Luciano Ramalho
luciano@ramalho.org

@pythonprobr

novembro/2013

Objetos Pythonicos

Orientação a objetos e padrões de projeto em Python

Aula 6

- Decoradores de funções e de classes
- Classes Abstratas
- Exemplo de herança múltipla: Django Generic CBV
  - Class-based views

Decoradores de funções

- Informalmente, já vimos alguns:
  - @property, @x.setter, @staticmethod
- Não são uma implementação do padrão de projeto "decorator"
- São funções que recebem a função decorada como argumento e produzem uma nova função que substitui a função decorada
  - Aprofundado em outro curso

Decoradores de métodos

- Usados na definição de métodos em classes
  - @property, @x.setter, @x.deleter: definem métodos getter, setter e deleter para propriedades
  - @classmethod, @staticmethod: definem métodos que não precisam de uma instância para operar
  - @abstractmethod, @abstractproperty: uso em classes abstratas

classmethod x staticmethod

- Métodos estáticos são como funções simples embutidas em uma classe: não recebem argumentos automáticos
- Métodos de classe recebem a classe como argumento automático

```
class Exemplo(object):
    @staticmethod
    def estatico(arg):
        return arg
    @classmethod
    def da_classe(cls, arg):
        return (cls, arg)
```

```
>>> Exemplo.estatico('bar')
'bar'
>>> Exemplo.da_classe('fu')
(<class '__main__.Exemplo'>, 'fu')
```

Exemplo de classmethod

- É conveniente no método **todas** ter acesso à classe para usar os atributos (**naipes**, **valores**) e para instanciar as cartas

```
class Carta(object):
    naipes = 'paus copas espadas ouros'.split()
    valores = 'A 2 3 4 5 6 7 8 9 10 J Q K'.split()
    def __init__(self, valor, naipe):
        self.valor = valor
        self.naipe = naipe
    def __repr__(self):
        return 'Carta(%r, %r)' % (self.valor, self.naipe)

    @classmethod
    def todas(cls):
        return [cls(v, n) for n in cls.naipes
                          for v in cls.valores]
```

Classe abstrata

- Forma tradicional:
  - um ou mais métodos levantam NotImplementedError
    - verificação somente ao invocar método
- Forma moderna (a partir de Python 2.6)
  - classe com metaclasse abc.ABCmeta
  - métodos marcados com @abc.abstractmethod
  - verificação na instanciação

Decoradores de classes

- Novidade do Python 2.6, ainda pouco utilizada na prática
- Exemplo na biblioteca padrão a partir do Python 2.7:
  - **functools.total_ordering** define automaticamente métodos para os operadores de comparação **< > <= >=**

