ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ РЕСПУБЛИКИ ХАКАСИЯ

«ХАКАССКИЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ»

Специальность 09.02.07

«Информационные системы и программирование»

Практическая работа №3

тема: **Разработка программного модуля с применением принципов ООП**

Руководитель:

\_\_\_\_\_\_\_\_ А.А. Отыргашева

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка, дата)

Выполнил:

Студент группы ИС(ПРО)-31

\_\_\_\_\_\_\_\_ Д. А. Коробкова

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Абакан 2025 г.

**Задание №1.** Написать и выполнить код. Модифицировать код таким образом, чтобы в классе поля были закрыты для других пользователей (определить сеттеры и геттеры), добавить конструкторы. Выполнить тест.

Добавленные элементы: геттер и сеттер для переменной life, конструктор \_\_init\_\_.

Геттер и сеттер для переменной life: они предоставляют доступ к приватной переменной класса — перемнной life. Метод get\_live позволяет получить текущее значение \_life, а метод set\_live позволяет изменить его.

Конструктор \_\_init\_\_: Конструктор используется для инициализации объекта при его создании. В данном коде он устанавливает начальное значение переменной \_life, равное 3. Это позволяет гарантировать, что каждый новый объект класса Rival будет начинаться с одинакового состояния, что важно для логики игры.

# Создание класса

class Rival:

# Создание метода для инициализации класса, который будет вызываться при создании нового объекта класса

def \_\_init\_\_(self):

# Установка начального количества жизней соперника равное 3

self.\_life = 3

# Создание геттера для получения текущего количества жизней соперника

def get\_live(self):

# Возвращение текущего количества жизней соперника

return self.\_life

# Создание сеттера для установки нового значения количества жизней соперника

def set\_live(self, live):

# Обновление количества жизней на полученное значение

self.\_live = live

# Создание метода для атаки соперника

def attack(self):

# Вывод строки

print("Ouch!")

# Текущее количество жизней становится на одну меньше

self.\_life -=1

# Создание метода для проверки количества жизней соперника

def checkLife(self):

# Условие if для проверки, если количество жизней меньше или равно 0

if self.\_life <=0:

# Вывод строки

print("You won!")

# В ином случае

else:

# Вывод строки

print(self.\_life)

# Создание новых объектов класса

thanos = Rival()

magneto = Rival()

# Вызов метода для атаки соперника

thanos.attack()

thanos.attack()

thanos.attack()

# Проверка жизней соперника

thanos.checkLife()

# Вывод пустой строки для удобства

print()

# Вызов метода для атаки соперника

magneto.attack()

magneto.attack()

magneto.attack()

# Проверка жизней соперника

magneto.checkLife()

**Тест №1**

1) 1. Класс – это шаблон, по которому создается объект.

2) 3. Объект класса – это экземпляр класса.

3) 3. Для того чтобы создать класс, нужно использовать ключевое слово class.

4) 1. У классов есть свойства.

5) 2. self – это обязательное ключевое слово в Python, которое записывается в аргумент функции класса.

6) def attack(self):

print(“Ouch!”)

self.life = -1

**Тест №2**

1) 1. Модули — это файлы с кодом Python.

2. 2. У файлов модулей расширение – .py.

3. Модули позволяют повторно использовать один и тот же код в программе.

2) 2. Для того, чтобы импортировать модуль, мы должны использовать ключевое слово import.

3) 2. from datetime import date as d

1. birthday = d(2017, 6, 1)

3. print(birthday)

4) 2. Когда хотим переименовать имя модуля.

5) 1. Пакет модулей должен содержать файл \_\_init\_\_.py.

2. Пакет модулей позволяет хранить модули вместе.

**Задание №3.** По описанию шагов разработать мини-игру. Дополнить своими функциями. Применить принципы ООП для программного кода. Листинг кода игры с комментариями. Описать примененные принципы ООП и способы их применения в вашем коде.

Примененные принципы ООП:

1) Инкапсуляция: заключается в том, что данные (в данном случае, параметры distance, speed, и fuel) и методы, которые работают с этими данными, объединены в одном классе SoyuzDocking.

