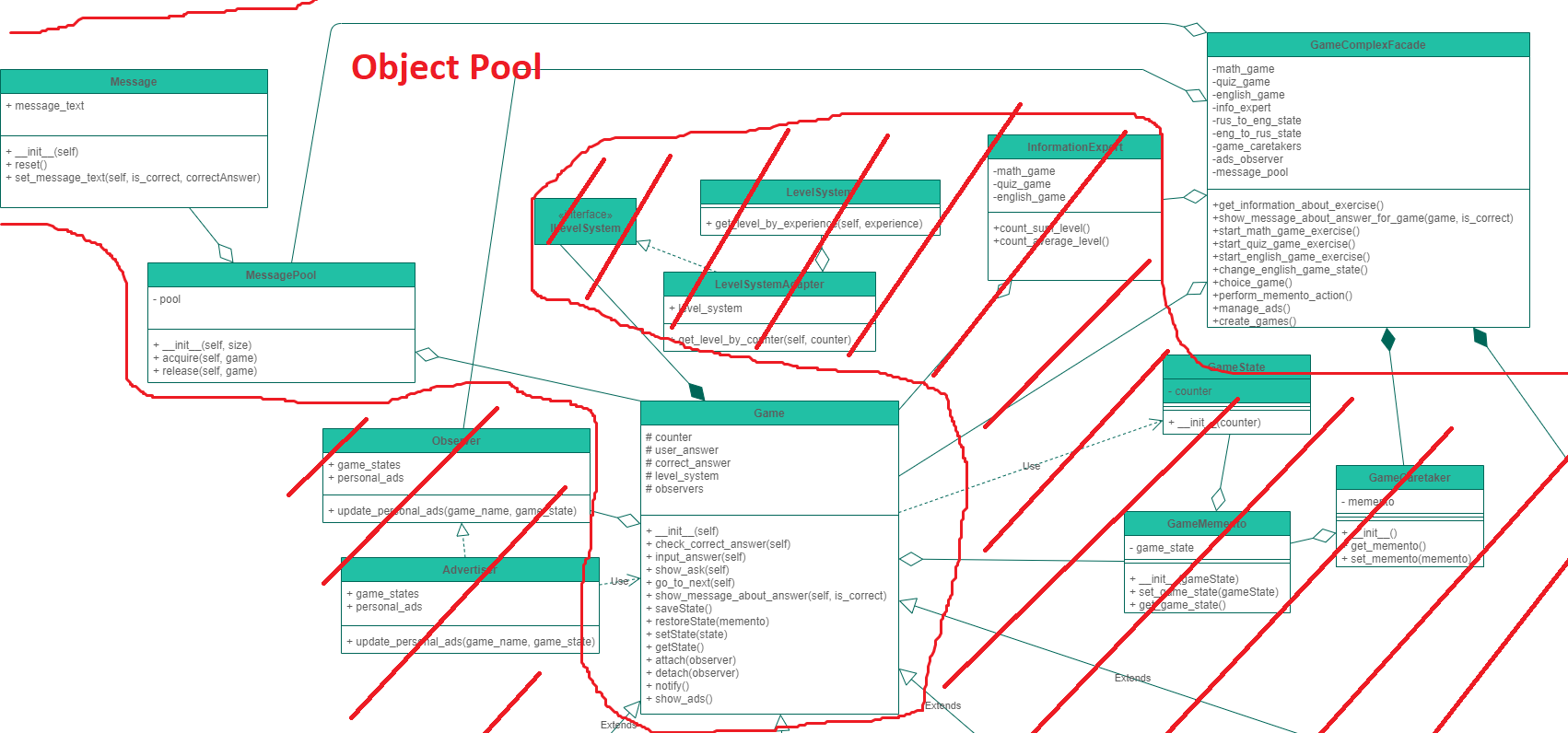
Важный момент: Объекты в пуле, которые будут использоваться многократно,

должны иметь метод для возврата в своё начальное состояние, чтобы при повторном

использовании объекта не было таких данных, которые остались со значениями от прошлого

использования.

Диаграмма классов:



* Пул объектов агрегирован в фасад
* У объектов, хранимых в пуле, есть метод reset(), который при возврате их в пул сбрасывает состояние
* Состояние объектов пула – шаблонный текст, в котором
* Данный паттерн позволяет не создавать объекты, которые используются часто. В данном примере состояние объектов пула простое, но если его функциональность увеличить (добавить, кроме текста картинки и тд), то паттерн поможет оптимизировать программу за счет того, что не придется каждый раз загружать той или иной ресурс. Мы будем просто хранить часто используемые данные в оперативной памяти/кэше или другом месте, куда доступ быстрее, нежели к БД на сервере.

Код:

**message.py**

INIT\_MESSAGE = '<<init message>>'  
  
class Message:  
 *"""шаблон сообщения"""* def \_\_init\_\_(self):  
 self.message\_text = INIT\_MESSAGE  
  
 def reset(self):  
 *"""Сбросить текст сообщения"""* self.message\_text = INIT\_MESSAGE  
  
 def set\_message\_text(self, is\_correct, correct\_answer=''):  
 *"""установка параметров сообщения"""* if is\_correct:  
 self.message\_text = f'Вы ответили правильно! (ответ: {correct\_answer})'  
 else:  
 self.message\_text = 'Вы ответили неверно!'