http://docs.python.org/library/functools.html#functools.total_ordering

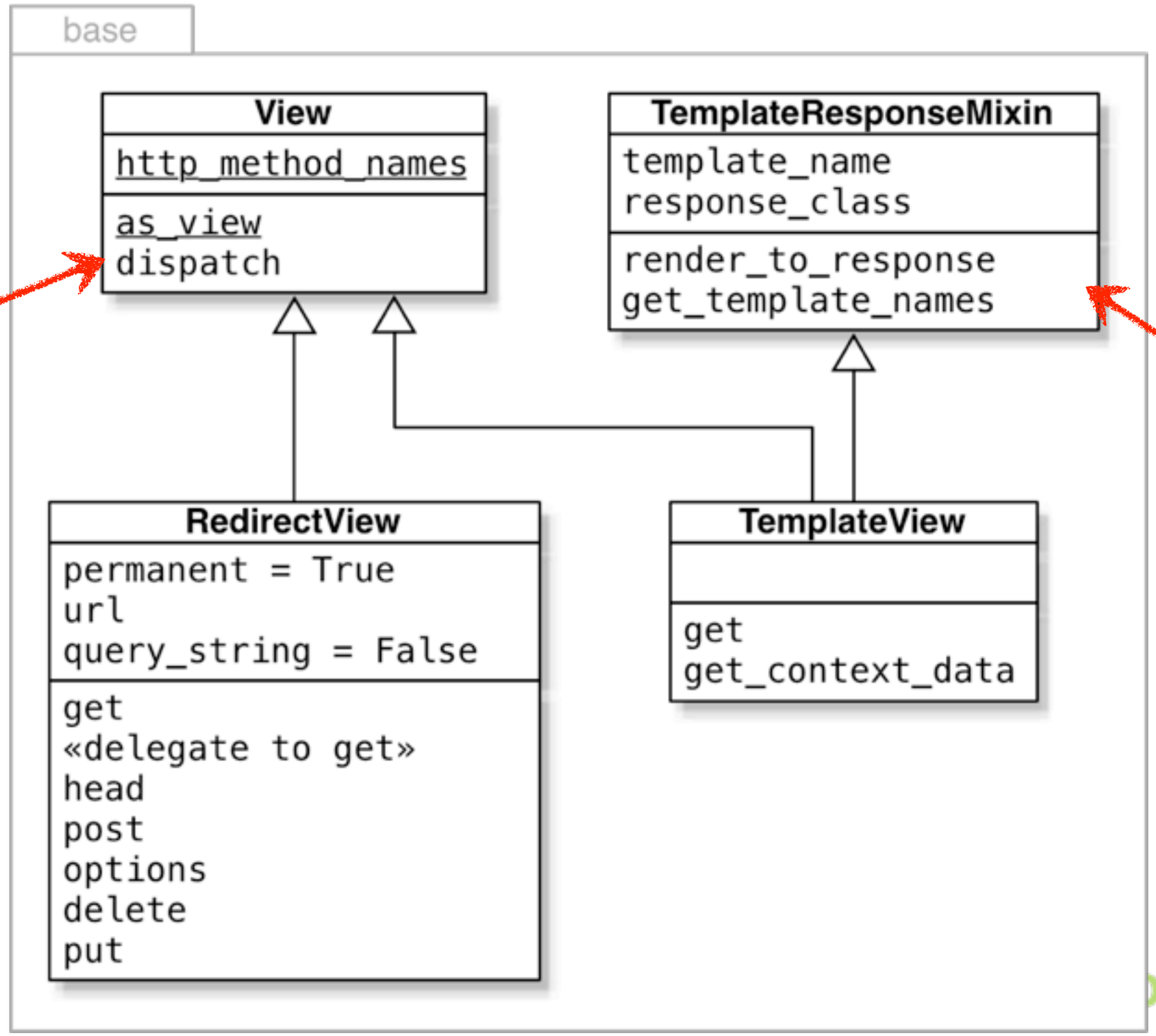
Exemplo de herança múltipla no Django

- Class-based views (CBV): classes para a construção de views, desde o Django 1.3
- Divisão de tarefas para a construção modular de views, diminuindo código repetitivo
- Vamos explorar views básicas e list/detail

API navegável: http://ccbv.co.uk/

Apostila (em desenvolvimento) com diagramas UML:
http://turing.com.br/material/acpython/mod3/django/views1.html

