

Дополнительные аспекты ООП

В Python все классы **наследуются** от класса `object`, обладающего некоторыми атрибутами по умолчанию (например `__init__`, `__doc__`, `__str__` и т.д.).

Дочерние классы могут изменять поведение атрибутов класса-родителя, переопределив (англ. *Override*) их. При этом, обычно, дочерний класс дополняет родительский метод, добавив свой код после кода родителя (используя функцию `super()`, предоставляющую ссылку на родительский класс).

Каждый класс также может получить информацию о своих «родителях» через метод `__bases__` или `isinstance()`.

Полиморфизм позволяет интерпретировать любой объект, как экземпляр не только текущего класса, но и любого из его базовых классов. В компилируемых языках программирования полиморфизм достигается за счет создания виртуальных методов, которые в отличие от неvirtуальных можно перегрузить в классе-потомке. В Python все методы являются виртуальными и, соответственно, доступными для перезагрузки.

Повторное использование кода существует в двух видах:

1. Наследование (отношение «является»);
2. Модель включения/делегации (отношение «имеет»).

МОДУЛЬНОСТЬ

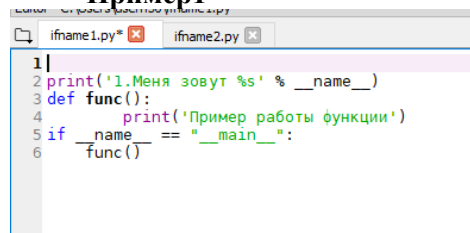
Отношение: «является»

Во многих модулях можно использовать конструкцию вида:

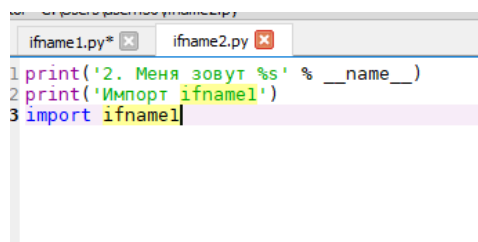
```
if __name__ == "__main__":  
    func()
```

Её основное назначение — разделение кода, который будет выполняться при вызове кода как модуля (при импортировании его в другой скрипт) — и при запуске самого модуля, как отдельного файла.

Пример1



```
1|  
2| print('1. Меня зовут %s' % __name__)  
3| def func():  
4|     print('Пример работы функции')  
5| if __name__ == "__main__":  
6|     func()
```



```
1| print('2. Меня зовут %s' % __name__)  
2| print('Импорт ifname1')  
3| import ifname1
```

```
In [3]: runfile('C:/Users/usern30
Reloaded modules: ifname1
2. Меня зовут __main__
Импорт ifname1
1.Меня зовут ifname1

In [4]: runfile('C:/Users/usern30
Reloaded modules: ifname1
1.Меня зовут __main__
Пример работы функции

In [5]:
```

Пример 2

```
ifname1.py x ifname2.py x ifname3.py x ifname4.py x
if __name__ == "__main__":
    print('Я работаю как независимая программа с названием = %s' % __name__)
else:
    print('Я работаю как импортированный модуль с именем = %s' % __name__)
```

```
ifname1.py x ifname2.py x ifname3.py x ifname4.py x
print('2. Меня зовут %s' % __name__)
print('Импорт ifname3')
import ifname3
```

```
In [7]: runfile('C:/Users/usern30/ifname3.py', wdir='C:/Users/usern30')
Я работаю как импортированный модуль с именем = ifname3
Reloaded modules: ifname3
Я работаю как независимая программа с названием = __main__

In [8]: runfile('C:/Users/usern30/ifname4.py', wdir='C:/Users/usern30')
2. Меня зовут __main__
Импорт ifname3
Я работаю как импортированный модуль с именем = ifname3
```

Пример 3

Предположим, существует **телефонная компания**, хранящая данные о своих клиентах. Для простого учета используется класс **Customer** (Клиент), содержащий атрибуты:

- поле name: имя клиента (чтение/запись);
- свойство balance: баланс счета клиента (только чтение!);
- метод record_payment(): выполняет пополнение баланса;
- метод record_call(): выполняет обработку звонка клиента в зависимости от:
 - типа звонка: «городской» (5 руб./мин.) и «мобильный» (1 руб./мин.);
 - количества минут разговора.

Создаем класс **Customer**, представляющий клиента телефонной компании:

```
class Customer:
    """Клиент телефонной компании"""

    def __init__(self, name, balance=0):
        self.name = name
        self._balance = balance

    def __str__(self):
        if self.balance<0:
            return "Клиент \"{ }\", у Вас отрицательный баланс: { } руб.\nНеобходимо пополнить!".format(self.name, self.balance)
        else:
            return "Клиент \"{ }\", баланс: { } руб.".format(self.name, self.balance)
```

```

@property
def balance(self):
    """Вернуть баланс клиента.
    Свойство 'balance' доступно только для чтения!"""
    return self._balance

def record_payment(self, refill):
    """Пополнить баланс клиента на 'refill' руб"""
    assert refill > 0, "Сумма пополнения должна быть > 0!"
    """ assert - булевы выражения, которые проверяют, является ли условие истинным ( True )"""
    self._balance += refill

def record_call(self, call_type, minutes):
    """Списать стоимость звонка с баланса клиента.
    Параметры:
    - call_type (str): тип звонка:
        "Г": городской;
        "М": мобильный;
    - minutes (float): количество минут разговора. """
    # При работе с БД, эти значения могут читаться из нее
    if call_type == "Г":
        self._balance -= minutes * 5
    elif call_type == "М":
        self._balance -= minutes * 1

if __name__ == "__main__":
    Klient1 = Customer("Смычник Софья Аркадьевна")
    Klient2 = Customer("Мирошниченко Никита Валентинович", 100)
    Klient1.record_call("Г", 120)
    Klient1.record_call("М", 10)
    Klient2.record_call("М", 10)
    Klient1.record_payment(155) # Пополнили телефон на 155 руб.
    print(Klient1)
    print(Klient2)

    Клиент "Смычник Софья Аркадьевна", у Вас отрицательный баланс: -455 руб.
    Необходимо пополнить!
    Клиент "Мирошниченко Никита Валентинович", баланс: 90 руб.

```

Расширим возможности компании, добавив наличие у клиента тарифного плана, в каждом из которых есть тип звонка («городской» и «мобильный»):

Повременный: «городской» (5 руб./мин.) и «мобильный» (1 руб./мин.);

После10МинутВ2РазаДешевле: после 10 минут звонка на городской номер каждая вторая минута бесплатно; в остальном как **Повременный**;

ПлатиМеньшеДо5Минут: до 5 минут разговора в 2 раза дешевле тарифа **Повременный**, после - в 2 раза дороже.

В следующем примере будет использование принципов наследования и полиморфизма. Унаследовав класс Customer, можно изменить только метод расчета (за счёт полиморфизма - переопределить работу метода), а далее использовать новый класс, если клиент использует новый тарифный план.