**message\_pool.py**  
  
class MessagePool:  
 *"""пул шаблонов сообщений"""* def \_\_init\_\_(self, \*games):  
 self.pool = {}  
 for game in games:  
 self.pool.setdefault(game, Message())  
  
 def acquire(self, game):  
 *"""забрать инициализированный объект сообщения"""* try:  
 return self.pool.pop(game)  
 except:  
 raise Exception('Не удалось получить шаблон сообщения из словаря')  
  
 def release(self, game, message):  
 *"""вернуть шаблон сообщения в пул"""* message.reset()  
 self.pool.setdefault(game, message)

**game\_complex\_facade.py**

class GameComplexFacade:  
 *"""Класс фасад для реализации простого интерфейса работы с играми"""* def \_\_init\_\_(self, info\_expert, math\_game\_proxy, quiz\_game, english\_game):  
 self.info\_expert = info\_expert  
 self.math\_game\_proxy = math\_game\_proxy  
 self.quiz\_game = quiz\_game  
 self.english\_game = english\_game  
 *# State* self.eng\_to\_rus\_state = EngToRus()  
 self.rus\_to\_eng\_state = RusToEng()  
 self.english\_game.set\_state(self.eng\_to\_rus\_state) *# установить состояние для игры EnglishGame  
 # Memento* self.game\_caretakers = { *# игры и их хранители снимков* self.math\_game\_proxy: GameCaretaker(),  
 self.quiz\_game: GameCaretaker(),  
 self.english\_game: GameCaretaker()  
 }  
 *# Observer* self.ads\_observer = Advertiser()  
 self.math\_game\_proxy.attach(self.ads\_observer) *# добавляем наблюдателей* self.quiz\_game.attach(self.ads\_observer)  
 self.english\_game.attach(self.ads\_observer)  
 *# Object pool* self.message\_pool = MessagePool(math\_game\_proxy, quiz\_game, english\_game)  
  
 def show\_information\_about\_level(self):  
 *"""получить информации, связанную с опытом"""* print(f'Суммарный уровень для всех игр {self.info\_expert.count\_sum\_level()}')  
 print(f'Средний уровень для всех игр {self.info\_expert.count\_average\_level()}')  
  
 def show\_message\_about\_answer\_for\_game(self, game, is\_correct):  
 *"""показать сообщение о результате игры, используя пул объектов сообщений"""* message = self.message\_pool.acquire(game)  
 message.set\_message\_text(is\_correct, game.correct\_answer)  
 game.show\_message\_about\_answer(message)  
 self.message\_pool.release(game, message)  
  
 def start\_math\_game\_exercise(self):  
 *"""запустить упражнение для математической игры"""* self.math\_game\_proxy.generate\_rand\_nums()  
 self.math\_game\_proxy.add()  
 self.math\_game\_proxy.show\_ask()  
 self.math\_game\_proxy.input\_answer()  
 is\_correct = self.math\_game\_proxy.check\_correct\_answer()  
 self.show\_message\_about\_answer\_for\_game(self.math\_game\_proxy, is\_correct)  
 if is\_correct:  
 self.math\_game\_proxy.go\_to\_next()  
 self.math\_game\_proxy.show\_ads()  
  
 def start\_quiz\_game\_exercise(self):  
 *"""запустить упражнение для игры-викторины"""* if self.quiz\_game.show\_ask():  
 self.quiz\_game.input\_answer()  
 is\_correct = self.quiz\_game.check\_correct\_answer()  
 self.show\_message\_about\_answer\_for\_game(self.quiz\_game, is\_correct)  
 if is\_correct:  
 self.quiz\_game.go\_to\_next()  
 self.quiz\_game.show\_ads()  
  
 def start\_english\_game\_exercise(self):  
 *"""запустить упражнение для игры-обучалки английского языка"""* if self.english\_game.show\_ask():  
 self.english\_game.input\_answer()  
 is\_correct = self.english\_game.check\_correct\_answer()  
 self.show\_message\_about\_answer\_for\_game(self.english\_game, is\_correct)  
 if is\_correct:  
 self.english\_game.go\_to\_next()  
 self.english\_game.show\_ads()  
  
 def change\_english\_game\_state(self):  
 *"""изменить состояние игры EnglishGame"""* while True:  
 choice = input(f'Текущий режим игры - {self.english\_game.cur\_state}'  
 f'\n\t1 - изменить режим на {self.rus\_to\_eng\_state}'  
 f'\n\t2 - изменить режим на {self.eng\_to\_rus\_state}\n'  
 f'\n\t0 - назад\n\t\t:\t')  
 if choice == '1':  
 self.english\_game.cur\_state = self.rus\_to\_eng\_state  
 return  
 if choice == '2':  
 self.english\_game.cur\_state = self.eng\_to\_rus\_state  
 return  
 if choice == '0':  
 return  
  
 def choice\_game(self):  
 *"""выбрать игру"""* while True:  
 choice = input(f'Выберите игру:'  
 f'\n\t1 - {self.math\_game\_proxy}'  
 f'\n\t2 - {self.quiz\_game}'  
 f'\n\t3 - {self.english\_game}\n')  
 if choice == '1':  
 return self.math\_game\_proxy  
 if choice == '2':  
 return self.quiz\_game  
 if choice == '3':  
 return self.english\_game  
  
 def perform\_memento\_action(self):  
 *"""выполнить действие, связанное со снимком"""* game = self.choice\_game()  
 while True:  
 choice = input(f'Выберите действия, связанные с прогрессом:'  
 f'\n\t1 - сохранить прогресс'  
 f'\n\t2 - восстановить сохраненное состояние прогресса\n'  
 f'\n\t0 - назад\n\t\t:\t')  
 if choice == '0':  
 return  
 if choice == '1':  
 self.game\_caretakers[game].set\_memento(game.save\_state())  
 return  
 if choice == '2':  
 memento = self.game\_caretakers[game].get\_memento()  
 if memento is None:  
 print('Нет бэкапов!')  
 else:  
 game.restore\_state(memento)  
 return  
  
 def manage\_ads(self):  
 *"""управлять рекламой"""* while True:  
 choice = input(f'Отключить рекламу?'  
 f'\n\t1 - да (выбрать игру)'  
 f'\n\t2 - да (для всех игр)'  
 f'\n\t0 - нет\n\t\t:\t')  
 if choice == '0':  
 return  
 if choice == '1':  
 game = self.choice\_game()  
 game.detach(self.ads\_observer)  
 return  
 if choice == '2':  
 self.math\_game\_proxy.detach(self.ads\_observer)  
 self.quiz\_game.detach(self.ads\_observer)  
 self.english\_game.detach(self.ads\_observer)  
 return  
  
 def create\_games(self):  
 *"""создать игры"""* try:  
 self.math\_game\_proxy = MathGame()  
 self.english\_game = EnglishGame('filename.txt')  
 self.quiz\_game = QuizGame('filename2.txt')  
 print('Создание игр произошло успешно')  
 except:  
 print('Создать игры не удалось')