CBV: divisão de tarefas



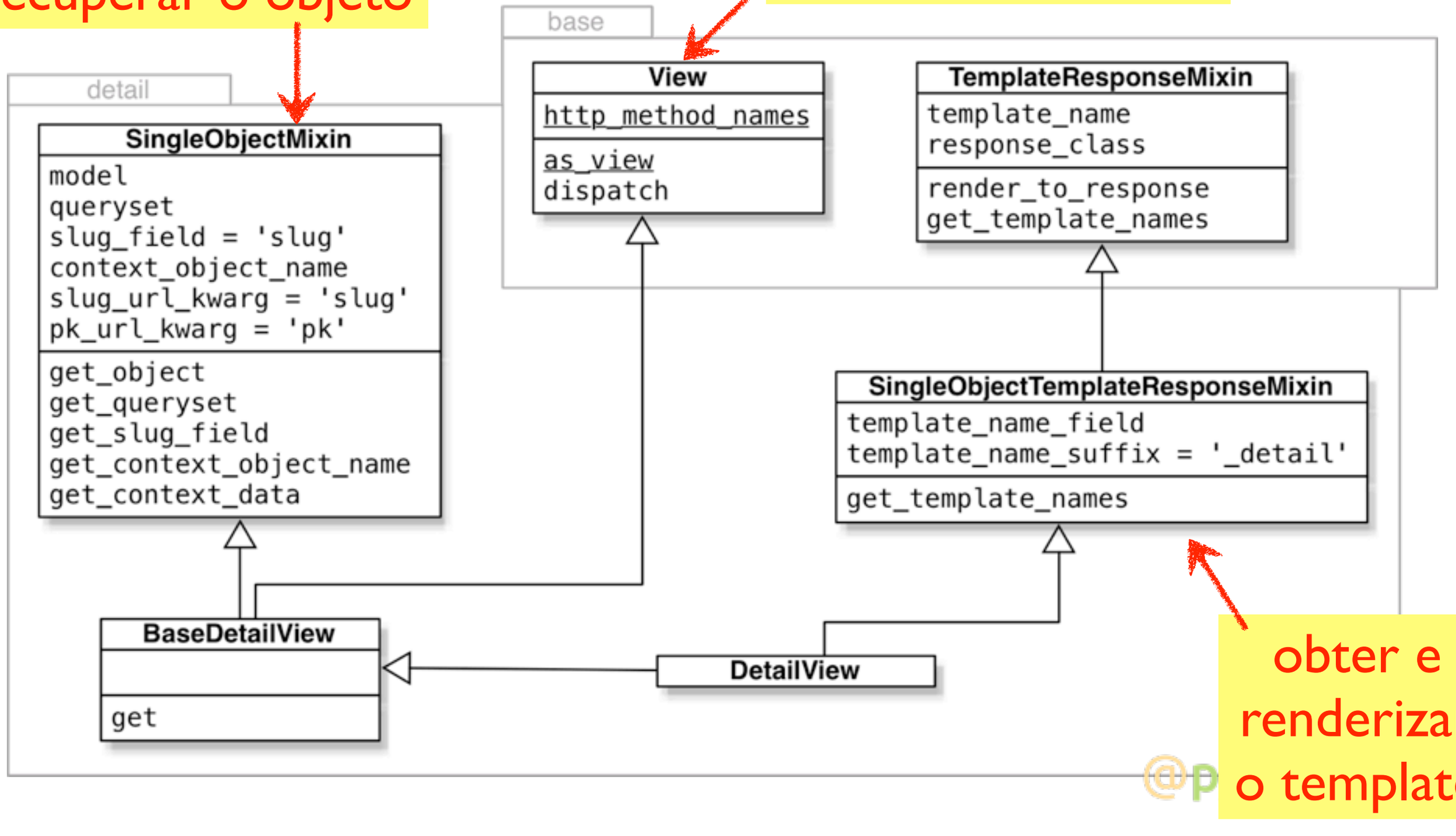
tratar o request e produzir o response

obter e renderizar o template

CBV: views de detalhe

identificar e recuperar o objeto

tratar request/response

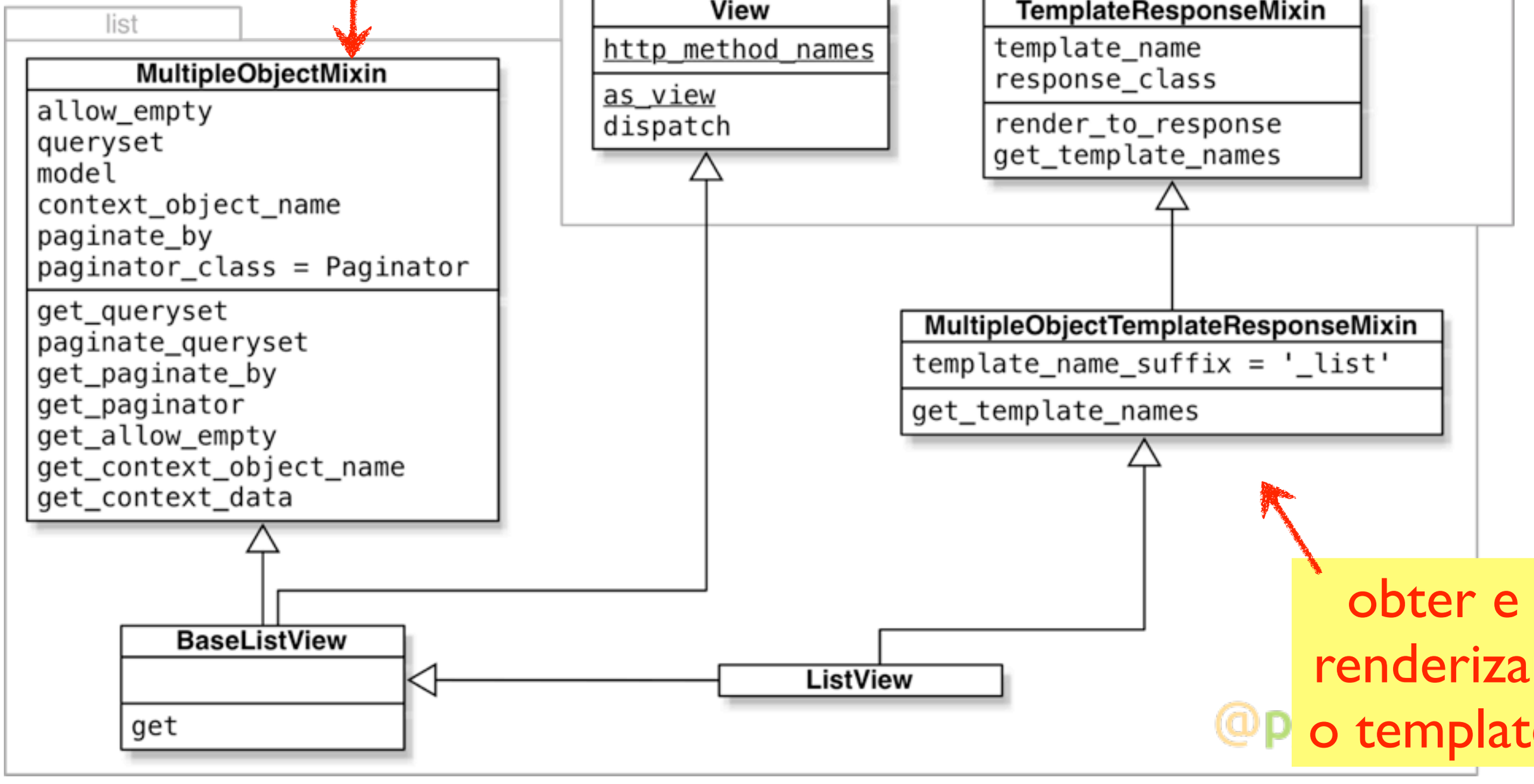


obter e renderizar o template

CBV: views de listagem

identificar e recuperar a lista de objetos

tratar request/response



obter e renderizar o template

Exemplo de uso de CBV

- **django-ibge**: API restful fornecendo JSON para JQuery mostrar regiões, estados e municípios
- No arquivo **municipios/views.py**:
  - uma subclasse bem simples de **ListView**
- No arquivo **municipios/api.py**
  - 4 subclasses de **BaseListView** com **JSONResponseMixin**

https://github.com/oturing/django-ibge