

Desenvolvimento de Sistemas Orientados a Objetos I

Serialização de Objetos

Jean Carlo Rossa Hauck, Dr.

jean.hauck@ufsc.br

<http://www.inf.ufsc.br/~jeanhauck>

Conteúdo Programático

- Técnicas de uso comum em sistemas orientados a objetos
 - Persistência de dados e objetos (serialização)

Persistência é a manutenção do estado de uma estrutura de dados entre execuções de uma aplicação

Uma forma simples de persistência é a **Serialização**, que consiste em gravar em mídia permanente uma imagem (*dump*) do objeto, que pode carregada (*load*) posteriormente

Persistência



[PACHECO & BEPPLER, 2010]



Gravando Objetos Serializados

Em Python é possível gravar objetos serializáveis diretamente em um arquivo binário:

- Utilizar o módulo *pickle*
- Primeiramente abrir o arquivo como escrita, com tipo binário (*wb*)
- Depois persistir os objetos (*dump*)

```
import pickle  
  
clientes = {}  
  
arq_clientes = open('clientes.pkl', 'wb')  
pickle.dump(clientes, arq_clientes)
```

Gravando Objetos Serializados

Em Python é possível gravar objetos serializáveis diretamente em um arquivo binário:

- Utilizar o módulo *pickle*
- Primeiramente abrir o arquivo como escrita, com tipo binário (*wb*)
- Depois persistir os ob

clientes: Um dicionário
que irá conter objetos da
classe **Cliente**

```
import pickle
clientes = {}

arq_clientes = open('clientes.pkl', 'wb')
pickle.dump(clientes, arq_clientes)
```

Gravando Objetos Serializados

Em Python é possível gravar objetos serializáveis diretamente em um arquivo binário:

- Utilizar o módulo *pickle*
- Primeiramente abrir o arquivo como escrita, com tipo binário (*wb*)
- Depois persistir os objetos (*dump*)

```
import pickle
clientes = {}
arq_clientes = open('clientes.pkl', 'wb')
pickle.dump(clientes, arq_clientes)
```

arq_clientes: Referência
para o arquivo de clientes

Gravando Objetos Serializados

Em Python é possível gravar objetos serializáveis diretamente em um arquivo binário:

- Utilizar o módulo *pickle*
- Primeiramente abrir o arquivo como escrita, com tipo binário (*wb*)
- Depois persistir os objetos (*dump*)

clientes.pkl: nome do arquivo.
Será gravado no mesmo
diretório da classe atual

```
arq_clientes = open('clientes.pkl', 'wb')  
pickle.dump(clientes, arq_clientes)
```


Gravando Objetos Serializados

Em Python é possível gravar objetos serializáveis diretamente em um arquivo binário:

- Utilizar o módulo *pickle*
- Primeiramente abrir o arquivo como escrita, com tipo binário (*wb*)
- Depois persistir os objetos (*dump*)

```
import pickle
clientes = {}
arq_clientes = open('clientes.pkl', 'wb')
pickle.dump(clientes, arq_clientes)
```

w: modo escrita (write)
b: arquivo binário

Gravando Objetos Serializados

Em Python é possível gravar objetos serializáveis diretamente em um arquivo binário:

- Utilizar o módulo *pickle*
- Primeiramente abrir o arquivo como escrita, com tipo binário (*wb*)
- Depois persistir os objetos (*dump*)

```
import pickle
clientes = {
    'arq_cliente': 'arq_clientes',
}
pickle.dump(clientes, arq_clientes)
```

dump: Grava o dicionário de clientes no arquivo

Lendo Objetos Serializados

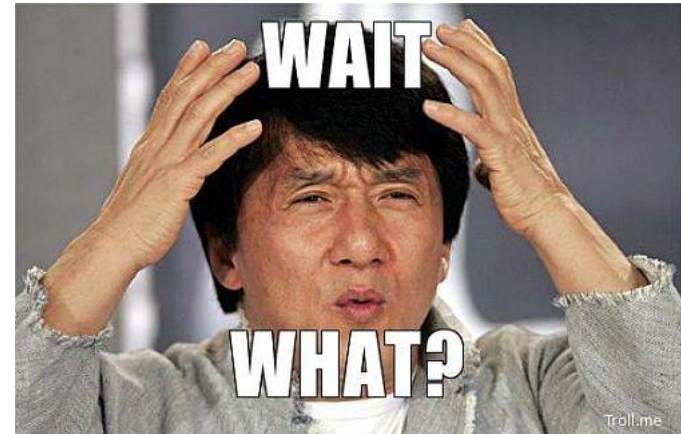
Para ler os objetos serializados de um arquivo binário:

- Utilizar o módulo *pickle*
- Abrir o arquivo como leitura (*read*), com tipo binário (*rb*)
- Carregar os objetos (*load*)

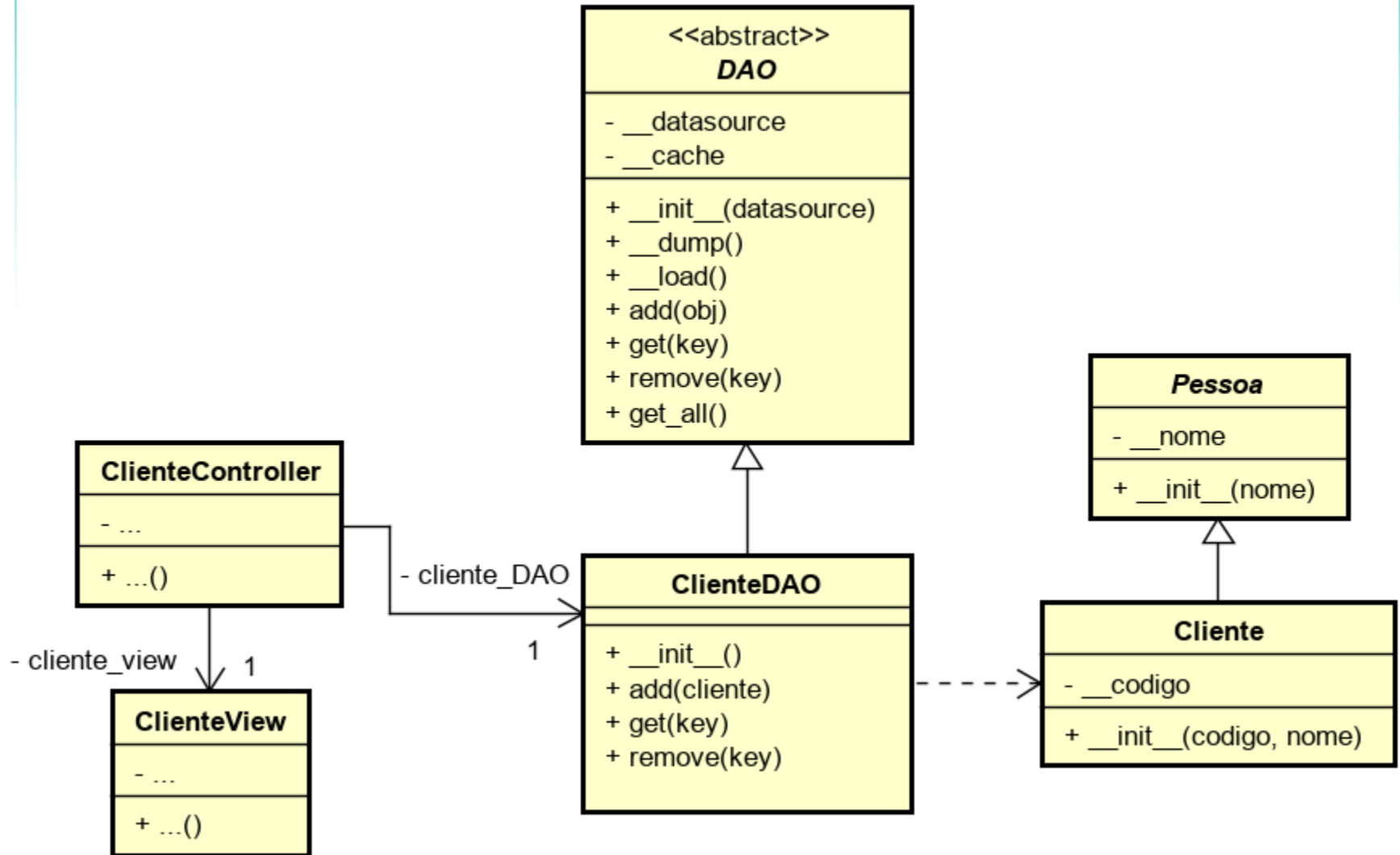
```
import pickle  
  
arq_clientes = open('clientes.pkl', 'rb')  
clientes = pickle.load(clientes, arq_clientes)
```

Passos para persistir

1. Definir quais classes serão persistentes
2. Criar uma classe para Acesso aos Dados da classe persistente
3. Implementar operação para persistir o objeto
4. Implementar operação para carregar os objetos do arquivo
5. Implementar operações de recuperação e inserção na lista controlada pela classe de Acesso aos Dados