Краткое описание архитектуры паттерна Singleton (одиночка) в программе

**Проблема:**

Необходимо, чтобы у класса существовал только один экземпляр, к которому нужно

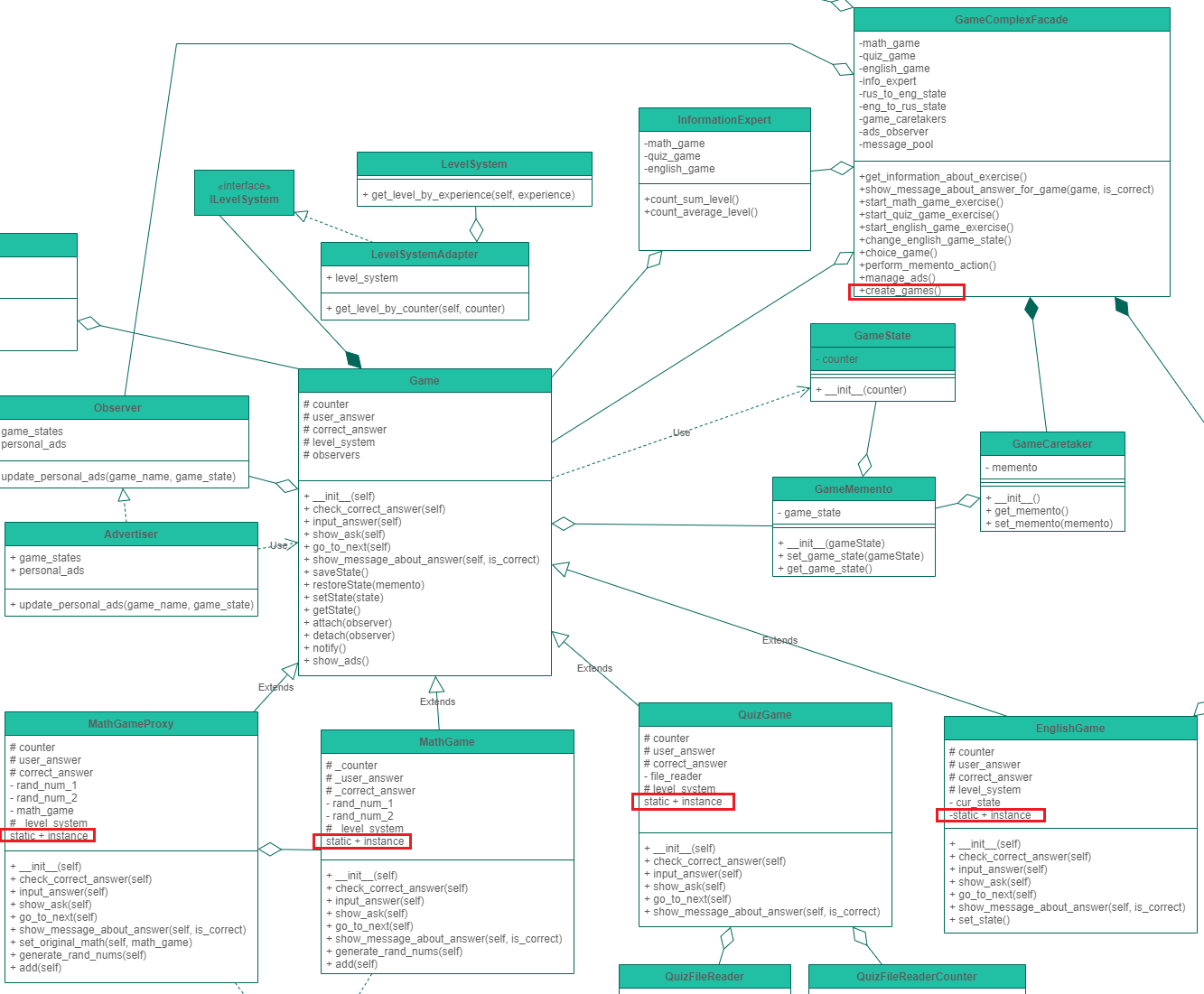
создать глобальную точку доступа.

**Решение:**

Сам класс контролирует то, что у него есть только один экземпляр, может запретить

создание дополнительных экземпляров, перехватывая запросы на создание новых объектов, и он же способен предоставить доступ к своему экземпляру.

**Диаграмма классов:**



* Внутри конструкторов классов игр присутствует проверка на то, был ли создан экземпляр.
* Объекты игр создаются вначале независимо от пользователя, а затем их можно попробовать создать снова с помощью функции create\_games(). При повторном вызове конструктора (без удаления игры) выбросится исключение, которое отлавливается в функции create\_games() и сообщает о том, что объекты уже созданы.
* Было бы лучше, если бы при повторной попытке создать объект из конструктора возвращался instance уже созданного, однако в данном ЯП return в конструкторе не работает.

Код:

math\_game.py

class MathGame(Game, IMath):  
 *"""игра - суммирование чисел (математика)"""* instance = None #static  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 if MathGame.instance is not None:  
 raise Exception("Объект класса Game уже создан!")  
 self.rand\_num\_1 = None  
 self.rand\_num\_2 = None  
 MathGame.instance = self  
  
 def \_\_str\_\_(self):  
 return "Игра <<Математика>>"  
  
 def show\_ask(self):  
 *"""показ вопроса"""* print(f'Вопрос №{self.counter + 1}\n\tЧему равно {self.rand\_num\_1} + {self.rand\_num\_2}?')  
  
 def input\_answer(self):  
 *"""ввод пользовательского ответа"""* num = input('Введите сумму чисел:\t')  
 self.user\_answer = num  
  
 def generate\_rand\_nums(self):  
 *"""генерация двух случайных чисел для суммирования"""* self.rand\_num\_1 = randint(MIN\_NUM, MAX\_NUM)  
 self.rand\_num\_2 = randint(MIN\_NUM, MAX\_NUM)  
  
 def add(self):  
 *"""суммирование 2 чисел"""* self.correct\_answer = str(self.rand\_num\_1 + self.rand\_num\_2)  
 return self.correct\_answer

