Наследование (inheritance)

- Механизм языка программирования, ассоциируется с ООП
- Дает возможность определить новый класс как модифицированную версию уже существующего класса
- Концепция наследования в Python обсуждается на примере классов, представляющих игральные карты

Класс карт

- Колода из 52 карт, каждая принадлежит одной из 4 *мастей* и одному из 13 *рангов*
- Масти – пики, червы, бубны и трефы (приведены в порядке убывания при игре в бридж)
- Ранги – Туз, 2, 3, 4, 5, 6, 7, 8, 9, 10, Валет, Дама и Король
- Задача – определить класс для представления игральной карты
- Атрибуты объектов: ранг и масть (*rank* и *suit*)
- Использование целых чисел для кодирования рангов и мастей

Кодирование рангов и мастей

- Масти:

Пики → 3

Червы → 2

Бубны → 1

Трефы → 0

- Ранги:

Числовые → 2, 3, 4, 5, 6, 7, 8, 9, 10

Туз → 1 (разновидность покера)

Валет → 11

Дама → 12

Король → 13

Класс *Card*

```
# Определяет обычную игральную карту
class Card:
   def __init__(self, suit=0, rank=2):
      self.suit = suit
      self.rank = rank
```

- Использование класса

```
>>> cDiamondsQueen = Card(1, 12)
```

Отображение карт

```
class Card:
    ...
    suitNames = ['Трефы', 'Бубны', 'Червы', 'Пики']
    rankNames = [None, 'Туз', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'Валет', 'Дама', 'Король']
    def __str__(self):
        return '%s масти %s' % (Card.rankNames[self.rank], Card.suitNames[self.suit])
```

- suitNames, rankNames – *атрибуты класса*
- suit, rank – *атрибуты экземпляра*

```
>>> card1 = Card(2, 11)
>>> print(card1)
Валет масти Червы
```

Сравнение карт

- Считаем, что масть более важна, поэтому все пики превосходят все бубны и так далее

```
class Card:
  …
  def __lt__(self, other):
    t1 = self.suit, self.rank
    t2 = other.suit, other.rank
    return t1 < t2
```

Колоды

```
class Deck:
    def __init__(self):
        self.cards = []
        for suit in range(4):
            for rank in range(1, 14):
                card = Card(suit, rank)
                self.cards.append(card)
```

Выдача колоды

```
class Deck:
  …
  def __str__(self):
    res = []
    for card in self.cards:
      res.append(str(card))
    return '\n'.join(res)
```

Результат выдачи

```
>>> deck = Deck()
>>> print(deck)
Туз масти Трефы
2 масти Трефы
3 масти Трефы
...
10 масти Пики
Валет масти Пики
Дама масти Пики
Король масти Пики
```

Манипулирование картами

- Удаление из колоды

```
class Deck:
  …
  # Удаление карты из колоды
  def popCard(self):
    return self.cards.pop()
  # Добавление карты в колоду
  def addCard(self, card):
    self.cards.append(card)
  # Перемешивание карт
  def shuffle(self):
    random.shuffle(self.cards)
```

Наследование классов

- Позволяет определить новый класс как модифицированную версию уже существующего класса

```
# Определяет игральные карты в руке
class Hand(Deck):
  def __init__(self, label=''):
    self.cards = []
    self.label = label
```

- Использование

```
>>> hand = Hand('новая рука')
>>> print(hand.cards)
[]
>>> print(hand.label)
'новая рука'
```

Унаследованные методы

```
>>> hand = Hand('новая рука')
>>> deck = Deck()
>>> card = deck.popCard()
>>> hand.addCard(card)
>>> print(hand)
King of Spades
```

Раздача карт «из руки»