Design Pattern: DAO - Data Access Object



Cliente: classe que será persistida

```
class Cliente(Pessoa):
```

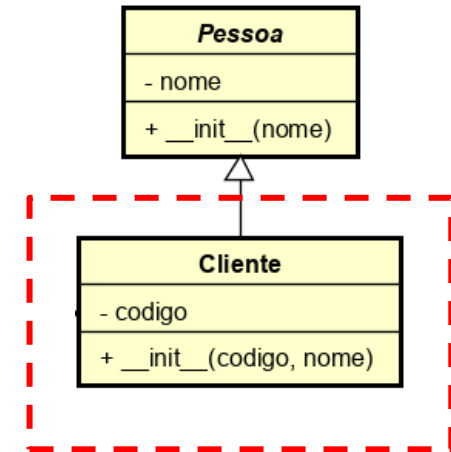
```
    def __init__(self, codigo: int, nome: str):  
        super().__init__(nome)  
        self.__codigo = codigo
```

```
@property
```

```
def codigo(self):  
    return self.__codigo
```

```
@codigo.setter
```

```
def codigo(self, codigo):  
    self.__codigo = codigo
```



Cliente: classe que será persistida

```
class Cliente(Pessoa):
```

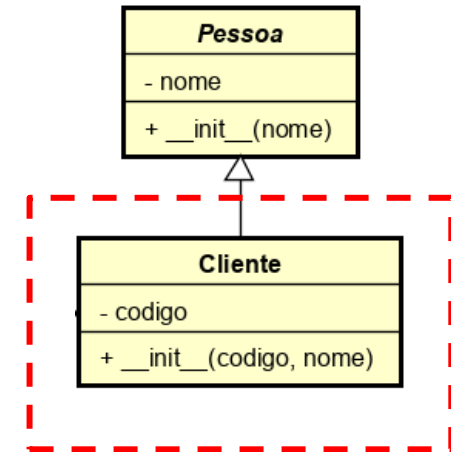
```
    def __init__(self, codigo: int, nome: str):  
        super().__init__(nome)  
        self.__codigo = codigo
```

```
@property
```

```
def codigo(self):  
    return self.__codigo
```

```
@codigo.setter
```

```
def codigo(self, codigo):  
    self.__codigo = codigo
```



Classe DAO abstrata

```
class DAO(ABC):  
    def __init__(self, datasource=''):  
        self.__datasource = datasource  
        self.__cache = {}  
        try:  
            self.__load()  
        except FileNotFoundError:  
            self.__dump()
```

```
    def __dump(self):  
        pickle.dump(self.__cache, open(self.__datasource, 'wb'))
```

```
    def __load(self):  
        self.__cache = pickle.load(open(self.__datasource, 'rb'))
```

...

<<abstract>>

DAO

- __datasource
- __cache

+ __init__(datasource)
+ __dump()
+ __load()
+ add(obj)
+ get(key)
+ remove(key)
+ get_all()

Classe DAO abstrata

```
class DAO(ABC):  
    def __init__(self, datasource=''):  
        self.__datasource = datasource  
        self.__cache = {}  
        try:  
            self.__load()  
        except FileNotFoundError:  
            self.__dump()  
  
    def __dump(self):  
        pickle.dump(self.__cache, open(self.__datasource, 'wb'))  
  
    def __load(self):  
        self.__cache = pickle.load(open(self.__datasource, 'rb'))
```

...

<<abstract>> DAO
- __datasource - __cache
+ __init__(datasource) + __dump() + __load() + add(obj) + get(key) + remove(key) + get_all()

Classe DAO abstrata

```
class DAO(ABC):  
    def __init__(self, datasource=''):  
        self.__datasource = datasource  
        self.__cache = {}  
        try:  
            self.__load()  
        except FileNotFoundError:  
            self.__dump()
```

<<abstract>> DAO
- __datasource - __cache
+ __init__(datasource) + __dump() + __load() + add(obj) + get(key) + remove(key) + get_all()

```
    def __dump(self):  
        pickle.dump(self.__cache, open(self.__datasource, 'wb'))  
  
    def __load(self):  
        self.__cache = pickle.load(open(self.__datasource, 'rb'))
```

...