**Аналогично в классах других игр.**

**Функция создания игр в фасаде:**

def create\_games(self):  
 *"""создать игры"""* try:  
 self.math\_game\_proxy = MathGame()  
 self.english\_game = EnglishGame('filename.txt')  
 self.quiz\_game = QuizGame('filename2.txt')  
 print('Создание игр произошло успешно')  
 except:  
 print('Создать игры не удалось')

Код программы:

game.py

from level\_system\_adapter import LevelSystemAdapter  
from level\_system import LevelSystem  
from game\_memento import GameMemento  
from game\_state import GameState  
  
  
class Game:  
 *"""игра"""* def \_\_init\_\_(self):  
 self.counter = 0  
 self.user\_answer = None  
 self.correct\_answer = None  
 self.level\_system = LevelSystemAdapter(LevelSystem())  
 self.observers = []  
  
 def check\_correct\_answer(self):  
 *"""проверка правильности ответа пользователя"""* if self.user\_answer == self.correct\_answer:  
 return True  
 return False  
  
 def input\_answer(self):  
 *"""ввод пользовательского ответа"""* pass  
  
 def show\_ask(self):  
 *"""показ вопроса"""* pass  
  
 def go\_to\_next(self):  
 *"""увеличение счетчика (переход на следующий уровень/вопрос)"""* self.counter += 1  
 print(f'Переход на следующее задание!\nВаш текущий уровень {self.get\_level()}!')  
 self.notify() *# передать новую информацию наблюдателям* def show\_message\_about\_answer(self, message):  
 *"""сообщение о правильности ответа"""* print(message.message\_text)  
  
 def get\_level(self):  
 *"""получить текущий уровень"""* return self.level\_system.get\_level\_by\_counter(self.counter)  
  
 def save\_state(self):  
 *"""создать и вернуть снимок состояния объекта Game"""* return GameMemento(GameState(self.counter))  
  
 def restore\_state(self, game\_memento):  
 *"""восстановить состояние Game из снимка"""* print('Прогресс восстановлен!')  
 self.counter = game\_memento.get\_game\_state().counter  
  
 def setState(self, state):  
 *"""установить состояние Game"""* self.counter = state.counter  
  
 def getState(self):  
 *"""получить состояние Game"""* return GameState(self.counter)  
  
 def attach(self, observer):  
 *"""добавить (закрепить) наблюдателя"""* self.observers.append(observer)  
  
 def detach(self, observer):  
 *"""удалить (открепить) наблюдателя"""* if observer in self.observers:  
 self.observers.remove(observer)  
  
 def notify(self):  
 *"""уведомить наблюдателей"""* for obs in self.observers:  
 obs.update\_personal\_ads(str(self), self.counter)  
  
 def show\_ads(self):  
 *"""показать рекламу (если есть рекламщик)"""* for obs in self.observers:  
 print(f'РЕКЛАМА: {obs.personal\_ads}')

**ifile\_reader.py:**

from abc import ABC, abstractmethod  
  
  
class IFileReader(ABC):  
 *"""читатель файла"""* @abstractmethod  
 def get\_concrete\_data(self, index):  
 *"""получение данных для уровня"""* pass

**ilevel\_system.py:**

from abc import ABC, abstractmethod  
  
class ILevelSystem(ABC):  
 *"""интерфейс уровневой системы"""* @abstractmethod  
 def get\_level\_by\_counter(self, counter):  
 *"""посчитать текущий уровень по количеству пройденных уровней"""* pass

**level\_system.py:**

class LevelSystem():  
 *"""уровневая система"""* def \_\_init\_\_(self):  
 pass  
  
 def get\_level\_by\_experience(self, experience):  
 *"""посчитать текущий уровень по количеству опыта  
 100 опыта = 1 уровень"""* return experience // 100

**level\_system\_adapter.py**

from ilevel\_system import ILevelSystem  
  
  
class LevelSystemAdapter(ILevelSystem):  
 *"""адаптер уровневой системы"""* def \_\_init\_\_(self, level\_system):  
 self.level\_system = level\_system  
  
 def get\_level\_by\_counter(self, counter):  
 *"""посчитать текущий уровень по количеству пройденных уровней  
 1 пройденное задание = 50 опыта  
 100 опыта = 1 уровень"""* experience = 50 \* counter  
 return self.level\_system.get\_level\_by\_experience(experience)

imath.py:

from abc import ABC, abstractmethod  
  
  
class IMath(ABC):  
 *"""интерфейс игры математика (сложение чисел)"""* @abstractmethod  
 def add(self):  
 *"""суммирование 2 чисел"""* pass

math\_game.py:

from game import Game  
from random import randint  
from imath import IMath  
  
  
MIN\_NUM = 1  
MAX\_NUM = 20  
  
  
class MathGame(Game, IMath):  
 *"""игра - суммирование чисел (математика)"""* instance = None  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 if MathGame.instance is not None:  
 raise Exception("Объект класса Game уже создан!")  
 self.rand\_num\_1 = None  
 self.rand\_num\_2 = None  
 MathGame.instance = self  
  
 def \_\_str\_\_(self):  
 return "Игра <<Математика>>"  
  
 def show\_ask(self):  
 *"""показ вопроса"""* print(f'Вопрос №{self.counter + 1}\n\tЧему равно {self.rand\_num\_1} + {self.rand\_num\_2}?')  
  
 def input\_answer(self):  
 *"""ввод пользовательского ответа"""* num = input('Введите сумму чисел:\t')  
 self.user\_answer = num  
  
 def generate\_rand\_nums(self):  
 *"""генерация двух случайных чисел для суммирования"""* self.rand\_num\_1 = randint(MIN\_NUM, MAX\_NUM)  
 self.rand\_num\_2 = randint(MIN\_NUM, MAX\_NUM)  
  
 def add(self):  
 *"""суммирование 2 чисел"""* self.correct\_answer = str(self.rand\_num\_1 + self.rand\_num\_2)  
 return self.correct\_answer

math\_game\_proxy.py:

from game import Game  
from imath import IMath  
from datetime import datetime  
  
  
MATH\_LOG\_FILE = 'math\_log.txt'  
  