Добавление тарифного плана через наследование класса:

**

```
class Customer:
    """Клиент телефонной компании"""

    def __init__(self, name, balance=0):
        self.name = name
        self._balance = balance

    def __str__(self):
        if self.balance < 0:
            return "Клиент \'{ }\' ,у Вас отрицательный баланс: { } руб.\nНеобходимо пополнить!".format(self.name, self.balance)
        else:
            return "Клиент \'{ }\' , баланс: { } руб.".format(self.name, self.balance)

    @property
    def balance(self):
        """Вернуть баланс клиента.
        Свойство 'balance' доступно только для чтения!"""
        return self._balance

    def record_payment(self, refill):
        """Пополнить баланс клиента на 'refill' руб"""
        assert refill > 0, "Сумма пополнения должна быть > 0!"
        """ assert - булевы выражения, которые проверяют, является ли условие истинным ( True )"""
        self._balance += refill

    def record_call(self, call_type, minutes):
        """Списать стоимость звонка с баланса клиента.
        Параметры:
        - call_type (str): тип звонка:
            "Г": городской;
            "М": мобильный;
        - minutes (float): количество минут разговора. """
        # При работе с БД, эти значения могут читаться из нее
        if call_type == "Г":
            self._balance -= minutes * 5
        elif call_type == "М":
            self._balance -= minutes * 1

class CustomerFree2(Customer):
    """Клиент телефонной компании (потомок Customer).
    После 10 минут звонка на городской номер каждая вторая минута бесплатно;
    в остальном как "Повременный".
    Все атрибуты, что предоставляет Customer, доступны и здесь, достаточно
    изменить только те, которые должны работать по-другому."""

    def record_call(self, call_type, minutes):
        # Данный метод переопределяет соответствующий метод родителя

        # Определяем количество бесплатных минут
        if call_type == "Г" and minutes > 10:
            bonus_minutes = (minutes - 10) // 2
        else:
            bonus_minutes = 0
```

```

# Вызываем родительский метод расчета
super().record_call(call_type, minutes - bonus_minutes)

class CustomerTwices(Customer):
    """Клиент телефонной компании (потомок Customer).
    До 5 минут разговора в 2 раза дешевле тарифа "Повременный",
    после - в 2 раза дороже.
    Все атрибуты, что предоставляет Customer, доступны и здесь, достаточно
    изменить только те, которые должны работать по-другому."""

    def record_call(self, call_type, minutes):
        # Данный метод переопределяет соответствующий метод родителя
        limit = 5
        if minutes > limit:
            cheap_minutes = limit
            expensive_minutes = minutes - limit
        else:
            cheap_minutes = minutes
            expensive_minutes = 0

        # Вызываем родительский метод расчета
        super().record_call(call_type, cheap_minutes / 2 +
                             expensive_minutes * 2)

if __name__ == "__main__":

    Klient1 = Customer("Мирошниченко Никита Валентинович", 100)
    # Klient2, Klient3 и Klient4 теперь экземпляры других классов
    Klient2 = CustomerFree2("Смычник Софья Аркадьевна", 100)
    Klient3 = CustomerFree2("Риттер Родион Александрович", 100)
    Klient4 = CustomerTwices("Куренков Сергей Александрович", 1000)

    Klient1.record_call("Г", 40)
    Klient2.record_call("Г", 10)
    Klient3.record_call("М", 50)
    Klient4.record_call("Г", 200)

    print(Klient1)
    print(Klient2)
    print(Klient3)
    print(Klient4)

    Клиент "Мирошниченко Никита Валентинович", у Вас отрицательный баланс: -100 руб.
    Необходимо пополнить!
    Клиент "Смычник Софья Аркадьевна", баланс: 50 руб.
    Клиент "Риттер Родион Александрович", баланс: 50 руб.
    Клиент "Куренков Сергей Александрович", у Вас отрицательный баланс: -962.5 руб.
    Необходимо пополнить!

```

За счет свойств наследования и полиморфизма при необходимости добавления нового тарифа в данном случае достаточно объявить новый класс (например, VipClient, в т.ч. унаследовать его не только от Customer, но и других классов) и реализовать только необходимые методы относительно родителя.

С дочерним классом возможно проводить следующие операции:

- добавлять собственные поля и методы; при этом данные изменения не коснутся родительского класса;
- заменять/дополнять реализацию уже существующих родительских атрибутов, переопределяя их.

Отношение: «имеет»

Другим возможным вариантом решения системы тарифов (и более логичным при решении данной проблемы) будет реализация тарификации по модели «включения/делегации».

Пусть существует 2 общих класса:

- **CallPlan** (Тариф):

- поле name: наименование тарифа (чтение/запись);
- метод record_call(): выполняет обработку звонка клиента в зависимости от:
 - типа звонка: «городской» (5 руб./мин.) и «мобильный» (1 руб./мин.);
 - количества минут разговора.

Наследники CallPlan (прочие тарифы) будут иметь изменять необходимые атрибуты.

Customer (Клиент): хранит информацию о пользователе и ссылку на тариф:

- поле name: имя клиента (чтение/запись);
- поле call_plan: тариф - ссылка на объекта класса Тариф (чтение/запись);
- свойство balance: баланс счета клиента (только чтение);
- метод record_payment(): выполняет пополнение баланса;
- метод record_call(): выполняет обработку звонка клиента на основании тарифа call_plan.

Алгоритм подсчета стоимости звонка для класса Customer изменится следующим образом:

- расчет стоимости делегируется объекту call_plan;
- результат (стоимость звонка по тарифу) вычитается из баланса клиента.

1) Включение/делегация классов в Python: класс **CallPlan**:

```
class CallPlan:
```

```
    """Абстрактный класс для всех тарифных планов."""
```

```
    def __init__(self):
```

```
        self.name = "Абстрактный тариф"
```

```
    def record_call(self, call_type, minutes):
```

```
        """Списать стоимость звонка с баланса клиента.
```

```
        Параметры:
```

```
        - call_type (str): тип звонка:
```

```
            "Г": городской;
```

```
            "М": мобильный;
```

```
        - minutes (float): количество минут разговора.
```

```
        """
```

```
        # Делегируем расчет стоимости отдельному методу
```

```
        # Так, наследнику достаточно будет переопределить каждый из них,
```

```
        # не меняя общую логику ниже
```

```
        if call_type == "Г":
```

```
            return self.record_call_g(minutes)
```

```
        elif call_type == "М":
```

```
            return self.record_call_m(minutes)
```

```
        else:
```

```
            return 0
```

```
    def record_call_g(self, minutes):
```

```
        """Вернуть стоимость звонка на городской номер для 'minutes' минут."""
```

```
        raise NotImplementedError # Должны реализовать дочерние классы
```

```
    def record_call_m(self, minutes):
```

```
"""Вернуть стоимость звонка на мобильный номер для 'minutes' минут."""  
raise NotImplementedError # Должны реализовать дочерние классы
```

```
class CallPlanSimple(CallPlan):
```

```
    def __init__(self):  
        self.name = "Повременный"  
  
    def record_call_g(self, minutes):  
        return minutes * 5  
  
    def record_call_m(self, minutes):  
        return minutes * 1
```

```
class CallPlanFree2(CallPlanSimple):
```

```
    def __init__(self):  
        self.name = "После_10 в2 раза дешевле"  
  
    def record_call_g(self, minutes):  
        if minutes > 10:  
            bonus_minutes = (minutes - 10) // 2  
        else:  
            bonus_minutes = 0  
  
        # Вызываем родительский метод расчета  
        return super().record_call_g(minutes - bonus_minutes)
```

```
class CallPlanTwice(CallPlanSimple):
```

```
    def __init__(self):  
        self.name = "Плати_меньше до 5 минут"  
  
    def record_call(self, call_type, minutes):  
        limit = 5  
        if minutes > limit:  
            cheap_minutes = limit  
            expensive_minutes = minutes - limit  
        else:  
            cheap_minutes = minutes  
            expensive_minutes = 0  
  
        # Вызываем родительский метод расчета  
        return super().record_call(call_type, cheap_minutes / 2 +  
                                   expensive_minutes * 2)
```

