Working with Classes in Python

About me

Trayan Iliev

– CEO of **IPT – Intellectual Products & Technologies**
http://www.iproduct.org

– Oracle® certified programmer 15+ Y

– end-to-end reactive fullstack apps with Java, ES6+, TypeScript, Angular, React and Vue.js

– 12+ years IT trainer: Spring, Java EE, Node.js, Express, GraphQL, SOA, REST, DDD & Reactive Microservices

– Voxxed Days, jPrime, Java2Days, jProfessionals, BGOUG, BGJUG, DEV.BG speaker

– Organizer RoboLearn hackathons and IoT enthusiast

Where to Find The Code and Materials?

https://github.com/iproduct/intro-python

Object Oriented Programming – OOP (Alan Key 1993)

- Всичко е някакъв обект
- Обработка на данни – чрез обмяна на съобщения между обекти
- Всеки обект има своя памет (състояние), която включва други обекти.
- Всеки обект е екземпляр (instance) на клас
- Класът е хранилище на поведение (реализирано чрез методи, които задават действия). Всички екземпляри на даден клас могат да изпълняват едни и същи действия.
- Всички класове са организирани в йерархично дърво, което представя йерархия на наследяване на свойства между класовете
- При проектирането на класовете и обектите се спазват принципите на абстракция, капсулация, модулност, йерархичност (наследяване, композиция) и полиморфизъм

OOP Main Concepts

- Всеки клас включва данни (атрибути) и функции (методи). Понякога говорим за атрибути данни и атрибути методи (функции).
- Всеки обект е представител (instance) на клас (от един клас могат да се създадат много обекти)
- Всеки вграден тип данни е клас
- Всяка константа (елемент) от тип данни е обект
- Класовете и типовете данни на свой ред са обекти от типа данни (класа type или друг мета-клас)

OOP Main Concepts - II

- Всеки обект в езика Python има собствена памет, която се съхранява с помощта на структура от данни от тип речник
- Всеки елемент от този речник представя отделен атрибут на обекта – всеки атрибут на свой ред е обект
- Всеки атрибут се представя в речника с двойка **name : val**, където **name** е името на атрибута, а **val** е неговата стойност
- Има два различни вида атрибути: за данни и за поведение (методи)
- Всеки обект има поведение, наследено от класа, което се реализира чрез методи (функции) дефинирани в класа.
- Всеки обект в езика Python има уникален идентификатор