**2) Абстракция: к**ласс SoyuzDocking абстрагирует сложные процессы, такие как управление топливом и скоростью, предоставляя простые методы для выполнения операций, таких как perform\_burn, update\_distance, и has\_docked.

Добавленные функции:

1) В метод perform\_burn было добавлено условие if для проверки достаточно ли топлива для выполнения сжигания.

2) Создание метода display\_status для вывода на экран текущего состояния корабля.

3) Создание метода enable\_autopilot для активации автопилота.

4) Создание метода hand\_docking\_check для проверки, успешно ли прошла ручная стыковка.

5) Создание блока обработки исключений.

# Инструкция для пользователя

print("Добро пожаловать в симуляцию стыковки Союз Т-6!")  
print("Ваша миссия – стыковка со станцией Салют-7.")  
print("Вы можете управлять скоростью космического корабля сжигая топливо.")  
print("Каждая единица сожженного топлива замедляет космический корабль на 1 м/с.")  
print("Удачи экипажу!\n")

# Определение класса  
class SoyuzDocking:

# Метод инициализатора  
 def \_\_init\_\_(self):

# Создание атрибута distance, который хранит расстояние до станции "Салют 7"  
 self.distance = 500

# Создание атрибута speed, который хранит скорость корабля  
 self.speed = 50

# Создание атрибута fuel, который хранит количество топлива  
 self.fuel = 100

# Создание метода для сжигания топлива  
 def perform\_burn(self, burn\_amount):

# В условии if проверяется, достаточно ли топлива для выполнения сжигания   
 if burn\_amount > self.fuel:

# Вывод строки в консоль  
 print("Топлива не так много, чтоб вот таким заниматься придумал тоже")

# Возвращение False, если топлива недостаточно   
 return False

# Уменьшение скорости корабля на burn\_amount, но не ниже нуля  
 self.speed = max(self.speed - burn\_amount, 0)

# Уменьшение количества топлива на burn\_amount, но не ниже 0  
 self.fuel = max(self.fuel - burn\_amount, 0)

# Возвращение True в случае, если операция выполнена успешно  
 return True

# Создание метода update\_distance для обновления расстояния до станции   
 def update\_distance(self):

# Уменьшение расстояния на значение скорости, но не ниже нуля  
 self.distance = max(self.distance - self.speed, 0)

# Создание метода has\_docked для проверки, достиг ли космический корабль станции  
 def has\_docked(self):

# Метод возвращает True, если расстояние меньше или равно 0  
 return self.distance <= 0

# Создание метода display\_status для вывода на экран текущего состояния корабля  
 def display\_status(self):

# Вывод на консоль расстояние до станции, скорость и количество топлива  
 print(f"Расстояние до станции Салют-7: {self.distance} метров")  
 print(f"Скорость: {self.speed} метров в секунду")  
 print(f"Топливо: {self.fuel} килограмм")

# Создание метода enable\_autopilot для активации автопилота  
 def enable\_autopilot(self):

# Вывод строки на консоль  
 print("Автопилот включен. Стыковка завершена.")

# Установка значений расстояния и скорости в ноль   
 self.distance = 0  
 self.speed = 0

# Создание метода для проверки, успешно ли прошла ручная стыковка  
 def hand\_docking\_check(self):

# Цикл if проверяет, находится ли космический корабль на расстоянии 11 метров или меньше и скорость меньше или равна расстоянию  
 if self.distance <= 11 and self.speed <= self.distance:

# В случае выполнения всех условий выводится строка  
 print("Стыковка подтверждена. Поздравляем экипаж!!")

# В ином случае выводит следующая строка  
 else:  
 print("Ничо не получилось, блин-блинский")  
  
# Создание экземпляра класса SoyuzDocking  
docking\_sequence = SoyuzDocking()  
# Создание цикла while, который будет работать пока космический корабль не достигнет станции  
while not docking\_sequence.has\_docked():

#Вывод внутри цикла текущего состояния корабля  
 docking\_sequence.display\_status()