2)Включение/делегация классов в Python: класс **Customer**:

```
from call_plan import CallPlanSimple
```

```
class Customer:
```

```
    """Клиент телефонной компании."""  
  
    def __init__(self, name, balance=0, call_plan=None):
```

```

        self.name = name
        self._balance = balance
        self.call_plan = call_plan
        # Если тарифный план не был указан, используем CallPlanSimple()
        if self.call_plan is None:
            self.call_plan = CallPlanSimple()

    def __str__(self):
        if self.balance < 0:
            return "Клиент \"{ }\", у Вас отрицательный баланс: { } руб., Тариф: \"{ }\" \nНеобходимо пополнить!".format(self.name, self.balance, self.call_plan.name)
        else:
            return "Клиент \"{ }\", баланс: { } руб., Тариф: \"{ }\"".format(self.name, self.balance, self.call_plan.name)

    @property
    def balance(self):
        """Вернуть баланс клиента.

        Свойство 'balance' доступно только для чтения:
        давать доступ на изменение его напрямую было бы неправильно."""
        return self._balance

    def record_payment(self, refill):
        """Пополнить баланс клиента на 'refill' руб"""
        assert refill > 0, "Сумма пополнения должна быть > 0!"
        """ assert - булевы выражения, которые проверяют, является ли условие истинным ( True
)"""
        self._balance += refill

    def record_call(self, call_type, minutes):
        """Списать стоимость звонка с баланса клиента.

        Параметры:
        - call_type (str): тип звонка:
            "Г": городской;
            "М": мобильный;

        - minutes (int): количество минут разговора.
        """
        # Делегируем определение стоимости звонка классу call_plan
        costs = self.call_plan.record_call(call_type, minutes)
        self._balance -= costs3) Включение/делегация классов в Python: основной модуль

from customer import Customer
from call_plan import CallPlanFree2, CallPlanTwice

if __name__ == "__main__":

    k1 = Customer("Мирошниченко Никита Валентинович", 100)

    # 1. Используется тариф по умолчанию
    k1.record_call("Г", 20)
    print(k1) # Клиент " Мирошниченко Никита Валентинович". Баланс: 0 руб. Тариф:
    "Повременный"

```



```

k1.record_payment(100 - k1.balance) # Пополнили телефон до 100 руб.

# 2. Меняем тариф на CallPlanFree2
k1.call_plan = CallPlanFree2()
k1.record_call("Г", 20)
print(k1) # Клиент "Мирошниченко Никита Валентинович". Баланс: 25 руб. Тариф:
"После10В2РазаДешевле"

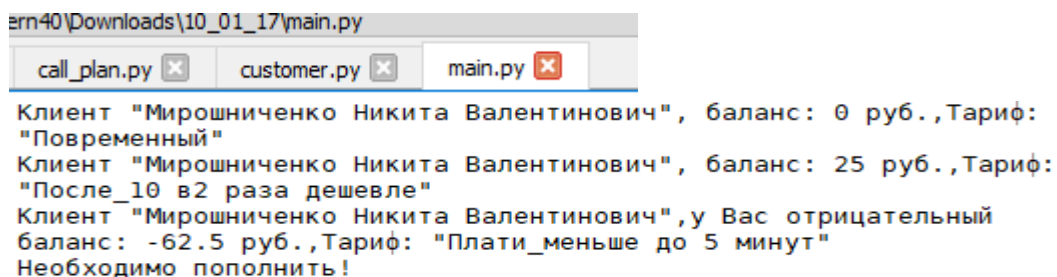
```

```

k1.record_payment(100 - k1.balance) # Пополнили телефон до 100 руб.

# 3. Меняем тариф на CallPlanTwice
k1.call_plan = CallPlanTwice()
k1.record_call("Г", 20)
print(k1) # Мирошниченко Никита Валентинович". Баланс: -62.5 руб. Тариф:
"ПлатиМеньшеДо5Минут"

```



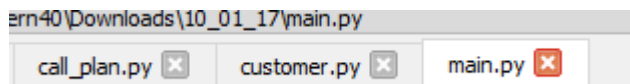
```

Клиент "Мирошниченко Никита Валентинович", баланс: 0 руб.,Тариф:
"Повременный"
Клиент "Мирошниченко Никита Валентинович", баланс: 25 руб.,Тариф:
"После_10 в2 раза дешевле"
Клиент "Мирошниченко Никита Валентинович", у Вас отрицательный
баланс: -62.5 руб.,Тариф: "Плати_меньше до 5 минут"
Необходимо пополнить!

```

Данный пример может быть расширен, например, добавлением журнала звонков, смены тарифа и т.д.

