# Programmazione Avanzata

Design Pattern: Singleton

Programmazione Avanzata a.a. 2021-22 A. De Bonis

56

# Il pattern Singleton

- Il pattern Singleton è un pattern **creazionale** ed è usato quando abbiamo bisogno di una classe che ha un'unica istanza che è la sola ad essere utilizzata dal programma.
- In particolare, è utile nelle seguenti situazioni:
  - Controllare l'accesso concorrente ad una risorsa condivisa
  - Se si ha bisogno di un punto globale di accesso per la risorsa da parti differenti del sistema.
  - Quando si ha bisogno di un unico oggetto di una certa classe

## Il pattern Singleton

#### Alcuni usi comuni:

- Lo spooler della stampante: vogliamo una singola istanza dello spooler per evitare il conflitto tra richieste per la stessa risorsa
- Gestire la connessione ad un database

Programmazione Avanzata a.a. 2021-22 A. De Bonis

58

## Il pattern Singleton

- Il pattern Singleton è usato quando abbiamo bisogno di una classe che ha un'unica istanza che è la sola ad essere utilizzata dal programma
- In python creare un singleton è un' operazione molto semplice
- Il Python Cookbook (trasferito presso <u>GitHub.com/activestate/code</u>) fornisce
  - una classe Singleton di facile uso. Ogni classe che discende da essa diventa un singleton
  - una classe Borg che ottiene la stessa cosa in modo differente

#### Il pattern Singleton: la classe Singleton

- L'implementazione della classe è in realta contenuta nella classe \_\_impl
- la variabile \_\_instance fara` riferimento all'unica istanza della classe Singleton che di fatto da un punto di vista implementativo sara` un'istanza della classe \_\_impl

class Singleton:

```
class __impl:
    """ Implementation of the singleton interface """

    def spam(self):
        """ Test method, return singleton id """
        return id(self)

# storage for the instance reference
__instance = None
```

Programmazione Avanzata a.a. 2021-22 A. De Bonis

60

# Il pattern Singleton: la classe Singleton

- Quando viene creata un'istanza di Singleton, \_\_init\_\_() verifica che non esista gia` un'istanza andando a controllare che \_\_instance sia None.
- Se non esiste gia` un'istanza questa viene creata. Nell'implementazione viene di fatto creata un'istanza di \_\_impl alla quale si accede attrraverso la variabile Singleton.\_\_instance.
- In \_\_dict\_\_ della "vera" istanza di Singleton si aggiunge l'attributo \_Singleton\_instance il cui valore è l'istanza di \_\_impl contenuta in Singleton.\_\_instance (unica per tutte le istanze di Singleton)

```
def __init__(self):
    """ Create singleton instance """
    # Check whether we already have an instance
    if Singleton.__instance is None:
        # Create and remember instance
        Singleton.__instance = Singleton.__impl()

# Store instance reference as the only member in the handle
    self.__dict__['_Singleton__instance'] = Singleton.__instance

        Programmazione Avanzata a.a. 2021-22
        A. De Bonis
```

## Il pattern Singleton: la classe Singleton

NB: se creiamo una nuova classe che è sottoclasse di Singleton allora

- se \_\_init\_\_ della nuova classe invoca \_\_init\_\_ di Singleton allora \_\_init\_\_ di Singleton non crea una nuova istanza (non invoca Singleton.\_impl() nell'if)
- se \_\_init\_\_ della nuova classe non invoca \_\_init\_\_ di Singleton allora è evidente che non viene creata alcuna nuova istanza perché a crearle è \_\_init\_\_ di Singleton

Programmazione Avanzata a.a. 2021-22 A. De Bonis

62

## Il pattern Singleton: la classe Singleton

 Ridefinisce \_\_getattr\_\_ e \_\_setattr\_\_ in modo che quando si accede a o si modifica un attributo di un'istanza di Singleton, di fatto si accede a o si modifica l'attributo omonimo di Singleton.\_\_instance

```
def __getattr__(self, attr):
    """ Delegate access to implementation """
    return getattr(self.__instance, attr)

def __setattr__(self, attr, value):
    """ Delegate access to implementation """
    return setattr(self.__instance, attr, value)

# Test it
s1 = Singleton()
print (id(s1), s1.spam())

s2 = Singleton()
print (id(s2), s2.spam())

Programmazione Avanzata a.a. 2021-22
```

# \_\_\_getattr\_\_\_ e \_\_\_getattribute\_\_\_

- object.\_\_getattr\_\_(self, name) restituisce il valore dell'attributo di nome name o lancia un'eccezione AttributeError.
- Quando si accede ad un attributo di un'istanza di una classe viene invocato il metodo object. getattribute (self, name).
- Se la classe definisce anche \_\_getattr\_\_() allora quest'ultimo metodo viene invocato nel caso in cui \_\_getattribute\_\_() lo invochi esplicitamente o lanci un'eccezione AttributeError.
- \_\_getattribute\_\_() deve restituire il valore dell'attributo o lanciare un'eccezione AttributeError.
- L'implementazione di \_\_getattribute\_\_ () deve invocare il metodo della classe base usando il nome della classe base per evitare la ricorsione infinita. Ad esempio, se si vuole invocare \_\_getattribute\_\_ () di object occorre scrivere object.\_\_getattribute\_\_ (self, name).