# Создание цикла if для проверки осталось ли топливо  
 if docking\_sequence.fuel <= 0:

# В случае выполнения условия выводится строка  
 print("Милорд!! Топливо кончилось!!!")

# Прерывание цикла  
 break

# Создание цикла if для проверки является ли расстояние до станции меньше или равно 11 метрам   
 if docking\_sequence.distance <= 11:

# Запрос у пользователя добра на запуск автопилота  
 autopilot = input("До станции Салют-7 осталось меньше 11 метров. Включить автопилот? (Да/Нет): ").lower()

# Если ответ пользователя да  
 if autopilot == 'да':

# То включается автопилот  
 docking\_sequence.enable\_autopilot()

# И цикл прерывается  
 break

# В ином случае выводится следующая строка  
 else:  
 print("Ну нет так нет, давай сам")

# Прерывание цикла  
 break

# Начало блока обработки исключений   
 try:

# Запрос у пользователя ввод количества топлива  
 burn\_amount = int(input("Сколько топлива сжечь для снижения скорости: "))

#Если введенное значение отрицательное выводится следующая строка   
 if burn\_amount < 0:

print("Количество топлива не может быть отрицательным, шутишь что ли")

# Цикл продолжается  
 continue

# Если сжигание топлива прошло успешно, то выполняется следующий блок кода

if docking\_sequence.perform\_burn(burn\_amount):

# Расстояние обновляется на основе текущей скорости  
 docking\_sequence.update\_distance()

# Если произошла ошибка при преобразовании ввода в целое число  
 except ValueError:

# Выводится строка   
 print("Пожалуйста, ну введи ты нормальное число, не до шуток")

# После завершения цикла вызывается метод для проверки, была ли стыковка успешной  
docking\_sequence.hand\_docking\_check()

**Задание по вариантам (Вариант №8).** Определить сущности, выделить их основные характеристики и поведение. Создать модули с классами (конструкторы, поля (защищенные) не менее 5, сеттеры и геттеры (методы get/set) для каждого приватного поля, методы (функции) не менее 3 разных видов (простой, с входными параметрами, с входными и выходными параметрами), наследованные классы - минимально 1, переопределение методов - минимум 1).

Описание сущностей, их характеристики и поведение. Программный код - Листинг программного кода с комментариями. Описать программный код (4-5 предложений).

**Предметная область**: кинотеатр

Сущность №1 — Билет

Характеристики: место, фильм и зал

Поведение: отображение информации о билете

Сущность №2 — Зал

Характеристики: номер, вместительность

Поведение: добавить фильм, удалить фильм, отображение информации о зале, отображение списка показываемых фильмов

Сущность №3 — Фильм

Характеристики: название, жанр, продолжительность, рейтинг, режиссер, год релиза

Поведение: отображение информации о фильме, проверка на продолжительность фильма, обновление рейтинга

Сущность №4 — Посетитель

Характеристики: фамилия, имя, возраст

Поведение: отображение информации о посетителе, проверка на совершеннолетие пользователя

Сущность №5 — Бронирование

Характеристики: место, фильм, зал, время

Поведение: отображение информации о бронировании

Описание кода: данный код реализует систему бронирования билетов для кинотеатра с использованием объектно-ориентированного программирования. Он включает несколько классов: ticket, hall, movie, visitor и booking. Класс ticket отвечает за хранение информации о месте, фильме и зале, предоставляет методы для получения и изменения этих данных. Класс hall управляет информацией о зале, включая его номер и вместимость, а также список показываемых фильмов. Класс movie хранит данные о фильме: название, жанр, продолжительность и рейтинг, и включает методы для отображения информации о фильме и проверки его длительности. Класс visitor представляет посетителя кинотеатра с его личными данными, а класс booking расширяет функциональность класса ticket, добавляя возможность указания времени бронирования.