  
MIN\_NUM = 1  
MAX\_NUM = 20  
  
  
def log\_add(filename):  
 *"""логирование (хранение истории обращений)"""* with open(filename, 'a') as file:  
 cur\_date = datetime.now()  
 adding\_log = f'addding; время: {cur\_date}\n'  
 file.write(adding\_log)  
  
  
class MathGameProxy(Game, IMath):  
 *"""логирующий заместитель игры - суммирование чисел (математика)"""  
 #TODO свойства MathGameProxy не ссылаются на свойства MathGame ->  
 # проблемы с получением свойств MathGame ->  
 # решение: сделать свойства через изменяемый объект, например, через словарь  
 # !или использовать геттеры и сеттеры для всех свойств* def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.math\_game = None  
 self.counter = None  
  
 def \_\_str\_\_(self):  
 return "Игра <<Математика>> (прокси)"  
  
 def set\_original\_math(self, math\_game):  
 *"""ленивая инициализация реального объекта"""* if self.math\_game is None:  
 self.math\_game = math\_game  
  
 def generate\_rand\_nums(self):  
 *"""генерация двух случайных чисел для суммирования"""* self.math\_game.generate\_rand\_nums()  
  
 def add(self):  
 *"""суммирование 2 чисел"""* self.math\_game.add()  
 log\_add(MATH\_LOG\_FILE)  
  
 def show\_ask(self):  
 *"""показать вопрос"""* self.math\_game.show\_ask()  
  
 def input\_answer(self):  
 *"""ввод пользовательского ответа"""* self.math\_game.input\_answer()  
  
 def check\_correct\_answer(self):  
 *"""проверка правильности ответа пользователя"""* return self.math\_game.check\_correct\_answer()  
  
 def show\_message\_about\_answer(self, is\_correct):  
 *"""сообщение о правильности ответа"""* self.math\_game.show\_message\_about\_answer(is\_correct)  
  
 def go\_to\_next(self):  
 *"""увеличение счетчика (переход на следующий уровень/вопрос)"""* self.math\_game.go\_to\_next()  
  
 def get\_level(self):  
 *"""получить текущий уровень"""* return self.math\_game.level\_system.get\_level\_by\_counter(self.math\_game.counter)

**advanced\_quiz\_file\_reader.py:**

from ifile\_reader import IFileReader  
import csv  
  
  
class AdvancedQuizFileReader(IFileReader):  
 *"""читатель-декоратор файла для игры викторина  
 (читает вопрос, варианты ответа, и номер правильного ответа)  
 дополняет функционал за счет способности подсчитывать количество оставшихся в файле уровней"""* def \_\_init\_\_(self, file\_reader):  
 pass  
  
 def get\_concrete\_data(self, index):  
 *"""получать вопрос, номер правильного ответа и варианты ответа"""* pass  
  
 def get\_filename(self):  
 *"""getter filename"""* pass  
  
 def \_\_get\_remaining\_data\_num(self, index):  
 *"""подсчитывает количество оставшихся вопросов в файле"""* pass

**quiz\_file\_reader.py:**

from ifile\_reader import IFileReader  
import csv  
  
  
class QuizFileReader(IFileReader):  
 *"""читатель файла для игры викторина (читает вопрос, варианты ответа, и номер правильного ответа"""* def \_\_init\_\_(self, filename):  
 self.filename = filename  
  
 def get\_filename(self):  
 *"""getter filename"""* return self.filename  
  
 def get\_concrete\_data(self, index):  
 *"""получать вопрос, номер правильного ответа и варианты ответа"""* with open(self.filename, 'r', newline='') as File:  
 reader = csv.reader(File, delimiter=',')  
 counter = 0  
 for ask\_i, row in enumerate(reader):  
 ask = row[0]  
 correct\_answer\_num = next(reader)[0]  
 answers = next(reader)  
 counter += 1  
 if ask\_i == index:  
 return [ask, correct\_answer\_num, answers]  
 return None

quiz\_file\_reader\_counter.py

from advanced\_quiz\_file\_reader import AdvancedQuizFileReader  
import csv  
  
  
class QuizFileReaderCounter(AdvancedQuizFileReader):  
 *"""читатель-декоратор файла для игры викторина  
 (читает вопрос, варианты ответа, и номер правильного ответа)  
 дополняет функционал за счет способности подсчитывать количество оставшихся в файле уровней"""* def \_\_init\_\_(self, file\_reader):  
 self.file\_reader = file\_reader  
  
 def get\_concrete\_data(self, index):  
 *"""получать вопрос, номер правильного ответа и варианты ответа"""* counter = self.\_\_get\_remaining\_data\_num(index)  
 print(f'Осталось вопросов: {counter}')  
 return self.file\_reader.get\_concrete\_data(index)  
  
 def get\_filename(self):  
 *"""getter filename"""* return self.file\_reader.get\_filename()  
  
 def \_\_get\_remaining\_data\_num(self, index):  
 *"""подсчитывает количество оставшихся вопросов в файле"""* with open(self.file\_reader.get\_filename(), 'r', newline='') as File:  
 reader = csv.reader(File, delimiter=',')  
 counter = 0  
 for ask\_i, row in enumerate(reader):  
 next(reader)  
 next(reader)  
 if ask\_i < index:  
 continue  
 counter += 1  
 return counter

english\_game.py

from game import Game  
*# import csv*ENGLISH\_WORDS\_FILE = 'english\_words.csv'  
  
  
class EnglishGame(Game):  
 *"""игра - обучалка слов по английскому языку"""* instance = None  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 if EnglishGame.instance is not None:  
 raise Exception("Объект класса Game уже создан!")  
 self.filename = ENGLISH\_WORDS\_FILE  
 self.cur\_state = None  
 EnglishGame.instance = self  
  
 def \_\_init\_\_(self, filename):  
 super().\_\_init\_\_()  
 if EnglishGame.instance is not None:  
 raise Exception("Объект класса Game уже создан!")  
 self.filename = filename  
 self.cur\_state = None  
 EnglishGame.instance = self  
  
 def \_\_str\_\_(self):  
 return "Игра <<Английские слова>>"  
  