За счёт наследования и полиморфизма также легко осуществляются групповые операции с классами. Далее приведен пример сравнения затрат по тарифам между собой.



```

from customer import Customer
from call_plan import CallPlanSimple, \
    CallPlanFree2ndMinuteAfter10, \
    CallPlanTwiceCheaperFirst5Minutes

if __name__ == "__main__":

    call_plans = (CallPlanSimple(),
                  CallPlanFree2ndMinuteAfter10(),
                  CallPlanTwiceCheaperFirst5Minutes())

    minutes = tuple(range(0, 26, 5)) # 0, 5, 10, 15, 20, 25 мин.

    # Сравним стоимости звонков для тарифов
    for call_type in ("Г", "М"):
        print("{:30}".format(call_type), end="")
        # Заголовок - минуты
        for minute in minutes:
            print("{:>8d} мин.".format(minute), end="")
        print()

    # Подсчет стоимости
    for call_plan in call_plans:

```

```

print("{:30}".format(call_plan.name), end="")
for minute in minutes:
    print("{:>8.2f} руб.".format(call_plan.record_call(call_type, minute)), end="")
print()

print()

```

Г	0 мин.	5 мин.	10 мин.	15 мин.	20 мин.	25 мин.
Повременный	0.00 руб.	25.00 руб.	50.00 руб.	75.00 руб.	100.00 руб.	125.00 руб.
После_10 в2 раза дешевле	0.00 руб.	25.00 руб.	50.00 руб.	65.00 руб.	75.00 руб.	90.00 руб.
Плати_меньше до 5 минут	0.00 руб.	12.50 руб.	62.50 руб.	112.50 руб.	162.50 руб.	212.50 руб.
М	0 мин.	5 мин.	10 мин.	15 мин.	20 мин.	25 мин.
Повременный	0.00 руб.	5.00 руб.	10.00 руб.	15.00 руб.	20.00 руб.	25.00 руб.
После_10 в2 раза дешевле	0.00 руб.	5.00 руб.	10.00 руб.	15.00 руб.	20.00 руб.	25.00 руб.
Плати_меньше до 5 минут	0.00 руб.	2.50 руб.	12.50 руб.	22.50 руб.	32.50 руб.	42.50 руб.

Проектирование иерархии классов и класс object

Объектно-ориентированная парадигма программирования (и, в частности, принцип наследования) подразумевает построение иерархии классов, которая бы в совокупности наиболее эффективным образом решала поставленную задачу.

Одной из наиболее популярных в мире профессиональных нотаций (системы условных обозначений) моделирования в объектно-ориентированном стиле является язык **UML** (англ. Unified Modeling Language). На рисунке 1 приведен пример изображения иерархии классов в данной нотации.

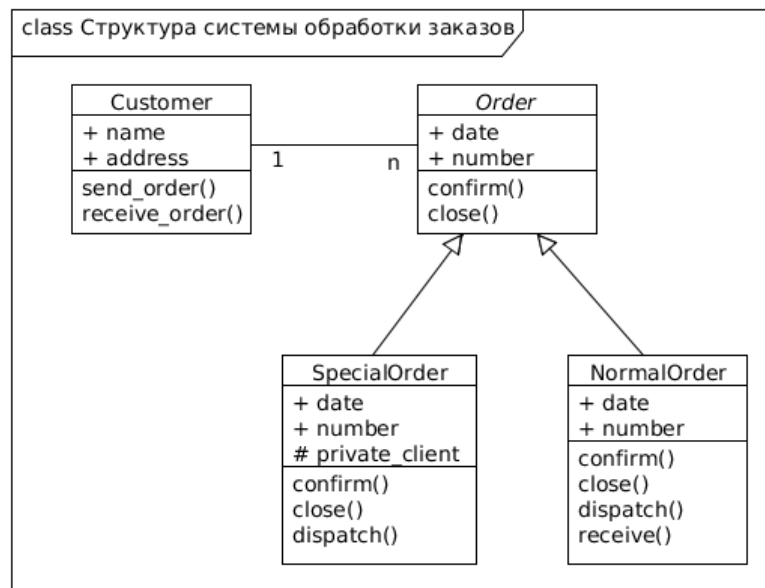


Рисунок 1 – Диаграмма классов UML

Язык UML позволяет детально смоделировать иерархию классов перед реализацией. В то же время, использование UML для малых проектов может оказаться избыточным, и достаточно простой схемы.

В Python все классы наследуются от класса `object`, так, для классов из программы ****** иерархия будет выглядеть следующим образом:

```

object -> Customer -> CustomerFree2ndMinuteAfter10
                    -> CallPlanTwiceCheaperFirst5Minutes

```

Встроенные классы Python также имеют свою иерархию, например, для типа bool она выглядит как:

object -> int -> bool

Множественное наследование

Python поддерживает концепцию множественного наследования, т.е. позволяет создать класс, имеющий нескольких родителей. В данном случае список наследуемых классов перечисляется при объявлении класса:

Пример 1

```
class Student:
    def pass_exam(self):
        print("Я сдал экзамен по ООП!")

class Worker:
    def work(self):
        print("Работаю, учиться не успеваю ...")

class ConferensUser:
    def score(self):
        print("Участвую в конференции!")

class StudentWorkerAnConferensUser(Student, Worker, ConferensUser):
    pass

if __name__ == "__main__":
    s = StudentWorkerAnConferensUser()
    s.pass_exam()
    s.work()
    s.score()

Я сдал экзамен по ООП!
Работаю, учиться не успеваю ...
Участвую в конференции!
```

Пример 2

В этом скрипте, родительский класс Vehicle наследуется двумя дочерними классами — Car и Cycle. Оба дочерних класса будут иметь доступ к vehicle_method() родительского класса.

```
# создаем класс Vehicle
class Vehicle:
    def vehicle_method(self):
        print("Это родительский метод из класса Vehicle")

# создаем класс Car, который наследует Vehicle
class Car(Vehicle):
    def car_method(self):
        print("Это дочерний метод из класса Car")

# создаем класс Cycle, который наследует Vehicle
```

```

class Cycle(Vehicle):
    def cycleMethod(self):
        print("Это дочерний метод из класса Cycle")

if __name__ == "__main__":
    car_a = Car()
    car_a.vehicle_method() # вызов метода родительского класса
    car_b = Cycle()
    car_b.vehicle_method() # вызов метода родительского класса

Это родительский метод из класса Vehicle
Это родительский метод из класса Vehicle

```

Пример 3

В скрипте ниже создали три класса: Camera, Radio, и CellPhone. Классы Camera и Radio наследуются классом CellPhone. Это значит, что класс CellPhone будет иметь доступ к методам классов Camera и Radio.

```

class Camera:
    def camera_method(self):
        print("Это родительский метод из класса Camera")

class Radio:
    def radio_method(self):
        print("Это родительский метод из класса Radio")

class CellPhone(Camera, Radio):
    def cell_phone_method(self):
        print("Это дочерний метод из класса CellPhone")

if __name__ == "__main__":
    cell_phone_a = CellPhone()
    cell_phone_a.camera_method()
    cell_phone_a.radio_method()

Это родительский метод из класса Camera
Это родительский метод из класса Radio

```

Множественное наследование часто **критикуется** и зачастую считается признаком неверного анализа и проектирования, поэтому его использование рекомендуется в случае крайней необходимости и оправданности такого решения.