# Создание класса ticket

class ticket:

# Создание метода \_\_init\_\_, который создает новый объект класса ticket с параметрами seat, movie и hall

def \_\_init\_\_(self, seat, movie, hall):

# Определение переменной self.\_seat для сохранения номера места в переменной \_seat

self.\_seat = seat

# Определение переменной self.\_movie для сохранения информации о фильме в переменной \_movie

self.\_movie = movie

# Определение переменной self.\_hall для сохранения информации о зале в переменной \_hall

self.\_hall = hall

# Создание геттера get\_seat для получения номера места

def get\_seat(self):

# Возвращение номера места

return self.\_seat

# Создание сеттера set\_seat для изменения номера места

def set\_seat(self, seat):

# Определение переменной self.\_seat для установки нового значения в переменную seat

self.\_seat = seat

# Создание геттера get\_movie для получения информации о фильме

def get\_movie(self):

# Возвращение информации о кино

return self.\_movie

# Создание сеттера set\_movie для изменения информации о фильме

def set\_movie(self, movie):

# Определение переменной self.\_movie для установки нового значения в атрибут movie

self.\_movie = movie

# Создание геттера get\_hall для получения информации о зале

def get\_hall(self):

# Возвращает информацию о зале

return self.\_hall

# Создание сеттера set\_hall для изменения информации о зале

def set\_hall(self, hall):

# Определение переменной self.\_hall для установки нового значения в атрибут hall

self.\_hall = hall

# Создание метода для отображения информации о билете

def show\_info\_ticket(self):

# Возвращение строки с информацией о месте, зале и фильме

return f"\nМесто: {self.\_seat} \nЗал: {self.\_hall.get\_number()} \nКино: {self.\_movie.show\_info\_movies()}"

# Создание класса hall

class hall:

# Создание метода \_\_init\_\_, который инициализирует новый объект класса hall с параметрами number и amount

def \_\_init\_\_(self, number, amount):

# Определение переменной self.\_number для сохранения номера зала в переменной \_number

self.\_number = number

# Определение переменной self.\_amount для сохранения номера зала в переменной \_amount

self.\_amount = amount

# Создание списка для хранения списка кино в зале

self.\_movies\_showing = []

# Создание геттера для получения информации о номере зала

def get\_number(self):

# Возвращение номера зала

return self.\_number

# Создание сеттера для изменения информации о номере зала

def set\_number(self, number):

# Определение переменной для установка нового значения номера зала в атрибут number

self.\_number = number

# Создание геттера для для получения информации о вместительности зала

def get\_amount(self):

# Возвращение значения вместительности зала

return self.\_amount

# Создание сеттера для изменения вместительности зала

def set\_amount(self, amount):

# Определение переменной для установки нового значения в атрибут amount

self.\_amount = amount

# Создание метода для добавления фильма в список показываемых фильмов

def add\_movies(self, movie):

# Если фильма нет в списке

if movie not in self.\_movies\_showing:

# То добавляем данные о нем в список

self.\_movies\_showing.append(movie)

# Создание метода для удаления фильма из списка показываемых

def delete\_movies(self, movie):

# Если фильм находится в списке

if movie in self.\_movies\_showing:

# То удаляем его оттуда

self.\_movies\_showing.remove(movie)

# Создание метода для показа информации о зале

def show\_info\_hall(self):

# Вывод информации о номере зала и его вместительности

return f"\nНомер зала: {self.\_number} \nВместительность зала: {self.\_amount}"

# Создание метода для показа списка фильмов

def what\_movies(self):

# Возвращение списка фильмов

return [movie.get\_name() for movie in self.\_movies\_showing]

# Создание класса movie

class movie:

# Создание метода, который создает новый объект класса с параметрами название, жанр, продолжительность, рейтинг, режиссер и год выпуска

def \_\_init\_\_(self, name, genre, duration, rating, director, release):

# Определение переменной для сохранения названия в переменной name

self.\_name = name

# Определение переменной для сохранения жанра в переменной genre

self.\_genre = genre

# Определение переменной для сохранения продолжительности в переменной duration

self.\_duration = duration

# Определение переменной для сохранения рейтинга в переменной rating

self.\_rating = rating

# Определение переменной для сохранения режиссера в переменной director

self.\_director = director

# Определение переменной для сохранения года выпуска фильма в переменной release

self.\_release = release

# Создание геттера для получения названия фильма

def get\_name(self):

# Возвращение названия фильма

return self.\_name

# Создание сеттера для изменения названия фильма

def set\_name(self, name):

# Определение переменной для установки нового значения в переменную name

self.\_name = name

# Создание геттера для получения жанра фильма

def get\_genre(self):