 def set\_state(self, state):  
 *"""установить состояние"""* self.cur\_state = state  
  
 def show\_ask(self):  
 *"""показ слова """* words = self.cur\_state.get\_word\_and\_translation(self.filename, self.counter)  
 if words is None:  
 print('Слова кончились!')  
 return False  
 word\_to\_translate = words[0]  
 self.correct\_answer = words[1]  
 print(f'Вопрос №{self.counter + 1}\n\tПереведите слово "{word\_to\_translate}"')  
 return True  
  
 def input\_answer(self):  
 *"""ввод пользовательского ответа"""* word = input('Введите перевод:\t')  
 self.user\_answer = word  
  
 def check\_correct\_answer(self):  
 *"""проверка правильности ответа и запись в список неправильных, если ответ неверный"""* if super().check\_correct\_answer():  
 return True  
 return False

quiz\_game.py:

from game import Game  
from quiz\_file\_reader import QuizFileReader  
from quiz\_file\_reader\_counter import QuizFileReaderCounter  
  
QUIZ\_ASK\_FILE = 'quiz\_asks.csv'  
  
  
class QuizGame(Game):  
 *"""игра - викторина (по русскому языку)"""* instance = None  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 if QuizGame.instance is not None:  
 raise Exception("Объект класса Game уже создан!")  
 self.file\_reader = QuizFileReaderCounter(QuizFileReader(QUIZ\_ASK\_FILE)) *# QuizFileReader(QUIZ\_ASK\_FILE)* QuizGame.instance = self  
  
 def \_\_init\_\_(self, filename):  
 super().\_\_init\_\_()  
 if QuizGame.instance is not None:  
 raise Exception("Объект класса Game уже создан!")  
 self.file\_reader = QuizFileReaderCounter(QuizFileReader(filename)) *# QuizFileReader(filename)* QuizGame.instance = self  
  
 def \_\_str\_\_(self):  
 return "Игра <<Русский язык>>"  
  
 def show\_ask(self):  
 *"""показ вопроса и вариантов овтета текущего уровня  
 concrete\_data= [ask, num\_correct\_answer, answers]"""* conrete\_data = self.file\_reader.get\_concrete\_data(self.counter)  
 if conrete\_data is None:  
 print('Вопросы кончились!')  
 return False  
 ask = conrete\_data[0]  
 self.correct\_answer = conrete\_data[1]  
 answers = conrete\_data[2]  
 print(f'Вопрос №{self.counter + 1}\n\t{ask}')  
 for i, answer in enumerate(answers):  
 print(f'\t\t{i + 1}) {answer}')  
 return True  
  
 def input\_answer(self):  
 *"""ввод пользовательского ответа"""* num = input('Введите номер правильного ответа:\t')  
 self.user\_answer = num

quiz\_asks.csv:

Какая часть слова является носителем его значения? 2

приставка,корень,суффикс,окончание

По какому принципу слова объединяются в части речи? 2

по назначению,по общности грамматических свойств,по смыслу,по историческому принципу

Укажите падеж выделенного имени существительного. Под ДЕРЕВЬЯМИ зеленеют обросшие брусничником кочки. (Соколов-Микитов И.)

4

родительный,дательный,винительный,творительный

Какой род прилагательных не выделяют в русском языке? 4

мужской,средний,женский,общий

Определите род прилагательного, которое употреблено в предложении: «Солнце скрылось за темной тучей».

3

мужской,средний,женский,общий

В каком из словосочетаний употреблено прилагательное мужского рода? 1

летний дождь,длинная дорога,новое платье,сладкая ягода

math\_log.txt:

addding; время: 2022-04-20 02:06:28.384450

addding; время: 2022-04-20 02:06:31.581248

addding; время: 2022-04-20 02:06:46.565661

addding; время: 2022-04-20 02:06:49.972286

addding; время: 2022-04-20 02:06:53.222613

addding; время: 2022-04-20 02:06:55.897448

addding; время: 2022-04-20 02:06:58.671007

addding; время: 2022-04-20 02:06:59.645661

addding; время: 2022-04-20 14:21:02.534213

game\_complex\_facade.py:

from rus\_to\_eng import RusToEng  
from eng\_to\_rus import EngToRus  
from game\_caretaker import GameCaretaker  
from advertiser import Advertiser  
from message\_pool import MessagePool  
from math\_game import MathGame  
from quiz\_game import QuizGame  
from english\_game import EnglishGame  
  
  
class GameComplexFacade:  
 *"""Класс фасад для реализации простого интерфейса работы с играми"""* def \_\_init\_\_(self, info\_expert, math\_game\_proxy, quiz\_game, english\_game):  
 self.info\_expert = info\_expert  
 self.math\_game\_proxy = math\_game\_proxy  
 self.quiz\_game = quiz\_game  
 self.english\_game = english\_game  
 *# State* self.eng\_to\_rus\_state = EngToRus()  
 self.rus\_to\_eng\_state = RusToEng()  
 self.english\_game.set\_state(self.eng\_to\_rus\_state) *# установить состояние для игры EnglishGame  
 # Memento* self.game\_caretakers = { *# игры и их хранители снимков* self.math\_game\_proxy: GameCaretaker(),  
 self.quiz\_game: GameCaretaker(),  
 self.english\_game: GameCaretaker()  
 }  
 *# Observer* self.ads\_observer = Advertiser()  
 self.math\_game\_proxy.attach(self.ads\_observer) *# добавляем наблюдателей* self.quiz\_game.attach(self.ads\_observer)  
 self.english\_game.attach(self.ads\_observer)  
 *# Object pool* self.message\_pool = MessagePool(math\_game\_proxy, quiz\_game, english\_game)  
  
 def show\_information\_about\_level(self):  
 *"""получить информации, связанную с опытом"""* print(f'Суммарный уровень для всех игр {self.info\_expert.count\_sum\_level()}')  
 print(f'Средний уровень для всех игр {self.info\_expert.count\_average\_level()}')  
  
 def show\_message\_about\_answer\_for\_game(self, game, is\_correct):  
 *"""показать сообщение о результате игры, используя пул объектов сообщений"""* message = self.message\_pool.acquire(game)  
 message.set\_message\_text(is\_correct, game.correct\_answer)  
 game.show\_message\_about\_answer(message)  
 self.message\_pool.release(game, message)  
  