Переопределение метода

Переопределение метода относится к наличию метода с одинаковым названием в дочернем и родительском классах. **Определение метода** отличается в родительском и дочернем классах, но название остается тем же.

```

# создание класса Vehicle
class Vehicle:
    def print_details(self):
        print("Это родительский метод из класса Vehicle")

# создание класса, который наследует Vehicle
class Car(Vehicle):
    def print_details(self):
        print("Это дочерний метод из класса Car")

```

```
# создание класса Cycle, который наследует Vehicle
class Cycle(Vehicle):
    def print_details(self):
        print("Это дочерний метод из класса Cycle")

if __name__ == "__main__":
    car_a = Vehicle()
    car_a.print_details()

    car_b = Car()
    car_b.print_details()

    car_c = Cycle()
    car_c.print_details()

Это родительский метод из класса Vehicle
Это дочерний метод из класса Car
Это дочерний метод из класса Cycle
```

В скрипте выше, классы Cycle и Car наследуют класс Vehicle. Класс Vehicle содержит метод print_details(), который переопределен дочерним классом. Теперь, если вы вызовете метод print_details(), выдача будет зависеть от объекта, через который вызывается метод.

Как вы видите, выдача отличается, к тому же метод print_details() вызывается через производные классы одного и того же базового класса. Однако, так как дочерние классы переопределены методом родительского класса, методы ведут себя по-разному.

Перегрузка операторов

Перегрузка операторов позволяет экземплярам классов участвовать в обычных операциях. Чтобы перегрузить оператор, необходимо в классе определить метод со специальным названием. В результате, для выполнения действия соответствующего данной операции будет вызываться этот метод. Перегрузка математических операторов производится с помощью следующих методов:

Выражение	Операция	Метод
x+y	сложение	x.add(y)
y+x	сложение (экземпляр класса справа)	x.radd(y)
x+=y	сложение и присваивание	x.iadd(y)
x-y	вычитание	x.sub(y)
y-x	вычитание (экземпляр класса справа)	x.rsub(y)
x-=y	вычитание и присваивание	x.isub(y)
x*y	умножение	x.mul(y)
y*x	умножение (экземпляр класса справа):	x.rmul(y)
x*=y	умножение и присваивание	x.imul(y)
x@y	матричное умножение	x.matmul(y)
y@x	матричное умножение (экземпляр класса справа)	x.rmatmul(y)
x@=y	Матричное умножение и присваивание	x.imatmul(y)
x/y	деление	x.truediv(y)
y/x	деление (экземпляр класса справа):	x.rtruediv(y)
x/=y	деление и присваивание	x.itruediv(y)
x//y	деление с округлением вниз	x.floordiv(y)
y/=x	деление с округлением вниз (экз. класса справа):	x.rfloordiv(y)
x/=y	деление с округлением вниз и присваивание	x.ifloordiv(y)
x%y	остаток от деления	x.mod(y)
y%x	остаток от деления (экземпляр класса справа):	x.rmod(y)
x%=y	остаток от деления и присваивание	x.imod(y)
x**y	возведение в степень	x.pow(y)
y**x	возведение в степень (экземпляр класса справа):	x.rpow(y)
x**=y	возведение в степень и присваивание	x.ipow(y)
-x	унарный минус	x.neg()
+x	унарный плюс	x.pos()
abs(x)	абсолютное значение	x.abs()

Пример перегрузки математических операторов:

```
class MyClass:
    def __init__(self,y):
        self.y = y
    def __add__(self,x):
        print( "Экземпляр слева")
        return self.y + x
    def __radd__(self,x):
        print( "Экземпляр справа")
        return self.y + x
    def __iadd__(self,x):
        print( "Сложение с присваиванием")
        self.y += x
        return self
    def __sub__(self,x):
        print( "Вычитание")
        self.y-= x
        return self

c = MyClass(50)

print( c + 10 ) # Вывод: Экземпляр слева 60
print( 20 + c ) # Вывод: Экземпляр справа 70
c += 30 # Вывод: Сложение с присваиванием
print(c.y)
c-=20 #Вывод: Вычитание
print( c.y )

Экземпляр слева
60
Экземпляр справа
70
Сложение с присваиванием
80
Вычитание
60
```

Перегрузка операторов сравнения производится с помощью следующих методов:

Выражение	Операция	Метод
<code>x==y</code>	равно	<code>x.__eq__(y)</code>
<code>x!=y</code>	не равно	<code>x.__ne__(y)</code>
<code>x<y</code>	меньше	<code>x.__lt__(y)</code>
<code>x>y</code>	больше	<code>x.__gt__(y)</code>
<code>y<=x</code>	меньше или равно	<code>x.__le__(y)</code>
<code>x>=y</code>	больше или равно	<code>x.__ge__(y)</code>
<code>x in y</code>	проверка на вхождение	<code>x.__contains__(y)</code>

Пример перегрузки операторов сравнения:

```
class MyClass:
    def __init__(self):
        self.x = 45 # 60
        self.arr = [1, 2, 3, 4, 5]
    def __eq__(self, y): # Перегрузка оператора ==
        return self.x == y
    def __contains__(self, y): # Перегрузка оператора in
        return y in self.arr
    def __ne__(self, y): # Перегрузка оператора !=
        return self.x != y

c = MyClass ()
print(c.x)
print(c.arr)
print("Равно" if c == 50 else "Не равно")
print("Равно" if c == 51 else "Не равно")
print("Есть" if 8 in c else "Нет")
print("Нет" if c != 60 else "Да")

45
[1, 2, 3, 4, 5]
Не равно
Не равно
Нет
Нет

60
[1, 2, 3, 4, 5]
Не равно
Не равно
Нет
Да
```

Пример ООП программы

Класс ForeignPassport является производным от класса Passport. Метод PrintInfo существует в обоих классах. PassportList представляет собой список, содержащий объекты обоих классов. Вызов метода PrintInfo для каждого элемента списка демонстрирует его полиморфное поведение.

```
class Passport():
    def __init__(self, first_name, last_name, country, date_of_birth,
numb_of_pasport):
        self.first_name = first_name
        self.last_name = last_name
        self.date_of_birth = date_of_birth
        self.country = country
        self.numb_of_pasport = numb_of_pasport
    def PrintInfo(self):
        print("\nFullname: ",self.first_name, " ",self.last_name)
        print("Date of birth: ",self.date_of_birth)
        print("Country: ",self.country)
        print("Passport number: ",self.numb_of_pasport)

class ForeignPassport(Passport):
    def __init__(self, first_name, last_name, country, date_of_birth,
numb_of_pasport,visa):
        super().__init__(first_name, last_name, country, date_of_birth,
numb_of_pasport)
        self.visa = visa

    def PrintInfo(self):
        super().PrintInfo()
        print("Visa: ",self.visa)
```

```

PassportList=[]
request = ForeignPassport('Иван', 'Петров', 'Россия', '12.03.2004',
'123456789','USA')
PassportList.append(request)
request = Passport('Петр', 'Иванов', 'Россия', '12.03.1967', '45001432')
PassportList.append(request)
request = ForeignPassport('Александр', 'Каменев', 'Россия', '01.03.1990',
'21435688','Germany')
PassportList.append(request)
for a in PassportList:
    a.PrintInfo()

    Fullname: Иван Петров
    Date of birth: 12.03.2004
    County: Россия
    Passport number: 123456789
    Visa: USA

    Fullname: Петр Иванов
    Date of birth: 12.03.1967
    County: Россия
    Passport number: 45001432

    Fullname: Александр Каменев
    Date of birth: 01.03.1990
    County: Россия
    Passport number: 21435688
    Visa: Germany

```

Класс как структура данных

В ряде случаев бывает полезным иметь структуру, похожую на структуру из языка Си (или запись из Паскаля), позволяющую логически сгруппировать данные. Для этого можно использовать словарь или класс с пустой реализацией.