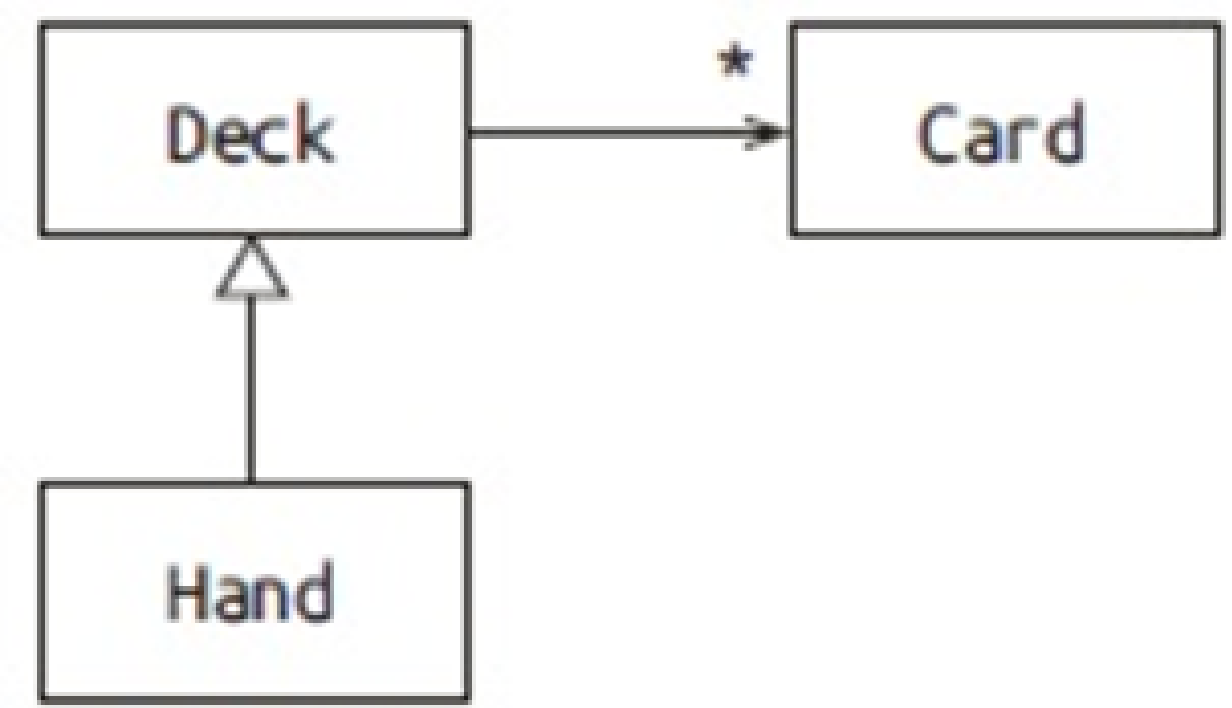
```
class Deck:
  …
  def moveCards(self, hand, num):
    for i in range(num):
      hand.addCard(self.popCard())
```

- Метод *moveCards*() принимает два аргумента: объект Hand и количество карт для раздачи
- Изменяет как self, так и hand
- В некоторых играх карты перемещаются из одной руки в другую или обратно в колоду
- Для каждой из этих операций можно использовать метод *moveCards*(): *self* может быть и колодой, и рукой, а *hand* также может вести себя как объект *Deck*

Отношения между классами

- Объекты класса могут содержать ссылки на объекты другого класса (*HAS-A*)
- Класс может наследоваться от другого (*IS-A*)
- Класс может *зависеть* от другого: объекты класса принимают объекты другого класса в качестве параметров или используют объекты во втором классе как часть вычисления (*dependency*)
- *UML*-диаграмма классов – это графическое представление отношений между классами

UML-диаграмма



Отладка классов

```
# Возвращает класс, предоставляющий определение метода
def findDefiningClass(obj, methName):
  for ty in type(obj).mro():
    if methName in ty.__dict__:
      return ty
```

- Пример использования

```
>>> hand = Hand()
>>> print(findDefiningClass(hand, 'shuffle'))
<class 'Card.Deck'>
```

- Таким образом, метод *shuffle*() для данной «руки» – это тот, который определен в классе *Deck*
- Метод *mro*() получает список объектов классов, в которых будет выполняться поиск методов.
- “MRO” означает «порядок разрешения метода» (method resolution order)

Словарь терминов

- Кодировать. Представить один набор значений, используя другой набор значений, определив между ними соответствие.
- Атрибуты класса. Атрибуты, связанные непосредственно с классом. Определяются внутри описания класса, но вне любого метода.
- Атрибуты объекта (экземпляра). Атрибуты, связанные с экземпляром класса.
- Наследование. Возможность определить новый класс – модифицированную версию ранее определенного класса.
- Родительский класс. Класс, от которого наследуется данный класс.
- Дочерний класс. Новый класс, созданный путем наследования от данного существующего класса. Называется также *подклассом*.
- Сокрытие информации. Принцип, согласно которому предоставляемый классом интерфейс не должен зависеть от его реализации, в частности, от способа представления его атрибутов.

Словарь терминов (продолжение)

- Отношения *IS-A*. Связь между дочерним классом и его родительским классом.
- Отношения *HAS-A*. Связь между двумя классами, когда экземпляры одного класса содержат ссылки на экземпляры другого.
- Зависимость. Связь между двумя классами, когда экземпляры одного класса используют экземпляры другого класса, но не хранят их как атрибуты.
- Диаграмма классов. Диаграмма, которая отражает классы в программе и отношения между ними.
- Множественность. Обозначение на диаграмме классов, которое отражает для отношения *HAS-A*, сколько ссылок существует на экземпляры другого класса.