 def start\_math\_game\_exercise(self):  
 *"""запустить упражнение для математической игры"""* self.math\_game\_proxy.generate\_rand\_nums()  
 self.math\_game\_proxy.add()  
 self.math\_game\_proxy.show\_ask()  
 self.math\_game\_proxy.input\_answer()  
 is\_correct = self.math\_game\_proxy.check\_correct\_answer()  
 self.show\_message\_about\_answer\_for\_game(self.math\_game\_proxy, is\_correct)  
 if is\_correct:  
 self.math\_game\_proxy.go\_to\_next()  
 self.math\_game\_proxy.show\_ads()  
  
 def start\_quiz\_game\_exercise(self):  
 *"""запустить упражнение для игры-викторины"""* if self.quiz\_game.show\_ask():  
 self.quiz\_game.input\_answer()  
 is\_correct = self.quiz\_game.check\_correct\_answer()  
 self.show\_message\_about\_answer\_for\_game(self.quiz\_game, is\_correct)  
 if is\_correct:  
 self.quiz\_game.go\_to\_next()  
 self.quiz\_game.show\_ads()  
  
 def start\_english\_game\_exercise(self):  
 *"""запустить упражнение для игры-обучалки английского языка"""* if self.english\_game.show\_ask():  
 self.english\_game.input\_answer()  
 is\_correct = self.english\_game.check\_correct\_answer()  
 self.show\_message\_about\_answer\_for\_game(self.english\_game, is\_correct)  
 if is\_correct:  
 self.english\_game.go\_to\_next()  
 self.english\_game.show\_ads()  
  
 def change\_english\_game\_state(self):  
 *"""изменить состояние игры EnglishGame"""* while True:  
 choice = input(f'Текущий режим игры - {self.english\_game.cur\_state}'  
 f'\n\t1 - изменить режим на {self.rus\_to\_eng\_state}'  
 f'\n\t2 - изменить режим на {self.eng\_to\_rus\_state}\n'  
 f'\n\t0 - назад\n\t\t:\t')  
 if choice == '1':  
 self.english\_game.cur\_state = self.rus\_to\_eng\_state  
 return  
 if choice == '2':  
 self.english\_game.cur\_state = self.eng\_to\_rus\_state  
 return  
 if choice == '0':  
 return  
  
 def choice\_game(self):  
 *"""выбрать игру"""* while True:  
 choice = input(f'Выберите игру:'  
 f'\n\t1 - {self.math\_game\_proxy}'  
 f'\n\t2 - {self.quiz\_game}'  
 f'\n\t3 - {self.english\_game}\n')  
 if choice == '1':  
 return self.math\_game\_proxy  
 if choice == '2':  
 return self.quiz\_game  
 if choice == '3':  
 return self.english\_game  
  
 def perform\_memento\_action(self):  
 *"""выполнить действие, связанное со снимком"""* game = self.choice\_game()  
 while True:  
 choice = input(f'Выберите действия, связанные с прогрессом:'  
 f'\n\t1 - сохранить прогресс'  
 f'\n\t2 - восстановить сохраненное состояние прогресса\n'  
 f'\n\t0 - назад\n\t\t:\t')  
 if choice == '0':  
 return  
 if choice == '1':  
 self.game\_caretakers[game].set\_memento(game.save\_state())  
 return  
 if choice == '2':  
 memento = self.game\_caretakers[game].get\_memento()  
 if memento is None:  
 print('Нет бэкапов!')  
 else:  
 game.restore\_state(memento)  
 return  
  
 def manage\_ads(self):  
 *"""управлять рекламой"""* while True:  
 choice = input(f'Отключить рекламу?'  
 f'\n\t1 - да (выбрать игру)'  
 f'\n\t2 - да (для всех игр)'  
 f'\n\t0 - нет\n\t\t:\t')  
 if choice == '0':  
 return  
 if choice == '1':  
 game = self.choice\_game()  
 game.detach(self.ads\_observer)  
 return  
 if choice == '2':  
 self.math\_game\_proxy.detach(self.ads\_observer)  
 self.quiz\_game.detach(self.ads\_observer)  
 self.english\_game.detach(self.ads\_observer)  
 return  
  
 def create\_games(self):  
 *"""создать игры"""* try:  
 self.math\_game\_proxy = MathGame()  
 self.english\_game = EnglishGame('filename.txt')  
 self.quiz\_game = QuizGame('filename2.txt')  
 print('Создание игр произошло успешно')  
 except:  
 print('Создать игры не удалось')

information\_expert.py:

from game import Game

class InformationExpert:  
 *"""информационный эксперт для инкапсулированных в него игр, выполняет расчеты, связанные со свойствами игр"""* def \_\_init\_\_(self, math\_game\_proxy, quiz\_game, english\_game):  
 self.math\_game\_proxy = math\_game\_proxy  
 self.quiz\_game = quiz\_game  
 self.english\_game = english\_game  
  
 def count\_sum\_level(self):  
 *"""подсчитать суммарный опыт для всех игр"""* math\_level = self.math\_game\_proxy.get\_level()  
 quiz\_level = self.quiz\_game.get\_level()  
 english\_level = self.english\_game.get\_level()  
 return math\_level + quiz\_level + english\_level  
  
 def count\_average\_level(self):  
 *"""подсчитать суммарный опыт для всех игр"""* game\_number = 3  
 math\_level = self.math\_game\_proxy.get\_level()  
 quiz\_level = self.quiz\_game.get\_level()  
 english\_level = self.english\_game.get\_level()  
 return (math\_level + quiz\_level + english\_level) / game\_number