Использование класса в Python как структуры в Си

```

class FamousPeople:
    pass

if __name__ == "__main__":

    a = FamousPeople()

    # При присвоении значения несуществующему полю класса Python дополняет класс
    # Синтаксис похож на словарь, где ключ указывается не в скобках [], а через точку
    a.name = "Водянова Наталья Михайловна"
    a.birthday = "28/02/1982"
    a.position = "Модель"
    a.height = 1.76
    a.weigh = 48

    print('ФИО: {} Дата рождения: {} Статус: {} Рост: {} см. Вес: {} кг.'.format(a.name,
a.birthday, a.position, a.height, a.weigh))

```

```

ФИО: Водянова Наталья Михайловна Дата рождения: 28/02/1982 Ствтус: Модель Рост: 1.76 см. Вес: 48 кг.

```

Преимущества и недостатки ООП

Объектно-ориентированное программирование как другие парадигмы имеет свои преимущества и недостатки.

Преимущества

1. Улучшение производительности разработки ПО.

ООП способствует модульности, расширяемости и повторному использованию кода (за счет аспекта наследования и полиморфизма).

2. Улучшение сопровождения ПО.

Благодаря модульности, расширяемости и повторному использованию кода его легче поддерживать.

3. Ускорение разработки.

Повторное использование кода позволяет выполнять разработку быстрее, в т.ч. в смежных проектах.

4. Снижение стоимости разработки.

Повторное использование кода также позволяет снизить издержки и сосредоточиться на объектно-ориентированном анализе и проектировании, снижающем общую стоимость ПО.

5. Повышение качества ПО.

Ускорение разработки и снижение затрат позволяет уделить больше времени и ресурсов на тестирование ПО.

Недостатки

1. Крутая кривая обучения.

Мышление в ООП-стиле требует определенных навыков, а переход от императивного стиля на взаимодействие объектов может потребовать времени.

2. Большой объем кода.

Как правило, ООП приводит появлению большего количества кода, нежели в императивном программировании.

3. Медленные программы.

ООП-программы чаще выполняются медленнее, т.к. содержат больше кода для выполнения.

4. ООП не подходит для всех случаев.

Ряд задач могут лучше решаться в императивном, логическом или функциональном стиле, где использование ООП не даст выигрыша.

Задания для самостоятельной работы:

Задание 1 (общее)

1. Класс Vector3D. Экземляр класса задается тройкой координат в трехмерном пространстве (x,y,z).

Обязательно должны быть реализованы методы:

- приведение вектора к строке с выводом координат (метод `__str__`),
- сложение векторов оператором `+` (метод `__add__`),
- вычитание векторов оператором `-` (метод `__sub__`),
- скалярное произведение оператором `*` (метод `__mul__`),
- умножение и деление на скаляр операторами `*` и `/` (метод `__mul__` и `__truediv__`),
- векторное произведение оператором `@` (метод `__matmul__`)

Задание 2

Для класса, созданного в предыдущем задании (ООП_2) создать класс-потомок с полями, указанными в индивидуальном задании.

Реализовать в классе-потомке методы:

- конструктор;
- функцию обработки данных, указанную в индивидуальном задании;
- функцию формирования строки информации об объекте.

Создать проект для демонстрации работы: ввод и вывод информации об объектах, классе-родителе и классе-потомке.

Варианты индивидуальных заданий:

№ вар.	Класс-родитель и его поля	Класс-потомок и его поля (поля класса-родителя выделены курсивом)	Функция-метод обработки данных объекта класса-потомка
1	Дата (три числа): день, месяц, год	Список друзей: ФИО, телефон, <i>дата рождения</i>	Количество дней до дня очередного рождения
2	Работник: фамилия, оклад, год поступления на работу	Работники предприятия: <i>фамилия, оклад, год поступления на работу,</i> год рождения	Определить, сколько лет нужно работать работнику до 60 лет, а если ему больше 60, то сколько лет он работает после 60 лет.
3	Книга: название, количество страниц, цена	Библиотека: <i>название, количество страниц, цена,</i> скидка в процентах	Стоимость книги с учетом скидки.
4	Время (три числа): часы, минуты, секунды	Расписание движения поездов: номер поезда, направление, <i>время отправления</i>	Количество минут до отправления поезда с указанным номером и введенное время
5	Товар: наименование, цена, год выпуска	Фирменный товар: <i>наименование, цена, год выпуска, дата поступления товара</i>	Количество дней после года выпуска товара до текущего дня.
6	Дата (три числа): день, месяц, год	Работник: ФИО, <i>дата поступления на предприятие</i>	Количество лет работы на предприятии
7	Книга: название, автор, год издания	Книжный магазин: <i>название, автор, год издания, цена</i>	Уменьшить стоимость книги на 20%, если книге больше 5 лет.

№ вар.	Класс-родитель и его поля	Класс-потомок и его поля (поля класса-родителя выделены курсивом)	Функция-метод обработки данных объекта класса-потомка
8	Работник: фамилия, оклад, дата рождения	Работники фирмы: <i>фамилия, оклад, дата рождения</i> , должность	Увеличить оклад работникам с должностью программист на 20%.
9	Время (три числа): часы, минуты, секунды	Абонент мобильной связи: фамилия, оператор, <i>текущее время</i>	Определить, является ли время льготным для абонента (время от 0 до 8 часов)
10	Четыре целых числа: a, b, c, d	Пять чисел: <i>четыре целых числа (a,b,c,d)</i> и число x	Вычислить сумму квадратов разности каждого из четырех чисел и числа x.
11	Работник: фамилия, должность, оклад	Работники предприятия: <i>фамилия, должность, оклад</i> , рейтинг (в 100-бальной системе)	Увеличить оклад работников на 20%, если их рейтинг от 60 до 75 баллов, на 40%, если их рейтинг от 75 до 90 баллов, на 60%, если их рейтинг от 90 до 100 баллов.
12	Книга: название, количество страниц, цена	Изданная книга: <i>название, количество страниц, цена</i> , дата выхода в печать	Возраст книги в месяцах
13	Дата (три числа): день, месяц, год	Лекарство: наименование, <i>дата выпуска</i> , фирма	Сколько прошло дней от изготовления лекарства
14	Товар: наименование, цена в рублях, изготовитель	Товар: <i>наименование, цена в рублях, изготовитель</i> , год выпуска, скидка в процентах	Изменить стоимость товара с учетом скидки для товаров, изготовленных фирмой более двух лет назад.
15	Время (три числа): часы, минуты, секунды	Расписание занятий: дисциплина, <i>время начала</i> , аудитория	Какая дисциплина по расписанию начинается в указанное время
16	Правильная дробь: числитель, знаменатель	Смешанная дробь: целая часть, <i>числитель и знаменатель</i>	Представить смешанную дробь в виде десятичного вещественного числа.
17	Комната: длина, ширина, высота (в метрах)	Помещения для офисов: <i>длина, ширина, высота комнат</i> , количество комнат, расход краски на 1 м ²	Определить количество краски, необходимое для покраски стен и потолка помещений офиса (в каждой комнате одно окно размером 2×15 м).
18	Комплексное число: действительная (a1) и мнимая (b1) части числа	Два комплексных числа: <i>действительная (a1) и мнимая (b1) части первого числа</i> ; действительная (a2) и мнимая (b2) части второго числа	Вычислить произведение двух комплексных чисел.