Classe DAO abstrata ... continuação

```
class DAO(ABC):
```

```
    ...
```

```
    def add(self, key, obj):
```

```
        self.__cache[key] = obj
```

```
        self.__dump()
```

```
    def get(self, key):
```

```
        try:
```

```
            return self.__cache[key]
```

```
        except KeyError:
```

```
            pass
```

```
    def remove(self, key):
```

```
        try:
```

```
            self.__cache.pop(key)
```

```
            self.__dump()
```

```
        except KeyError:
```

```
            pass
```

```
    def get_all(self):
```

```
        return self.__cache.values()
```

<<abstract>>

DAO

- __datasource
- __cache

+ __init__(datasource)
+ __dump()
+ __load()
+ add(obj)
+ get(key)
+ remove(key)
+ get_all()

Classe DAO abstrata ... continuação

```
class DAO(ABC):  
    ...  
  
    def add(self, key, obj):  
        self.__cache[key] = obj  
        self.__dump()  
  
    def get(self, key):  
        try:  
            return self.__cache[key]  
        except KeyError:  
            pass  
  
    def remove(self, key):  
        try:  
            self.__cache.pop(key)  
            self.__dump()  
        except KeyError:  
            pass  
  
    def get_all(self):  
        return self.__cache.values()
```

<<abstract>> DAO
- __datasource - __cache
+ __init__(datasource) + __dump() + __load() + add(obj) + get(key) + remove(key) + get_all()

EAFP (*Easier to Ask for Forgiveness than Permission*)

É mais “barato” deixar estourar a exceção do que todas as vezes procurar se existe

Classe ClienteDAO

```
class ClienteDAO(DAO):  
    def __init__(self):  
        super().__init__('clientes.pkl')  
  
    def add(self, cliente: Cliente):  
        if (isinstance(cliente.codigo, int)) and (cliente is not None) \  
            and isinstance(cliente, Cliente):  
            super().add(cliente.codigo, cliente)  
  
    def get(self, key: int):  
        if isinstance(key, int):  
            return super().get(key)  
  
    def remove(self, key: int):  
        if isinstance(key, int):  
            return super().remove(key)
```

ClienteDAO

- + __init__()
- + add(cliente)
- + get(key)
- + remove(key)

Classe ClienteDAO

**Define na Classe-Pai
DAO o nome do arquivo**

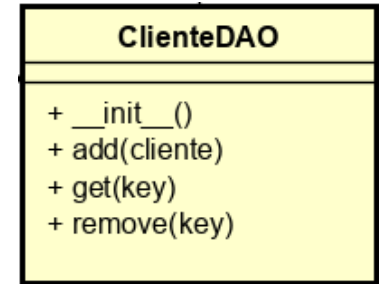
```
class ClienteDAO(DAO):  
    def __init__(self):  
        super().__init__('clientes.pkl')  
  
    def add(self, cliente: Cliente):  
        if (isinstance(cliente.codigo, int)) and (cliente is not None) \  
            and isinstance(cliente, Cliente):  
            super().add(cliente.codigo, cliente)  
  
    def get(self, key: int):  
        if isinstance(key, int):  
            return super().get(key)  
  
    def remove(self, key: int):  
        if isinstance(key, int):  
            return super().remove(key)
```

ClienteDAO

- + __init__()
- + add(cliente)
- + get(key)
- + remove(key)

Classe ClienteDAO

```
class ClienteDAO(DAO):  
    def __init__(self):  
        super().__init__('clientes.pkl')  
  
    def add(self, cliente: Cliente):  
        if (isinstance(cliente.codigo, int)) and (cliente is not None) \  
            and isinstance(cliente, Cliente):  
            super().add(cliente.codigo, cliente)  
  
    def get(self, key: int):  
        if isinstance(key, int):  
            return super().get(key)  
  
    def remove(self, key: int):  
        if isinstance(key, int):  
            return super().remove(key)
```



Garante tipos de objetos e faz validações de negócio. Depois repassa objeto para “add” da Classe-Pai DAO

**Agora implemente no
trabalho ...**

**Desafio: Como fazer um
DAO Genérico?**

**Preparado
para sofrer
um pouquinho?**





Atribuição-Uso-Não-Comercial-Compartilhamento pela Licença 2.5 Brasil

Você pode:

- copiar, distribuir, exhibir e executar a obra
- criar obras derivadas

Sob as seguintes condições:

Atribuição — Você deve dar crédito ao autor original, da forma especificada pelo autor ou licenciante.

Uso Não-Comercial — Você não pode utilizar esta obra com finalidades comerciais.

Compartilhamento pela mesma Licença — Se você alterar, transformar, ou criar outra obra com base nesta, você somente poderá distribuir a obra resultante sob uma licença idêntica a esta.

Para ver uma cópia desta licença, visite <http://creativecommons.org/licenses/by-nc-sa/2.5/br/> ou mande uma carta para Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.