# Возвращения жанра фильма

return self.\_genre

# Создание сеттера для изменения жанра фильма

def set\_genre(self, genre):

# Определение переменной для установки нового значения в переменную genre

self.\_genre = genre

# Создание геттера для получения информации о продолжительности фильма

def get\_duration(self):

# Возвращение продолжительности фильма

return self.\_duration

# Создание сеттера для изменения продолжительности фильма

def set\_duration(self, duration):

# Создание переменной для установки нового значения в переменную duration

self.\_duration = duration

# Создание геттера для получения информации о рейтинге

def get\_rating(self):

# Возвращение рейтинга

return self.\_rating

# Создание сеттера для изменения рейтинга фильма

def set\_rating(self, rating):

# Создание переменной для установки нового значения в переменную rating

self.\_rating = rating

# Создание геттера для получения информации о режиссере

def get\_director(self):

# Возвращение информации о режиссере

return self.\_director

# Создание сеттера для изменения информации о режиссере

def set\_director(self, director):

# Создание переменной для установки нового значения в переменную director

self.\_director = director

# Создание геттера для получения информации о годе релиза фильма

def get\_release(self):

# Возвращение года релиза

return self.\_release

# Создание сеттера для изменения информации о годе релиза

def set\_release(self, release):

# Создание переменной для установки нового значения в переменную release

self.\_release = release

# Создание метода для показа информации о кино

def show\_info\_movies(self):

# Возвращение информации о названии, жанре, годе релиза, режиссере, продолжительности и рейтинге фильма

return f"\nНазвание: '{self.\_name}' ({self.\_release}) \nЖанр: {self.\_genre} \nРежиссер: {self.\_director} \nПродолжительность: {self.\_duration} минут \nРейтинг: {self.\_rating}"

# Создание для проверки может ли фильм называться продолжительным

def is\_movie\_long(self):

# Если продолжительность фильма больше 120

if self.\_duration > 120:

# То возвращает строку

return ("ДА")

# В противоположном случае

else:

# Возвращает строку

return ("НЕТ")

# Создание метода для обновления рейтинга фильма

def update\_rating(self, new\_rating):

# Если новый рейтинг от 1 до 10

if 1 <= new\_rating <= 10:

# То происходит обновление рейтинга фильма

self.set\_rating(new\_rating)

# Функция возвращает True, указывая на то, что рейтинг был успешно обновлен

return True

# В ином случает функция вернет False

return False

# Создание класса visitor

class visitor:

# Создание метода, который создает новый объект класса с параметрами фамилия, имя и возраст

def \_\_init\_\_(self, surname, name, age):

# Определение переменной для сохранения названия в переменной surname

self.\_surname = surname

# Определение переменной для сохранения названия в переменной name

self.\_name = name

# Определение переменной для сохранения названия в переменной age

self.\_age = age

# Создание геттера для получения фамилии посетителя

def get\_surname(self):

# Возвращение фамилии посетителя

return self.\_surname

# Создание сеттера для изменения фамилии посетителя

def set\_surname(self, surname):

# Определение переменной для установки нового значения в переменную surname

self.\_surname = surname

# Создание геттера для получения имени посетителя

def get\_name(self):

# Возвращение имени посетителя

return self.\_name

# Создание сеттера для изменения имени посетителя

def set\_name(self, name):

# Определение переменной для установки нового значения в переменную name

self.\_name = name

# Создание геттера для получения возраста посетителя

def get\_age(self):

# Возвращение возраста посетителя

return self.\_age

# Создание сеттера для изменения возраста посетителя

def set\_age(self, age):

# Определение переменной для установки нового значения в переменную age

self.\_age = age

# Создание функции для показа информации о посетителе

def show\_info\_visitor(self):

# Возвращение информации о посетителе

return f"\nФамилия: {self.\_surname} \nИмя: {self.\_name} \nВозраст: {self.\_age} лет"

# Создание функции для проверки совершеннолетний ли посетитель

def is\_adult(self):

# Если значение переменной возраст больше 18

if self.\_age > 18:

# То выводится строка

return ("ДА")

# В ином случае

else:

# Выводится строка

return ("НЕТ")

# Создание дочернего класса booking

class booking(ticket):

# Создание метода, который создает новый объект класса с параметрами место, фильм, зал и время

def \_\_init\_\_(self, seat, movie, hall, time):

# Создание функции, которая возвращает методы родительского класса

super().\_\_init\_\_(seat, movie, hall)

# Определение переменной для сохранения названия в переменной time

self.\_time = time

# Создание геттера для получения времени бронирования

def get\_time(self):

# Возвращение времени бронирования

return self.\_time

# Создание сеттера для изменения времени бронирования

def set\_time(self, time):

# Определение переменной для установки нового значения в переменную time

self.\_time = time

# Создание метода для показа информации о бронировании

def show\_info\_booking(self):

# Вызов метода show\_info\_ticket() из родительского класса текущего класса с помощью функции super()

base = super().show\_info\_ticket()

# Возвращение информации о бронировании

return f"{base} \nВремя бронирования: {self.\_time}"

# Создание объектов класса movie

movie1 = movie("Майор Гром: Игра", "боевик, приключения", 168, 7.5, "Олег Трофим", 2024)

movie2 = movie("Корпорация монстров", "мультфильм, фэнтези, комедия, приключения, семейный", 92, 8.1, "Пит Доктер", 2001)

movie3 = movie("Гадкий я 4", "мультфильм, фантастика, комедия, криминал, приключения, семейный", 94, 6.5, "Крис Рено", 2024)

# Создание объектов класса hall

hall1 = hall(1, 40)

hall2 = hall(2, 25)

# Добавление фильмов в списки показываемых фильмов в залах

hall1.add\_movies(movie1)

hall1.add\_movies(movie2)

hall2.add\_movies(movie1)

hall2.add\_movies(movie2)

hall2.add\_movies(movie3)

# Создание объектов класса visitor

visitor1 = visitor("Hayes", "Nora", 22)

visitor2 = visitor("Harris", "Alice", 14)

visitor3 = visitor("Martinez", "Chris", 37)

# Создание объектов класса ticket

ticket1 = ticket(12, movie2, hall2)

ticket2 = ticket(15, movie1, hall1)

# Создание объектов класса booking

booking1 = booking(12, movie2, hall2, "12:30")

booking2 = booking(15, movie1, hall1, "21:00")

# Вывод строки для удобства

print("\nИНФОРМАЦИЯ О КИНО")

# Вывод метода show\_info\_movies

print(movie1.show\_info\_movies())

print(movie2.show\_info\_movies())

print(movie3.show\_info\_movies())

# Вывод строки для удобства

print("\nИНФОРМАЦИЯ О ЗАЛАХ")

# Вывод метода show\_info\_hall

print(hall1.show\_info\_hall())

print(hall2.show\_info\_hall())

# Вывод строки для удобства  
print("\nИНФОРМАЦИЯ О ПОСЕТИТЕЛЯХ")

# Вывод метода show\_info\_visitors

print(visitor1.show\_info\_visitor())

print(visitor2.show\_info\_visitor())

print(visitor3.show\_info\_visitor())

# Вывод строки для удобства

print("\nИНФОРМАЦИЯ О БИЛЕТАХ")

#Вывод метода show\_info\_ticket

print(ticket1.show\_info\_ticket())

print(ticket2.show\_info\_ticket())

# Вывод строки для удобства

print("\nИНФОРМАЦИЯ О БРОНИРОВАНИИ")

#Вывод метода show\_info\_booking

print(booking1.show\_info\_booking())

print(booking2.show\_info\_booking())

# Вывод строки для удобства

print("\nПРОВЕРКА РАБОТЫ МЕТОДА 'what\_movies'")

#Вывод метода what\_movies

print(f"Кино в первом зале:{hall1.what\_movies()}")

print(f"Кино во втором зале:{hall2.what\_movies()}")

# Вывод строки для удобства

print("\nПРОВЕРКА РАБОТЫ МЕТОДА 'is\_movie\_long'")

