Министерство образования Республики Беларусь

УО «Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №6

По дисциплине: “Языки программирования”

Тема: “Классы. Инкапсуляция. Наследование”Вариант №6

Выполнил:

студент 2 курса

группы ПО-7

Комиссаров Андрей

Проверила: Дряпко А. В.

Брест 2021

**Цель работы**: ознакомиться с принципами инкапсуляции и наследования

**Постановка задачи**

**Задание 1**

1. Определить пользовательский класс – «Банк»

2. Определить счетчик

3. Определить в классе конструкторы с параметрами и без. Конструктор должен выводить сообщение о количестве объектов.

4. Определить в классе внешние компоненты-функции для получения и установки полей данных.

5. Написать демонстрационную программу, в которой объекты пользовательского класса создаются с помощью неявного использования конструкторов без параметров.

6. Показать в программе явное использование конструкторов с параметрами. Определение пользовательского класса

Поля класса «Банк»:

* name: str название
* money: int капитализация
* workers: int количество сотрудников

**Код программы:**

class Bank:

counter = 0

\_\_name: str

\_\_money: int

\_\_workers: int

def \_\_init\_\_(self, name: str = 'без параметра', money: int = 0, workers: int = 0) -> None:

Bank.counter += 1

self.\_\_name = name

self.\_\_money = money

self.\_\_workers = workers

print(f'Создан {self}')

def \_\_str\_\_(self) -> str:

return f'Банк (name: {self.\_\_name}, капитал: ${self.\_\_money}, ' \

f'рабочие: {self.workers}; счётчик: {Bank.counter})'

def \_\_del\_\_(self) -> None: #деструктор

print(f'Удалён банк "{self.\_\_name}"')

@property

def name(self) -> str:

return self.\_\_name

@name.setter

def name(self, name: str) -> None:

self.\_\_name = name

@property

def money(self) -> int:

return self.\_\_money

@money.setter

def money(self, money: int) -> None:

self.\_\_money = money

@property

def workers(self) -> int:

return self.\_\_workers

@workers.setter

def workers(self, workers: int) -> None:

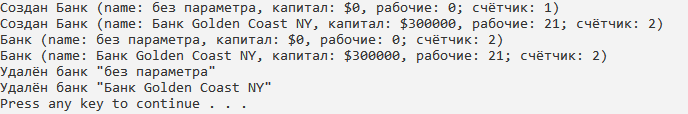
self.\_\_workers = workers

defaultBank=Bank()

specialBank=Bank('Банк Golden Coast NY', 300000, 21)

print(defaultBank)

print(specialBank)

**Результат работы программы:**

**Задание 2**

Для варианта «Спортсмен-Персона-Призер» выполнить следующее:

1. Построить модель предметной области в соответствии со своим вариантом. Разработать классы в соответствии с моделью предметной области

3. Для каждого класса создать конструктор

4. Для каждого класса создать внешние функции установки и получения полей данных

5. Для каждого класса разработать функции, позволяющие представить на экране значения полей данных

6. Для каждого класса разработать функции, позволяющие вводить с консоли значения полей данных

7. Написать демонстрационную программу, иллюстрирующую использование разработанных классов

8. Построить диаграмму классов

9. Разработанные классы разместить в одном пакете

Модель предметной области

Родительским классом является персона. У него есть поля: имя, возраст и пол. От него наследуется спортсмен с дополнительным полем вид спорта и призёр c дополнительным полем занятое призовое место.

**Код программы:**

# Родительским классом является персона.

# У него есть поля: имя, возраст и пол.

# От него наследуется спортсмен с дополнительным полем вид спорта

# и призёр c дополнительным полем занятое призовое место.

class Person:

\_name: str

\_age: int

\_sex: str

def \_\_init\_\_(self, name='nameless', age=0, sex='none') -> None:

self.\_name = name

self.\_age = age

self.\_sex = sex

print(f'Создана персона: {self.\_name}, {self.\_age}, {self.\_sex}')

def \_\_str\_\_(self) -> str:

return f'Персона: {self.\_name}, {self.\_age}, {self.\_sex}'

def \_\_del\_\_(self) -> None:

print(f'Удалён(удалена) {self}')

@property

def name(self) -> str:

return self.\_name

@name.setter

def name(self, name) -> None:

self.\_name = name

@property

def age(self) -> int:

return self.\_age

@age.setter

def age(self, age) -> None:

self.\_age = age

@property

def sex(self) -> str:

return self.\_sex

@sex.setter

def sex(self, sex) -> None:

self.\_sex = sex

class Sportsman(Person):

\_sport: str

def \_\_init\_\_(self, name='nameless', age=0, sex='none', sport='no sport') -> None:

super().\_\_init\_\_(name, age, sex)

self.\_sport = sport

prev: str = super().\_\_str\_\_()

print(f'Создан спортсмен {prev[prev.find(":"):]}, {self.\_sport}')

def \_\_str\_\_(self) -> str:

prev: str = super().\_\_str\_\_()

return 'Спортсмен' + prev[prev.find(':'):] + f', {self.\_sport}'

@property

def sport(self) -> str:

return self.\_sport

@sport.setter

def sport(self, sport) -> None:

self.\_sport = sport

class Prizewinner(Sportsman):

\_place: int

def \_\_init\_\_(self, name='nameless', age=0, sex='none', sport='no sport', place=0) -> None:

super().\_\_init\_\_(name, age, sex, sport)

self.\_place = place

prev: str = super().\_\_str\_\_()

print(f'Создан призёр {prev[prev.find(":"):]}, {self.\_place}')

def \_\_str\_\_(self) -> str:

prev: str = super().\_\_str\_\_()

return 'Призёр' + prev[prev.find(':'):] + f', {self.\_place}'

@property

def place(self) -> int:

return self.\_place

@place.setter

def place(self, place) -> None:

self.\_place = place

def main():

print('Создание экземпляров класса:')

person = Person('человек', 60, 'муж')

sportsman = Sportsman('Дарья', 25, 'жен', 'лыжные гонки')

prizewinner = Prizewinner('Виктор', 21, 'муж', 'футбол', 2)

people = [person, sportsman, prizewinner]

person.name = 'Анна'

sportsman.sport = 'баскетбол'

prizewinner.place = 1

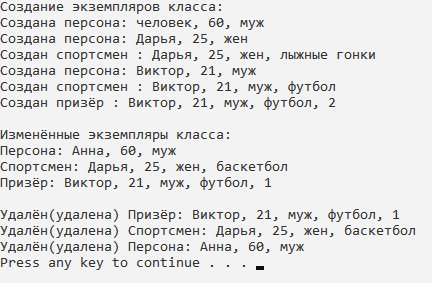
print('\nИзменённые экземпляры класса:', \*people, sep='\n')

print('')

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Результаты программы:**



**Вывод:** в ходе выполнения я изучил принципы инкапсуляции и наследования