№ вар.	Класс-родитель и его поля	Класс-потомок и его поля (поля класса-родителя выделены курсивом)	Функция-метод обработки данных объекта класса-потомка
19	Координаты изображения прямоугольника: x1, y1, x2, y2	Изображение конверта (прямоугольник с линиями диагоналей): <i>координаты прямоугольника</i> , цвет линий	Площадь верхнего (наддиагонального) треугольника в пикселях
20	Параллелепипед: длины сторон	Металлический брус: ширина, высота, длина, удельный вес	Определить вес металлического бруса.

Задание 3

Написать программу согласно заданию.

Во всех классах описать необходимые конструкторы, при помощи которых будут создаваться объекты классов. Параметры создаваемых объектов задавать константами или вводить с клавиатуры (по желанию студента) и передавать в конструкторы объектов в виде параметров. Вывод информации должен осуществляться на консоль.

№ вар.	Задача
1	<p>Создать класс Автомобиль со свойствами: Название, Максимальная скорость (в км/ч). Определить 2 виртуальных метода: метод «Стоимость» – стоимость автомобиля, рассчитываемую по формуле. Максимальная скорость * 100 и метод «Обновление модели», увеличивающий максимальную скорость на 10. Определить также метод «Информация», который возвращает строку, содержащую информацию об объекте: Название, Максимальную скорость и Стоимость.</p> <p>Создать также класс наследник Представительский автомобиль, в котором переопределить методы: метод «Стоимость» возвращает число, равное. Максимальная скорость * 250, а метод «Обновление модели» увеличивает скорость на 5 км/ч.</p> <p>В главной программе создать объект класса Автомобиль с максимальной скоростью 140 км/ч и класса Представительский автомобиль с максимальной скоростью 160 км/ч. Вывести на экран информацию об автомобилях. Обновить модели автомобилей и снова вывести информацию о них.</p>
2	<p>Создать класс Треугольник, заданный значениями длин трех сторон (a, b, c), с методами «Периметр» и «Площадь». Определить также метод «Информация», который возвращает строку, содержащую информацию о треугольнике: длины сторон, периметр и площадь.</p> <p>Создать также класс наследник Четырехугольник, с дополнительными параметрами – длиной четвертой стороны (d) и длинами диагоналей (e, f) и переопределить методы «Периметр» (сумма всех сторон) и «Площадь». Площадь вычислять по следующей формуле</p> $S = \sqrt{\frac{4e^2f^2 - (b^2 + d^2 - a^2 - c^2)^2}{16}}$ <p>формуле</p> <p>В главной программе создать объект класса Треугольник и объект класса Четырехугольник и вывести информацию о них. Для упрощения проверки рекомендуется в качестве конкретного объекта класса четырехугольник взять квадрат.</p>
3	<p>Создать класс Компьютер со свойствами: Частота процессора (в МГц), количество ядер, объем памяти (в МБ), объем жесткого диска (в ГБ). Определить два виртуальных метода: «Стоимость», возвращающую примерную расчетную стоимость компьютера, рассчитываемую по формуле: Частота процессора * количество ядер / 100 + количество памяти / 80 + объем жесткого диска / 20 и логический метод «Пригодность», возвращающий истину (true), если частота процессора не менее 2000</p>

№ вар.	Задача
	<p>МГц, количество ядер не менее 2, объем памяти не менее 2048 МБ, и объем жесткого диска не менее 320 Гб. Определить также метод «Информация», который возвращает строку, содержащую информацию о компьютере: частоту процессора, количество ядер, объем памяти, объем жесткого диска, стоимость и пригодность для наших нужд. Создать также класс наследник Ноутбук, с дополнительным свойством. Продолжительность автономной работы (в минутах) и переопределить методы: метод «Стоимость» возвращает число, равное стоимости обычного компьютера + количество минут автономной работы / 10, а метод «Пригодность» возвращает истину, когда и ноутбук пригоден как обычный компьютер, и Продолжительность автономной работы не меньше 60 минут.</p> <p>В главной программе создать обычный компьютер и ноутбук и вывести информацию о них.</p>
4	<p>Создать класс Прямоугольник, заданный значениями длин двух сторон (a и b), с виртуальными методами «Периметр» и «Площадь», возвращающими периметр и площадь соответственно, а также виртуальный метод «Увеличить в два раза», увеличивающий в два раза каждую из сторон. Определить также метод «Информация», который возвращает строку, содержащую информацию об треугольнике: длины сторон, периметр и площадь.</p> <p>Создать также класс наследник Прямоугольник со скругленными углами, с дополнительным параметром радиус скругления (r). Для него переопределить: Периметр по формуле $p = 8 \cdot r + 2 \cdot \pi \cdot r$, где p – периметр обычного прямоугольника с теми же сторонами, а Площадь по формуле $S = 4 \cdot r^2 + \pi \cdot r^2$, где S – площадь обычного прямоугольника. Также переопределить метод «Увеличить в два раза» так, чтобы он также увеличивал в два раза радиус скругления (по-прежнему увеличивая стороны в два раза).</p> <p>В главной программе создать обычный прямоугольник и прямоугольник со скругленными углами и вывести информацию о них. После этого увеличить оба прямоугольника в два раза и выдать обновленную информацию.</p>
5	<p>Создать класс Фотоаппарат со свойствами: Модель, Оптическое увеличение (Zoom, вещественное число от 1 до 35) и материал корпуса (металл либо пластик). Определить виртуальный метод: метод «Стоимость» – возвращает число – стоимость фотоаппарата (в \$), рассчитываемую по формуле $(Zoom+2) \cdot 10$, если корпус пластиковый и $(Zoom+2) \cdot 15$, если материал металлический. Определить также метод «Информация», который возвращает строку, содержащую информацию об объекте: Модель, Zoom и Стоимость. Также определить логический метод «Дорогой», который будет возвращать истину (true), если стоимость фотоаппарата больше 200\$.</p> <p>Создать также класс наследник Цифровой фотоаппарат, в котором будет дополнительный целый параметр – количество мегапикселей и переопределить метод «Стоимость», который будет возвращать число, равное стоимости обычного фотоаппарата умножить на количество мегапикселей, а также определить новый метод «Обновление модели», который увеличивает количество мегапикселей на 2.</p> <p>В главной программе создать объект класса Фотоаппарат с 4-х кратным оптическим увеличением (Zoom=4) и пластиковым корпусом, а также Цифровой фотоаппарат с металлическим корпусом, 8-ю мегапикселями и 3-кратным оптическим увеличением. Вывести на экран информацию о фотоаппаратах и о том, являются ли они дорогими. Обновить модели цифрового фотоаппарата и снова вывести информацию о нем.</p>
6	<p>Создать класс Студент со свойствами: ФИО, факультет, курс, минимальная оценка по экзаменам за последнюю сессию (по 5-ти бальной системе). Определить виртуальные методы: «Перевести на следующий курс», увеличивающий курс на 1, если минимальная оценка не менее 3, иначе не делающий ничего, а также «Стипендия», возвращающий стипендию: 0 руб., если минимальная оценка не выше 3, 1500 руб.,</p>