Programmazione Avanzata a.a. 2021-22 A. De Bonis

64

## Il pattern Singleton: la classe Borg

- Nella classe Borg tutte le istanze sono diverse ma condividono lo stesso stato.
- Nel codice in basso, lo stato condiviso è nell'attributo \_shared\_state e tutte le nuove istanze di Borg avranno lo stesso stato così come è definito dal metodo \_new\_\_.
- In genere lo stato di un'istanza è memorizzato nel dizionario \_\_\_dict\_\_ proprio dell'istanza. Nel codice in basso assegnamo la variabile di classe \_shared\_state a tutte le istanze create

		• • •
new	0	ınıt
116 44	て	11111

- \_\_new\_\_ crea un oggetto
- \_\_init\_\_ inizializza le variabili dell'istanza
- quando viene creata un'istanza di una classe viene invocato prima \_\_new\_ e poi \_init\_\_
- \_\_new\_\_ accetta cls come primo parametro perché quando viene invocato di fatto l'istanza deve essere ancora creata
- \_\_init\_ accetta self come primo parametro

Programmazione Avanzata a.a. 2021-22 A. De Bonis

66

# \_\_new\_\_ e \_\_init\_\_

- tipiche implementazioni di \_\_new\_ creano una nuova istanza della classe cls invocando il metodo \_\_new\_\_ della superclasse con super(currentclass, cls).\_\_new\_\_(cls,...) . Tipicamente prima di restituire l'istanza new modifica l'istanza appena creata.
- Se \_\_new\_\_ restituisce un'istanza di cls allora viene invocato il metodo \_\_init\_\_(self,...), dove self è l'istanza creata e i restanti argomenti sono gli stessi passati a \_\_new\_\_
- Se \_\_new\_\_ non restituisce un'istanza allora \_\_init\_\_ non viene invocato.
- \_\_new\_\_ viene utilizzato soprattutto per consentire a sottoclassi di tipi immutabili (come ad esempio str, int e tuple) di modificare la creazione delle proprie istanze.

Programmazione Avanzata a.a. 2021-22

#### Il pattern Singleton: la classe Borg

- creiamo istanze diverse di Borg: borg e another\_borg
- · creiamo un'istanza della sottoclasse Child di Borg
- aggiungiamo la variabile di istanza only\_one\_var a borg
- siccome lo stato è condiviso da tutte le istanze di Borg, anche child avrà la variabile di istanza only\_one\_var

68

# Il pattern Singleton: la classe Borg

 Se vogliamo definire una sottoclasse di Borg con un altro stato condiviso dobbiamo resettare \_shared\_state nella sottoclasse come segue

```
class AnotherChild(Borg):
    __shared_state = {}

>>> another_child = AnotherChild()

>>> another_child.only_one_var
AttributeError: AnotherChild instance has no attribute
'shared staté
```

#### Il pattern Singleton

- Il libro di Summerfield "Python in Practice ...", il modo piu` semplice per realizzare le funzionalita` del singleton in Python è di creare un modulo con lo stato globale di cui si ha bisogno mantenuto in variabili "private" e l'accesso fornito da funzioni "pubbliche".
- Immaginiamo di avere bisogno di una funzione che restituisca un dizionario di quotazioni di valute dove ogni entrata è della forma (nome chiave, tasso di cambio).
- La funzione potrebbe essere invocata piu` volte ma nella paggior parte dei casi i valori dei tassi verrebbero acquisiti una sola volta.
- Vediamo come usare il design pattern Singleton per ottenere quanto descritto.

Programmazione Avanzata a.a. 2021-22 A. De Bonis

70

#### Il pattern Singleton

- All'interno di un modulo Rates.py possiamo definire una funzione get(), che è la funzione pubblica che ci permette di accedere ai tassi di cambio.
- La funzione get() ha un attributo rates che è il dizionario contenente i tassi di cambio della valute.
- I tassi vengono prelevati da get(), ad esempio accedendo ad un file pubblicato sul Web, solo la prima volta che viene invocata o quando i tassi devono essere aggiornati.
  - L'aggiornamento dei tassi potrebbe essere richiesto a get() mediante un parametro booleano, settato per default a False (aggiornamento non richiesto).

#### Module-level singleton

- Tutti i moduli sono per loro natura dei singleton per il modo in cui vengono importati in Python
- Passi per importare un modulo:
- 1. Se il modulo è già stato importato, questo viene restituito; altrimenti dopo aver trovato il modulo, questo viene inizializzato e restituito.
- 2. Inizializzare un modulo significa eseguire un codice includendo tutti gli assegnamenti a livello del modulo
- 3. Quando si importa un modulo per la prima volta, vengono fatte tutte le inizializzazioni. Quando si importa il modulo una seconda volta, Python resituisce il modulo inizializzato per cui l'inzializzazione non viene fatta.

Programmazione Avanzata a.a. 2021-22 A. De Bonis

72

## Module-level singleton

• Per realizzare velocemente il pattern singleton, eseguiamo i seguenti passi e mantieniamo i dati condivisi nell'attributo del modulo.

```
condivisi nell'attributo del modulo.
    singleton.py:

only_one_var = "I'm only one var"

module1.py:

import singleton
print (singleton.only_one_var )
singleton.only_one_var += " after modification" #una nuova variabile only_one_var
import module2 # import singleton in module2 non inizializza singleton perche'
    #singleton è gia` stato importato in module1

module2.py:
import singleton
print (singleton.only one var)
```