Classes and Objects

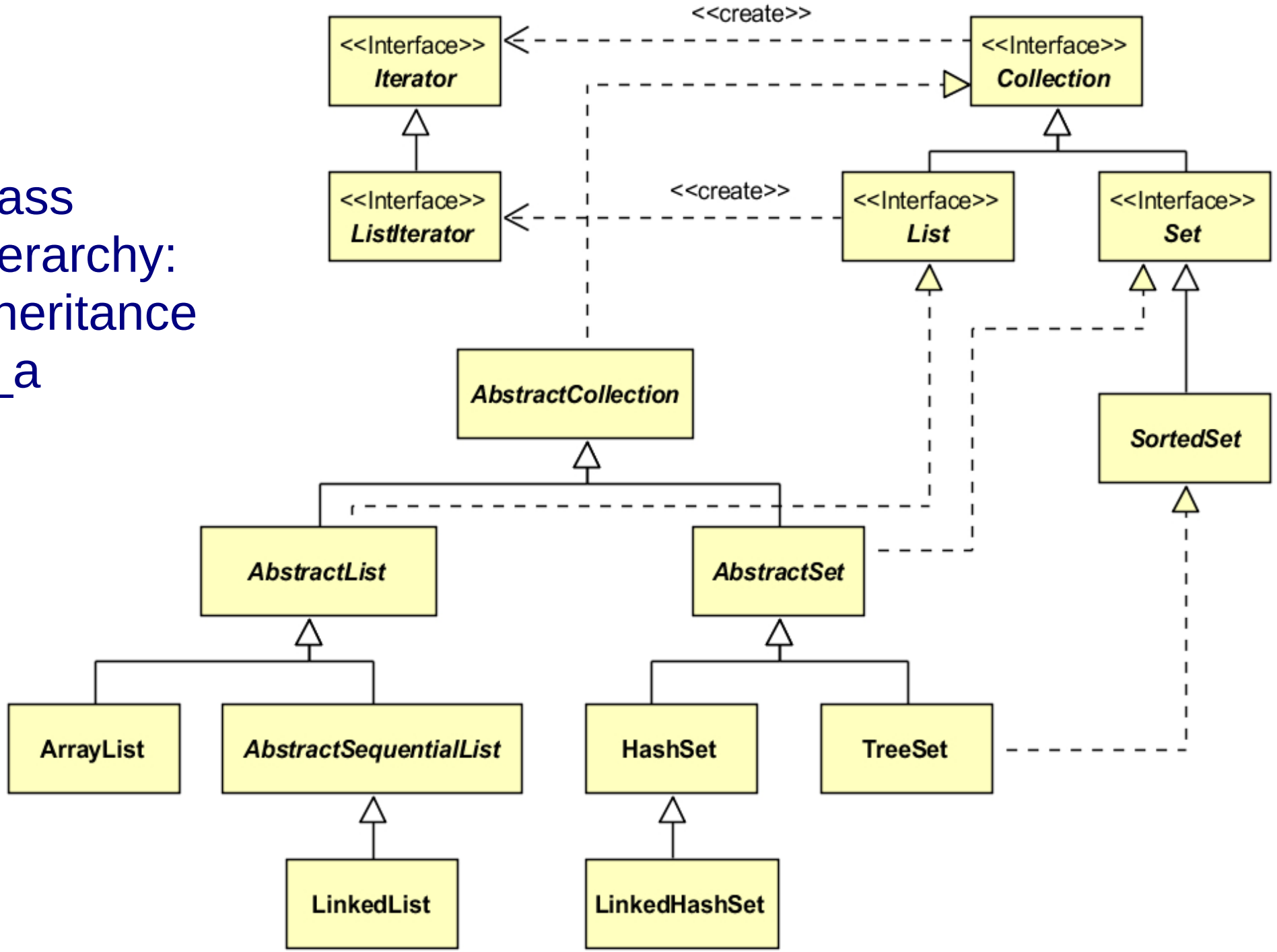
Class – describes common features for a set of objects: structure, behavior and possible links to objects of other classes = **objects type**

- structure = attributes, properties, member variables
- behavior = methods, operations, member functions, messages
- relations between classes: association, inheritance, aggregation, composition – modeled as attributes (references to objects from the connected class)