advertiser.py

from observer import Observer  
  
class Advertiser(Observer):  
 *"""наблюдатель-рекламщик"""* def \_\_init\_\_(self):  
 self.game\_states = {}  
 self.personal\_ads = f'Общая реклама' *# сюда будет записана персонализированная реклама* def update\_personal\_ads(self, game\_name, counter):  
 *"""подобрать подходящую рекламу"""* if self.game\_states.get(game\_name):  
 self.game\_states.update({game\_name: counter})  
 else:  
 self.game\_states.setdefault(game\_name, counter)  
 self.personal\_ads = f'Реклама для тех, кто предпочитает {max(self.game\_states, key=self.game\_states.get)}'  
 print(f'\nDEBUG\n'  
 f'max = {max(self.game\_states, key=self.game\_states.get)}\n'  
 f'self.game\_states = {self.game\_states}\n')

eng\_to\_rus.py

from english\_game\_state import EnglishGameState  
import csv  
  
  
class EngToRus(EnglishGameState):  
 *"""состояние игры EnglishGame (с английского на русский)"""* def \_\_str\_\_(self):  
 *"""преобразование в строку"""* return "<<перевод с английского на русский>>"  
  
 def get\_word\_and\_translation(self, filename, index):  
 *"""получение слова и его перевода по индексу (номеру упражнения)"""* with open(filename, 'r', newline='') as File:  
 reader = csv.reader(File, delimiter=';')  
 counter = 0  
 for row\_i, row in enumerate(reader):  
 word\_to\_translate = row[0] *# слово для перевода (английское)* word\_translation = row[1] *# сам перевод слова (русское)* counter += 1  
 if row\_i == index:  
 return [word\_to\_translate, word\_translation]  
 return None

rus\_to\_eng.py

from english\_game\_state import EnglishGameState  
import csv  
  
  
class RusToEng(EnglishGameState):  
 *"""состояние игры EnglishGame (с русского на английский)"""* def \_\_str\_\_(self):  
 *"""преобразование в строку"""* return "<<перевод с русского на английский>>"  
  
 def get\_word\_and\_translation(self, filename, index):  
 *"""получение слова и его перевода по индексу (номеру упражнения)"""* with open(filename, 'r', newline='') as File:  
 reader = csv.reader(File, delimiter=';')  
 counter = 0  
 for row\_i, row in enumerate(reader):  
 word\_translation = row[0] *# слово для перевода (русское)* word\_to\_translate = row[1] *# сам перевод слова (английское)* counter += 1  
 if row\_i == index:  
 return [word\_to\_translate, word\_translation]  
 return None

observer.py

from abc import ABC, abstractmethod  
  
  
class Observer(ABC):  
 *"""наблюдатель"""* @abstractmethod  
 def \_\_init\_\_(self):  
 self.game\_states = {}  
 self.personal\_ads = None  
  
 @abstractmethod  
 def update\_personal\_ads(self, game\_name, game\_state):  
 *"""подобрать подходящую рекламу"""* pass

message.py

INIT\_MESSAGE = '<<init message>>'  
  
  
class Message:  
 *"""шаблон сообщения"""* def \_\_init\_\_(self):  
 self.message\_text = INIT\_MESSAGE  
  
 def reset(self):  
 *"""Сбросить текст сообщения"""* self.message\_text = INIT\_MESSAGE  
  
 def set\_message\_text(self, is\_correct, correct\_answer=''):  
 *"""установка параметров сообщения"""* if is\_correct:  
 self.message\_text = f'Вы ответили правильно! (ответ: {correct\_answer})'  
 else:  
 self.message\_text = 'Вы ответили неверно!'

message\_pool.py

from message import Message  
  
  
class MessagePool:  
 *"""пул шаблонов сообщений"""* def \_\_init\_\_(self, \*games):  
 self.pool = {}  
 for game in games:  
 self.pool.setdefault(game, Message())  
  
 def acquire(self, game):  
 *"""забрать инициализированный объект сообщения"""* try:  
 return self.pool.pop(game)  
 except:  
 raise Exception('Не удалось получить шаблон сообщения из словаря')  
  
 def release(self, game, message):  
 *"""вернуть шаблон сообщения в пул"""* message.reset()  
 self.pool.setdefault(game, message)

main.py

from quiz\_game import QuizGame  
from math\_game import MathGame  
from math\_game\_proxy import MathGameProxy  
from english\_game import EnglishGame  
from game\_complex\_facade import GameComplexFacade  
from information\_expert import InformationExpert  
  
QUIZ\_ASK\_FILE = 'quiz\_asks.csv'  
ENGLISH\_WORDS\_FILE = 'english\_words.csv'  
  
  
def main():  
 print('Обучающий игровой комплекс для младших школьников')  
 quiz\_game = QuizGame(QUIZ\_ASK\_FILE)  
 *# math\_game\_proxy = MathGameProxy()* math\_game = MathGame()  
 *# math\_game\_proxy.set\_original\_math(math\_game)* english\_game = EnglishGame(ENGLISH\_WORDS\_FILE)  
 info\_expert = InformationExpert(math\_game, quiz\_game, english\_game)  
 simple\_game\_interface = GameComplexFacade(info\_expert, math\_game, quiz\_game, english\_game)  
  
 while True:  
 choice = input(f'\n{"-" \* 100}\nСделайте выбор:\n\t1 - математика'  
 f'\n\t2 - викторина (по русскому языку)'  
 f'\n\t3 - английские слова'  
 f'\n\t4 - изменить направление перевода в "Английских словах"'  
 f'\n\t5 - показать информацию об опыте в играх'  
 f'\n\t6 - сохранить/восстановить прогресс'  
 f'\n\t7 - управлять рекламой'  
 f'\n\t8 - создать объекты игр'  
 f'\n\n\t0 - выйти\n:')  
 if choice == '1':  
 simple\_game\_interface.start\_math\_game\_exercise()  
 elif choice == '2':  
 simple\_game\_interface.start\_quiz\_game\_exercise()  
 elif choice == '3':  
 simple\_game\_interface.start\_english\_game\_exercise()  
 elif choice == '4':  
 simple\_game\_interface.change\_english\_game\_state()  
 elif choice == '5':  
 simple\_game\_interface.show\_information\_about\_level()  
 elif choice == '6':  
 simple\_game\_interface.perform\_memento\_action()  
 elif choice == '7':  
 simple\_game\_interface.manage\_ads()  
 elif choice == '8':  
 simple\_game\_interface.create\_games()  
 elif choice == '0':  
 return  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()