# Вывод метода is\_movie\_long

print(f"Фильм '{movie1.get\_name()}' идет более двух часов?"+ f" Ответ: {movie1.is\_movie\_long()}")

print(f"Фильм '{movie2.get\_name()}' идет более двух часов?"+ f" Ответ: {movie2.is\_movie\_long()}")

print(f"Фильм '{movie3.get\_name()}' идет более двух часов?" + f" Ответ: {movie3.is\_movie\_long()}")

# Вывод строки для удобства

print("\nПРОВЕРКА РАБОТЫ МЕТОДА 'update\_rating'")

# Запрос у пользователя ввода нового рейтинга для movie1

rat1 = float(input(f"Введите рейтинг для фильма {movie1.get\_name()} (от 1 до 10): "))

# Вывод текущего рейтинга фильма

print(f"\nРейтинг фильма '{movie1.get\_name()}' - {movie1.\_rating}")

# Проверка попадает ли новый рейтинг в диапозон

if movie1.update\_rating(rat1):

# Вывод строки

print(f"Рейтинг фильма '{movie1.get\_name()}' успешно обновлен!!")

# Если новый рейтинг не попадает в диапозон

else:

# То выводится следующая строка

print(f"ОШИБКА: Рейтинг фильма '{movie1.get\_name()}' должен быть от 1 до 10")

# Вывод обновленного рейтинга фильма

print(f"Актуальный рейтинг фильма '{movie1.get\_name()}' - {movie1.\_rating}")

#Запрос у пользователя ввода нового рейтинга для movie2

rat2 = float(input(f"\nВведите рейтинг для фильма {movie2.get\_name()} (от 1 до 10): "))

# Вывод текущего рейтинга фильма

print(f"\nРейтинг фильма '{movie2.get\_name()}' - {movie2.\_rating}")

# Проверка попадает ли новый рейтинг в диапозон

if movie2.update\_rating(rat2):

# Вывод строки

print(f"Рейтинг {movie2.get\_name()} успешно обновлен!!")

# Если новый рейтинг не попадает в диапозон

else:

# То выводится следующая строка

print(f"ОШИБКА: Рейтинг фильма '{movie2.get\_name()}' должен быть от 1 до 10")

# Вывод обновленного рейтинга фильма

print(f"Актуальный рейтинг фильма '{movie2.get\_name()}' - {movie2.\_rating}")

# Запрос у пользователя ввода нового рейтинга для movie3

rat3 = float(input(f"\nВведите рейтинг для фильма {movie3.get\_name()} (от 1 до 10): "))

# Вывод текущего рейтинга фильма

print(f"\nРейтинг фильма '{movie3.get\_name()}' - {movie3.\_rating}")

# Проверка попадает ли новый рейтинг в диапозон

if movie3.update\_rating(rat3):

# Вывод строки

print(f"Рейтинг {movie3.get\_name()} успешно обновлен!!")

# Если новый рейтинг не попадает в диапозон

else:

# То выводится следующая строка

print(f"ОШИБКА: Рейтинг фильма '{movie3.get\_name()}' должен быть от 1 до 10")

# Вывод обновленного рейтинга фильма

print(f"Актуальный рейтинг фильма '{movie3.get\_name()}' - {movie3.\_rating}")

# Вывод строки для удобства

print("\nПРОВЕРКА РАБОТЫ МЕТОДА 'delete\_movies'")

# Вывод метода what\_movies

print(f"Список фильмов в зале {hall2.get\_number()}: {hall2.what\_movies()}")

hall2.delete\_movies(movie1)

print(f"Обновленный список фильмов в зале {hall2.get\_number()}: {hall2.what\_movies()}")

# Вывод строки для удобства

print("\nПРОВЕРКА РАБОТЫ МЕТОДА 'is\_adult'")

# Вывод метода is\_adult

print(f"{visitor1.\_surname} {visitor1.\_name} совершеннолетний (-яя)?" + f" Ответ: {visitor1.is\_adult()}")

print(f"{visitor2.\_surname} {visitor2.\_name} совершеннолетний (-яя)?" + f" Ответ: {visitor2.is\_adult()}")

print(f"{visitor3.\_surname} {visitor3.\_name} совершеннолетний (-яя)?" + f" Ответ: {visitor3.is\_adult()}")