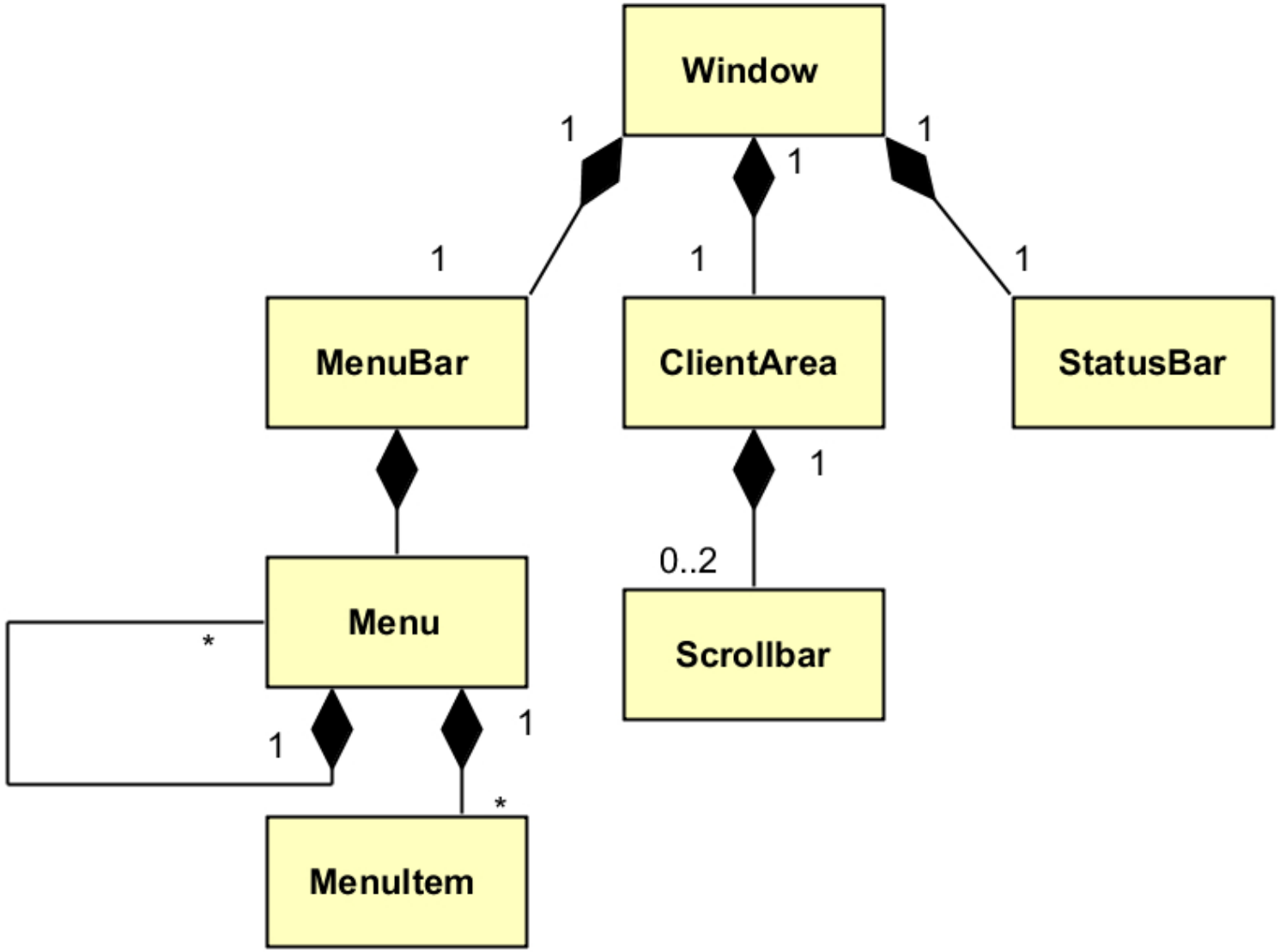
Objects are instances of the class, which in addition have:

- own state
- unique identifier = reference pointing towards object

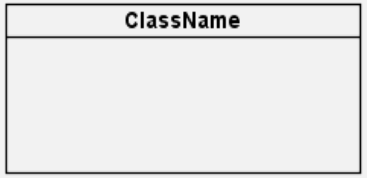
Class Hierarchy: Inheritance is_a

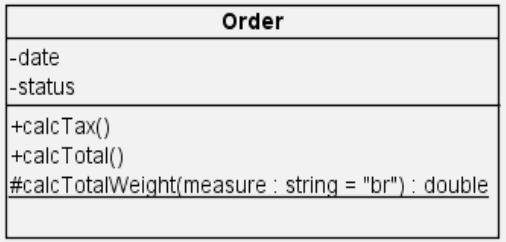


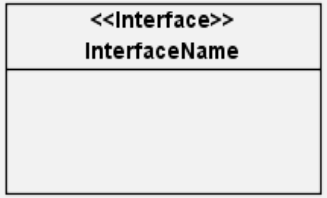
Object Hierarchy: Composition, has_a

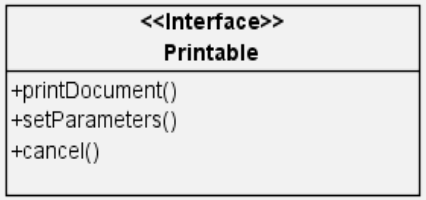


Elements of Class Diagrams





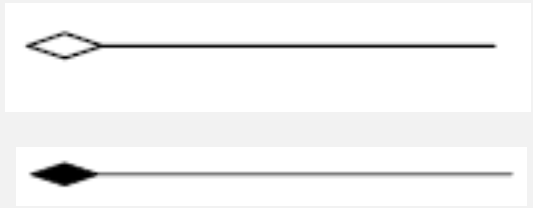




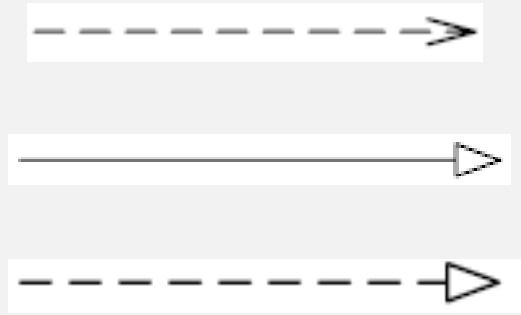
Types of connections:

- Association

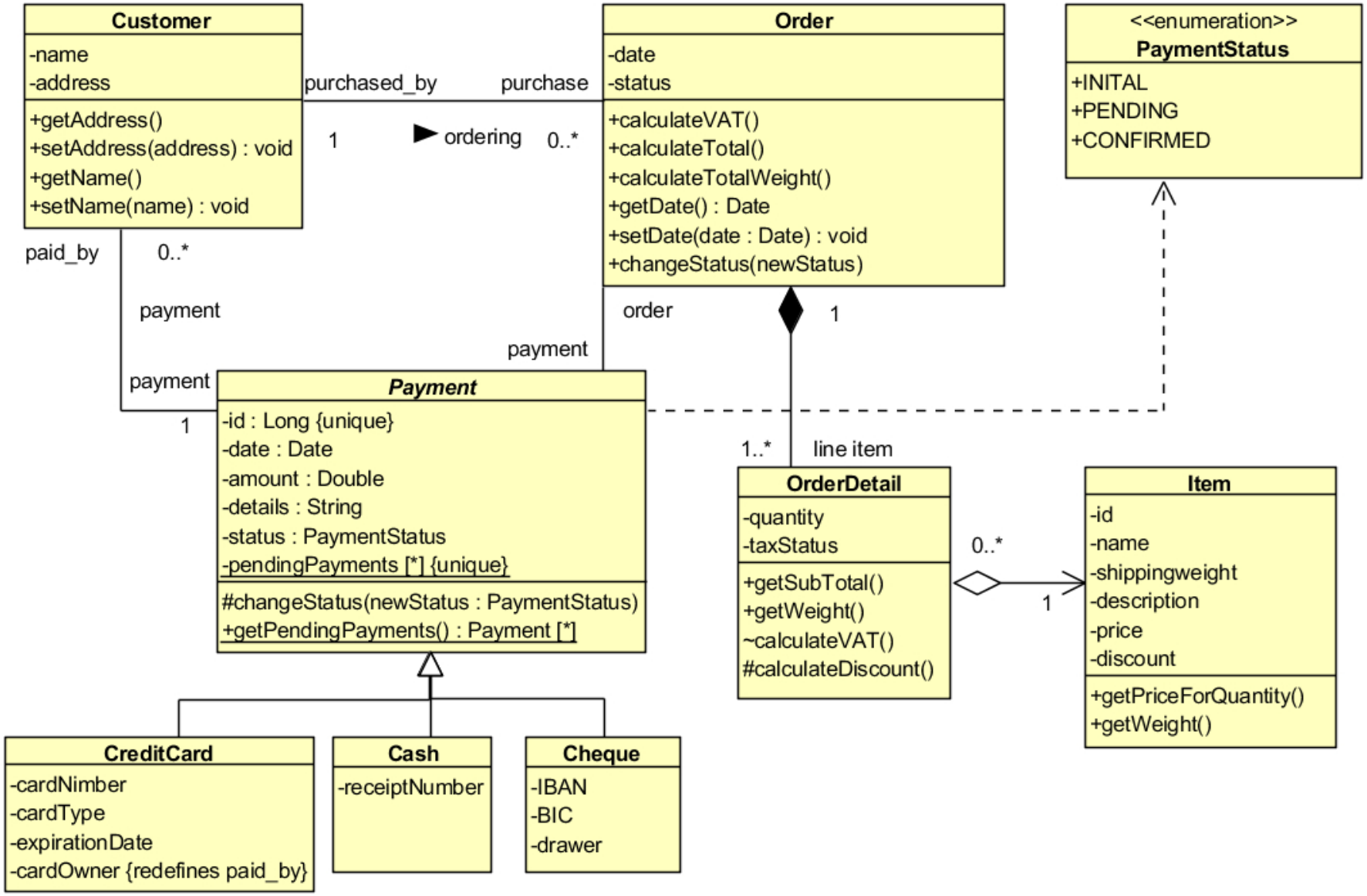
- aggregation
- composition



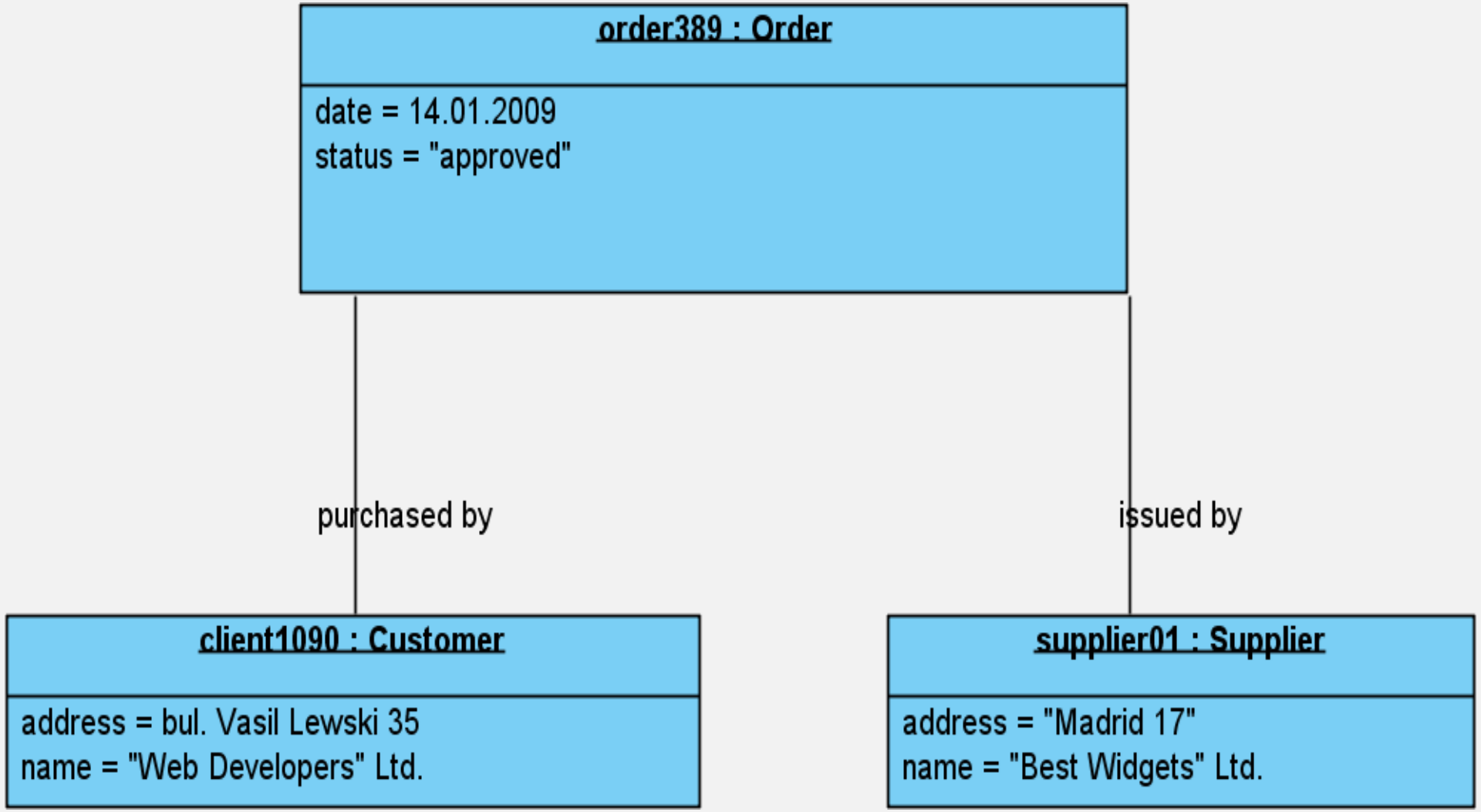
- dependence
- generalization
- realization