№ вар.	Задача
	<p>если минимальная оценка равна 4 и 2000 руб., если минимальная оценка равна 5. Определить также метод «Информация», который возвращает строку, содержащую информацию о студенте: ФИО, факультет, курс, минимальная оценка по экзаменам и начисленную стипендию.</p> <p>Создать также класс наследник Студент-контрактник, в котором будет дополнительный логический параметр – уплачен ли контракт и переопределены методы «Перевести на следующий курс», увеличивающий курс на 1, если минимальная оценка не менее 3 и за контракт уплачено, а также «Стипендия» возвращающий всегда 0 руб.</p> <p>В главной программе создать объект класса Студент и 2 объекта класса Студент-контрактник (один из которых оплатил за контракт, а другой нет). Выдать информацию о студентах, затем применить к ним метод «Перевести на следующий курс» и снова выдать информацию о них.</p>
7	<p>Создать класс Круг заданный своим радиусом (r), с виртуальным методом «Площадь», возвращающим площадь круга, а также виртуальный метод «Увеличить» с одним вещественным параметром – во сколько раз увеличить, увеличивающий радиус в заданное число раз. Определить также метод «Информация», который возвращает строку, содержащую информацию о круге: радиус и площадь.</p> <p>Создать также класс наследник Кольцо, с дополнительным параметром — внутренним радиусом (r_{in}), при этом унаследованный от родителя радиус будет обозначать внешний радиус. Переопределить метод «Площадь», как разницу между площадью внешнего круга минус площадь внутреннего круга. Также доопределить метод «Увеличить», чтобы он увеличивал также и внутренний радиус.</p> <p>В главной программе создать обычный круг и кольцо и вывести информацию о них. После этого увеличить оба объекта в полтора раза и выдать обновленную информацию.</p>
8	<p>Создать класс Табуретка со свойствами: Высота (h, в см), Качество изделия (низкое, среднее, высокое). Определить два виртуальных метода: «количество древесины», которое требует табуретка, по формуле $4 \cdot h + 12$, если качество низкое, и $5 \cdot h + 14$, если качество среднее или высокое, а также «стоимость», равная $d \cdot 2$, для низкого качества, $d \cdot 3$, для среднего качества, $d \cdot 4$, для высокого качества, где d – количество древесины, которое требует данный объект. Определить также метод «Информация», который возвращает строку, содержащую информацию об объекте: Высоту, качество материала, количество древесины и стоимость.</p> <p>Создать также класс наследник Стул с дополнительным свойством: высота спинки (h_2, в см), и переопределить метод «количество древесины», по формуле $d + 2h_2 + 5$, где d – количество древесины, которые требует табуретка с такими же параметрами. Метод «стоимость» не переопределять.</p> <p>В главной программе создать экземпляры классов Табуретка и Стул, и напечатать информацию в таком виде: «табуретка» + информация о табуретке и «стул» + информация о стуле.</p>
9	<p>Создать класс Фильм со свойствами: Название, Режиссер, длительность (в минутах), количество актеров. Определить виртуальный метод: «Стоимость», возвращающую примерную расчетную стоимость фильма (в тыс. \$), рассчитываемую по формуле $\text{длительность} \cdot 20 + \text{количество актеров} \cdot 30$, но если режиссер = «Стивен Спилберг» или «Джеймс Кэмерон», то стоимость в два раза выше (по сравнению с вышеуказанной формулой). Определить также метод «Информация», который возвращает строку, содержащую информацию о фильме: Название, режиссера, длительность, количество актеров и стоимость.</p> <p>Создать также класс наследник Мультфильм, в котором переопределить метод «Стоимость» по формуле $\text{длительность} \cdot 25 + \text{количество актеров} \cdot 10$ (вне зависимости</p>

№ вар.	Задача
	от режиссера). В главной программе создать 2 фильма с режиссерами: «Стивен Спилберг» и «Ежи Гофман», а также мультфильм и вывести информацию о них.
10	Создать класс Самолет со свойствами: Марка, Модель, Максимальная скорость (в км/ч), Максимальная высота (в метрах). Определить виртуальный метод «Стоимость» – стоимость самолета, рассчитываемую по формуле $\text{Максимальная скорость} * 1000 + \text{Максимальная высота} * 100$ Определить также метод «Информация», который возвращает строку, содержащую информацию об объекте: Марка, Модель, Максимальную скорость, Максимальную высоту и Стоимость. Создать также класс наследник Бомбардировщик , в котором переопределить метод «Стоимость», который вернет удвоенную стоимость относительно формулы для класса Самолет. Также создать класс Истребитель – наследник класса Самолет, для которого переопределить метод «Стоимость» как утроенную стоимость, относительно формулы стоимости для Самолета. В главной программе создать объект класса Самолет, класса Бомбардировщик, класса Истребитель. Вывести на экран информацию о самолетах.

Задание 4 (общее)

Программно промоделировать стрельбу по мишени группой человек.

Каждый человек имеет свое имя, возраст (в годах) и стаж обучения стрельбе (в годах).

Люди делятся на новичков, опытных и ветеранов (потомки класса человек). Для каждого человека определите полиморфный метод. «Стрелять» без параметров, возвращающих логическое значение (попал – true, не попал – false). Попадание определяется случайным образом, причем для новичка вероятность попасть равна $0,01 * \text{стаж обучения}$; для опытного = $0,05 * \text{стаж обучения}$ стрельбе; для ветерана = $0,9 - 0,01 * \text{возраст}$.

Люди стреляют по очереди, начиная с первого, пока кто-то не попадет в мишень. Стрельба прекращается после того, как кто-то попал или все выстрелили по одному разу. После каждого выстрела нужно выводить на экран всю информацию о стреляющем и результат стрельбы.

В главной программе создайте массив из 7 людей в таком порядке: новичок, опытный, ветеран, опытный, новичок и произведите стрельбу с выводом ее результатов.