Class Diagram



Object Diagram



Code (Component) Reuse and Design Patterns

- Advantages of code reuse
- Ways of implementation:
  - Objects composition – patterns like composite, singleton, decorator, mixin, etc.
  - Inheritance of classes (object types) – features/patterns like dynamic polymorphism, prototype, template method, strategy, etc.

Classes in Python

```
class student:
        """Клас представящ атрибутите за студент"""
        def __init__(self,n,a):
                self.full_name = n
                self.age = a
        def get_age(self):
                return self.age


b = student("Bob", 21)
```

__init__ connstructor

- Методът (функцията) **__init__** може да има произволен брой аргументи
- Както за всяка функция, аргументите могат да бъдат зададени със стойности по подразбиране, правейки ги по този начин незадължителни
- Първият аргумент **self** в дефиницията на **__init__** е задължителен и специален – това е обектът който се инициализира от тази функция и представлява новия обект

self

- Въпреки че self трябва да се зададе явно и задължително при дефиниране на метод, може да бъде изпускан при обръщение към метода. Python автоматично го подава вместо нас
- Дефиниране на метод: (в дефиниция на клас)

```
def set_age(self, num):
        self.age = num
```

- Обръщение към метод:

```
>>> x.set_age(23)
```

Object and Class destructors

- **del(<име_обект>)** за явно унищожаване на обект с указаното име
- Деструктор **__del__** който се извиква преди унищожаване на обекта
- **del(<име_клас>)** за явно унищожаване на клас
- Автоматично изтриване на обект когато няма свързана с него променлива
- Класовете не се изтриват автоматично и нямат деструктор

Accessing class attributes

- Нормален достъп до атрибут <attr> от клас / обект <obj>:

<obj>.<attr> # стандартен достъп

- Аналогичен достъп с вградена функция:

getattr(<obj>, '<attr>') # директен достъп с вградена функция

Example

```
>>> f = student(“Bob Smith”, 23)
>>> f.full_name # достъп до атрибут данни
“Bob Smith”
>>> f.get_age() # достъп до метод
23
```

getattr(object_instance, string)

```
>>> f = student('Bob Smith', 23)
>>> getattr(f, 'full_name')
'Bob Smith'
>>> getattr(f, 'get_age')
<method get_age of class studentClass at 010B3C2>
>>> getattr(f, 'get_age')() # вика го
23
>>> getattr(f, 'get_birthday')
# AttributeError – Няма такъв метод!
```

hasattr(object_instance,string)

```
>>> f = student('Bob Smith', 23)
>>> hasattr(f, 'full_name')
True
>>> hasattr(f, 'get_age')
True
>>> hasattr(f, 'get_birthday')
False
```

setattr(object_instance, string, val)

```
>>> f = student('Bob Smith', 23)
>>> setattr(f, 'full_name', 'Rob Jhon')
>>> getattr(f, 'full_name')
'Rob Jhon'
>>> setattr(f, 'birthday', '12_05_1986')
>>> getattr(f, 'birthday')
'12_05_1986'
```

Instance and class attributes

- Атрибути на обект (instance attributes)
- Обекти налични в екземпляр (instance) на клас
- Всеки екземпляр има обект с различна стойност
- По правило се инициализират с метода __init__
- Атрибути на клас (class attributes)
- Налични само в пространството на класа
- Общи са за всички екземпляри
- В някои езици за програмирани се наричат още статични (“static”) променливи
- Полезни за общи константи за клас или за броячи (например колко пъти е извикан клас или метод)

Class attributes

- Всички обекти споделят едно копие на атрибут. Ако някой обект промени стойността му, е за всички
- Class attributes се задават вътре в дефиницията на клас и извън дефинициите на всички методи
- Тъй като има само един такъв за клас и не за обект,
- достъпът до тях е по специален начин - например чрез: **self.__class__.name**

```
class Sample:
x = 23
def increment(self):
        self.__class__.x += 1
```

```
>>> a = Sample()
>>> a.increment()
>>> a.__class__.x
24
```

Using __class__ attribute

- Всички обекти споделят едно копие на атрибут. Ако някой обект промени стойността му, е за всички
- Class attributes се задават вътре в дефиницията на клас и извън дефинициите на всички методи
- Тъй като има само един такъв за клас и не за обект,
- достъпът до тях е по специален начин - например чрез: **self.__class__.name**

```
class sample:
x = 23
def increment(self):
        self.__class__.x += 1
```

Static methods

```
class Sample:
        x = 23

        @staticmethod
        def increment():
                __class__.x += 1
```

```
>>> a = Sample()

>>> sample.increment()

>>> a.x

24
```

Class methods

```
class Sample:
        x = 23

        @classmethod
        def increment(cls):
                cls.x += 1
```

```
>>> a = Sample()

>>> Sample.increment()

>>> a.x

24
```

“Magic” attributes

- Атрибути, чиито имена започват и завършват с по два знака за подчертаване:
- __init__
- __dict__
- __str__
- Какво е особеното за тях
  - Имат вградено действие и поведение
  - Не се извикват директно, а косвено

“Magic” attributes

- При обръщение към вградени функции

```
len([1, 2, 3]) # __len__()
```

- Използване на оператори

```
1 + 1 # __add__()
```

```
5 > 3 # __gt__()
```

- При цикъл for

```
for x in [1, 2, 3]: # __iter__()
```

```
print(x) # __next__()
```

Special methods

__new__ : създава празен обект за клас

__init__ : инициализира атрибути на обект

__cmp__ : как == ще действа за обекти от класа

__len__ : как да се определя len() за обект

__copy__ : как да се направи копие на обект

__getattr__ : за получаване стойност на атрибут

__setattr__ : за присвояване стойност на атрибут

__delattr__ : за изтриване стойност на атрибут

__str__ : за показване аналогично с функцията str

__repr__ : за официално показване с функция repr

Special methods

__doc__ : Низ с документация за класа

__name__ : съдържа името на класа

__module__ : в кой модул е дефиниран клас

__dict__ : речник с пространство на имената за клас

__dict__ : речник с пространство на имена на обект

__slots__ : задава явно имена на атрибути за обект,
като се забранява използването на __dict__

__bases__: редица с имената на родителски класове

__annotations__: речник с анотации на променливи

__class__ : от кой клас е обекта

Special methods example

```
class Point(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def distance(self, other):
        x_diff_sq = (self.x - other.x) ** 2
        y_diff_sq = (self.y - other.y) ** 2
        return (x_diff_sq + y_diff_sq) ** 0.5
    def __str__(self):
        return "<" + self.x.__str__() + "," + self.y.__str__() + ">"
```

Special methods example II

```
if __name__ == '__main__':
    p = Point(4, 3)
    print(p)
    q = Point(0, 0)
    print(q)
    print(p.distance(q))
```

```
<4,3>
<0,0>
5.0
```

Thank's for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD