Pattern Recognition
and Applications Lab

La Programmazione ad Oggetti in Python

Docente: Ambra Demontis

Anno Accademico: 2023 - 2024

Corso di Laurea in Ingegneria Elettrica, Elettronica e Informatica

University of Cagliari,
Italy

Department of Electrical and
Electronic Engineering

La Programmazione ad Oggetti in Python

In queste slide vedremo:

- Attributi di Classe
- Metodi di Classe
- Metodi Statici

Attributi di Istanza

Gli attributi dei quali abbiamo parlato fino ad ora **possono avere valori differenti per ogni istanza di classe**.

Ad esempio, ogni studente ha il suo nome e il suo cognome, quindi valori differenti per l'attributo nome e cognome.

Questo tipo di attributi, viene quindi chiamato **attributi di istanza**.

Attributi di Istanza

Come abbiamo visto, la **sintassi** per utilizzare **attributi di istanza è:**

<istanza>.<nome_attributo>

Attributi di Classe

In alcuni casi pratici, **il valore di un attributo non dipende dalla specifica istanza ma solo dalla sua classe di appartenenza**.

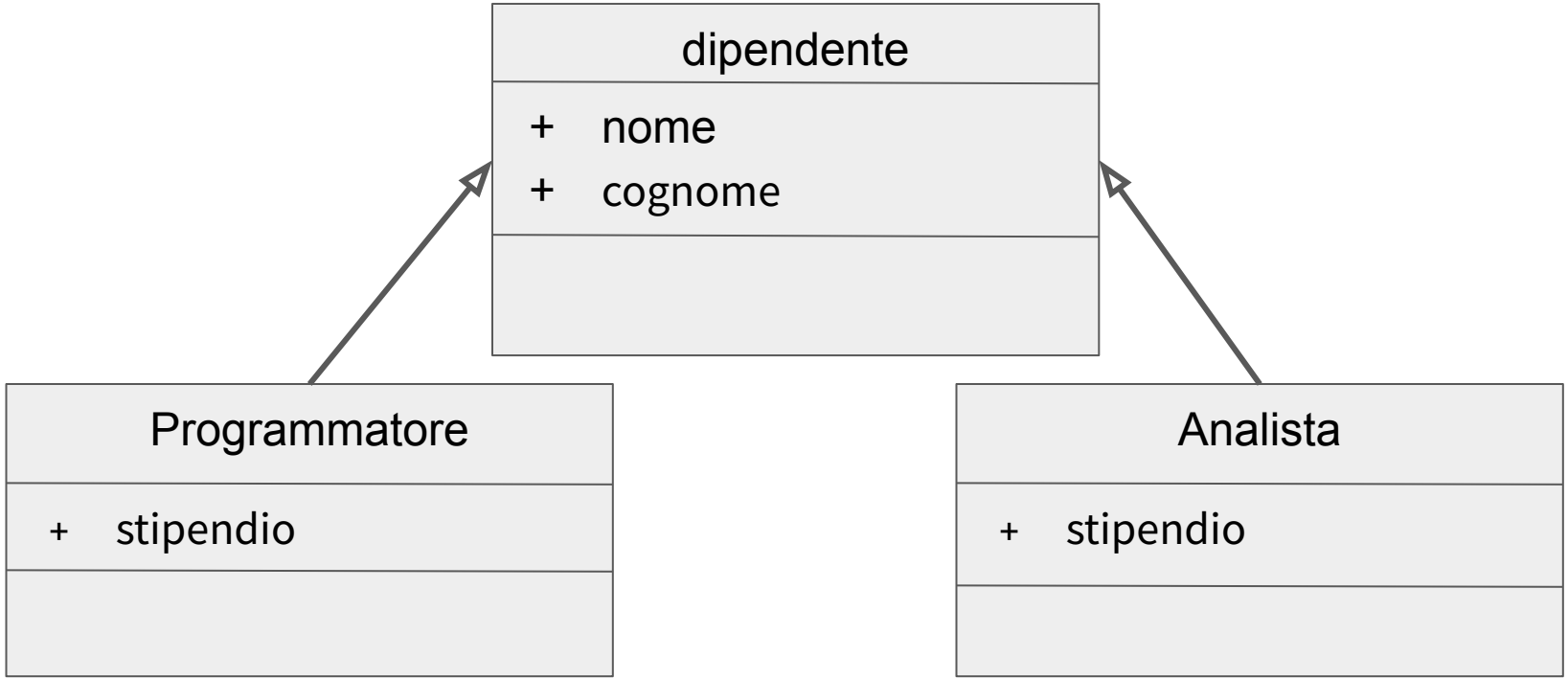
Ad esempio, in un'azienda lo stipendio di un dipendente è generalmente uguale per tutti i dipendenti che ricoprono lo stesso ruolo.

Se esso subisce variazioni, subisce variazioni per tutti i dipendenti che ricoprono quel ruolo.

In Python, questo tipo di attributo viene chiamato **attributo di classe**.

Attributi di Classe

Il diagramma delle classi dell'esempio illustrato nella slide precedente:



Attributi di Classe

La **sintassi** per **definire attributi di classe** è:

```
class <nome_classe>:
    <nome_attributo_di_classe> = <valore>
    …

    def …
```

Vanno definiti subito dopo l'istruzione class <nome_classe>(..).

Attributi di Classe

```
class CDipendente:
  def __init__(self, nome, cognome):
    self.nome = nome
    self.cognome = cognome

class CProgrammatore(CDipendente):
  stipendio = 1500

class CAnalista(CDipendente):
  stipendio = 1800
```

Definiamo l'attributo di classe "stipendio".

Attributi di Classe

Per **accedere al valore dell'attributo di classe,** sia all'interno che all'esterno della definizione di classe, si utilizza la seguente sintassi:

<nome_classe>.<nome_attributo_di_classe>

Questa sintassi ci permette sempre di utilizzare sempre il valore dell'attributo di classe, che sarà identico per tutte le istanze appartenenti a quella classe.

Attributi di Classe

Esempio di utilizzo di un attributo di classe:

```
print("Lo stipendio base di un analista è ", CAnalista.stipendio)
```

Verrà stampato:

Lo stipendio base di un analista è 1800

Attributi di Classe e Attributi di Istanza con lo stesso Nome

E’ possibile definire un attributo di classe e uno di istanza con lo stesso nome.

```
class CDipendente:
 def __init__(self, nome, cognome):
    self.nome = nome
    self.cognome = cognome

class CProgrammatore(CDipendente):
 stipendio = 1500

 def cambia_stipendio(self, value):
    self.stipendio = value
```

Attributi di Classe e Attributi di Istanza con lo stesso Nome

In questo caso, utilizzando la sintassi:

<classe>.<attributo>

Si otterrà il valore dell'attributo di classe.

Attributi di Classe e Attributi di Istanza con lo stesso Nome

Utilizzando la sintassi:

<istanza>.<attributo_di_classe>

Se non esiste un attributo di istanza con il nome <attributo_di_classe> si otterrà il valore dell'attributo di classe, altrimenti quello dell'attributo di istanza.

Attributi di Classe e Attributi di Istanza con lo stesso Nome

```
class CDipendente:
 def __init__(self, nome, cognome):
    self.nome = nome
    self.cognome = cognome

class CProgrammatore(CDipendente):
 # definisce l'attributo di classe stipendio
 stipendio = 1500

 def cambia_stipendio(self, value):
    # definisce e modifica il valore dell'attributo di istanza stipendio
    self.stipendio = value
```

Attributi di Classe e Attributi di Istanza con lo stesso Nome

```
programmatore_anna = CProgrammatore("Anna", "Bianchi")
programmatore_mario = CProgrammatore("Mario", "Rossi")

programmatore_anna.cambia_stipendio(1600)
print("Lo stipendio di anna è ", programmatore_anna.stipendio)
print("Lo stipendio di Mario è ", programmatore_mario.stipendio)
```

Stamperà:

Lo stipendio di anna è 1600

Lo stipendio di Mario è 1500

Attributi di Classe

Per **modificare il valore di un attributo di classe**, sia all'interno che all'esterno della definizione di classe, si utilizza la seguente sintassi:

<nome_classe>.<nome_attriuto_di_classe> = <valore>

Il valore dell'attributo verrà modificato per tutte le istanze di quella classe.

Attributi di Classe

Esempio:

```
programmatore_anna = CProgrammatore("Anna", "Bianchi")

CProgrammatore.stipendio = 1600
print("Lo stipendio di anna è ", programmatore_anna.stipendio)
```

Stamperà:

Lo stipendio di anna è 1600

Esercizio sugli Attributi di Classe

1) Definire la classe CRagioniere che ha un attributo di classe chiamato “stipendio” con valore 1600 e gli attributi di istanza: nome e cognome, età.
2) Scrivere il codice per creare due oggetti appartenenti alla classe CRagioniere.
3) Modificare il valore dell’attributo età e il valore dell’attributo stipendio
4) Scrivere il codice per stampare a schermo, per entrambi gli oggetti, il valore dell’attributo età e quello dell’attributo stipendio.

Esercizio sugli Attributi di Classe

```
class CRagioniere:
  stipendio = 1600

  def __init__(self):
    self.nome = input("inserisci il nome del ragioniere ")
    self.cognome = input("inserisci il cognome del ragioniere ")
    self.età = int (input("inserisci l'età del ragioniere "))
```

Esercizio sugli Attributi di Classe

```
ogg_rag_1 = CRagioniere()
ogg_rag_2 = CRagioniere()

print("età del primo ragioniere ", ogg_rag_1.età)
print("età del secondo ragioniere ", ogg_rag_2.età)

print("età del primo ragioniere ", ogg_rag_1.stipendio)
print("età del secondo ragioniere ", ogg_rag_2.stipendio)
```

Metodi di Istanza

I metodi che abbiamo visto fino ad ora sono i più utilizzati e sono i **metodi di istanza**, cioè **quelli che quando invocati ricevono come primo parametro l'istanza della classe**.

Metodi di Classe

I metodi di classe sono dei metodi che non utilizzano dati dell'istanza quindi non hanno bisogno di ricevere l'istanza dell'oggetto come primo parametro.

Per funzionare, hanno però bisogno di ricevere la classe al quale l'oggetto appartiene. Come primo argomento ricevono quindi in automatico la classe di appartenenza dell'oggetto.

Metodi di Classe

La sintassi per definire un metodo di classe è:

```
class <nome_classe>:
  ..
 @classmethod
  def <nome_metodo>(cls, <parametro1> .. <parametron>):
       …
```

(I riquadri neri evidenziano le differenze con la sintassi per la definizione di un metodo di istanza).

Decoratori

La sintassi per definire un metodo di istanza è:

```
class <nome_classe>:
  ..
  @classmethod
  def <nome_metodo>(cls, <parametro1> .. <parametron>):
      ...
```

"@classmethod" è un "decoratore". I decoratori sono uno strumento per alterare il comportamento di tutte le funzioni che vengono "decorate". Il decoratore @classmethod altera il metodo facendo si che il primo argomento che viene passato in automatico non sia l'istanza dell'oggetto ma la classe.

Metodi di Classe

Esempio di definizione di un metodo di classe:

```
class CAnalista(CDipendente):
  stipendio_base = 1800

  @classmethod
  def stampa_stipendio_base(cls):
    print("Lo stipendio base è: ", cls.stipendio_base)
```

Metodi di Classe

I metodi di classe possono essere invocati utilizzando indifferentemente la sintassi:

<oggetto>.<metodo_di_classe>()

Oppure:

<classe>.<metodo_di_classe>()

Quest'ultima ci permette di utilizzare un metodo di classe senza bisogno di creare un'istanza.

Metodi di Classe

Nota: a prescindere dalla sintassi utilizzata, i metodi di classe ricevono come primo argomento la classe al quale l'oggetto appartiene.

Metodi di Classe

```
# stampa_stipendio_base stampa il valore dell'attributo di classe stipendio_base
CAnalista.stampa_stipendio_base()

oggetto_analista = CAnalista("Anna", "Bianchi")
oggetto_analista.stampa_stipendio_base()
```

Questo codice stamperà:

Lo stipendio base è: 1800

Lo stipendio base è: 1800

Esercizio sui Metodi di Classe

Create un programma che permetta ad un'azienda multinazionale di gestire gli stipendi dei suoi programmatori.

Tutti i programmatori dell'azienda hanno uno stipendio pari a 1500 euro.

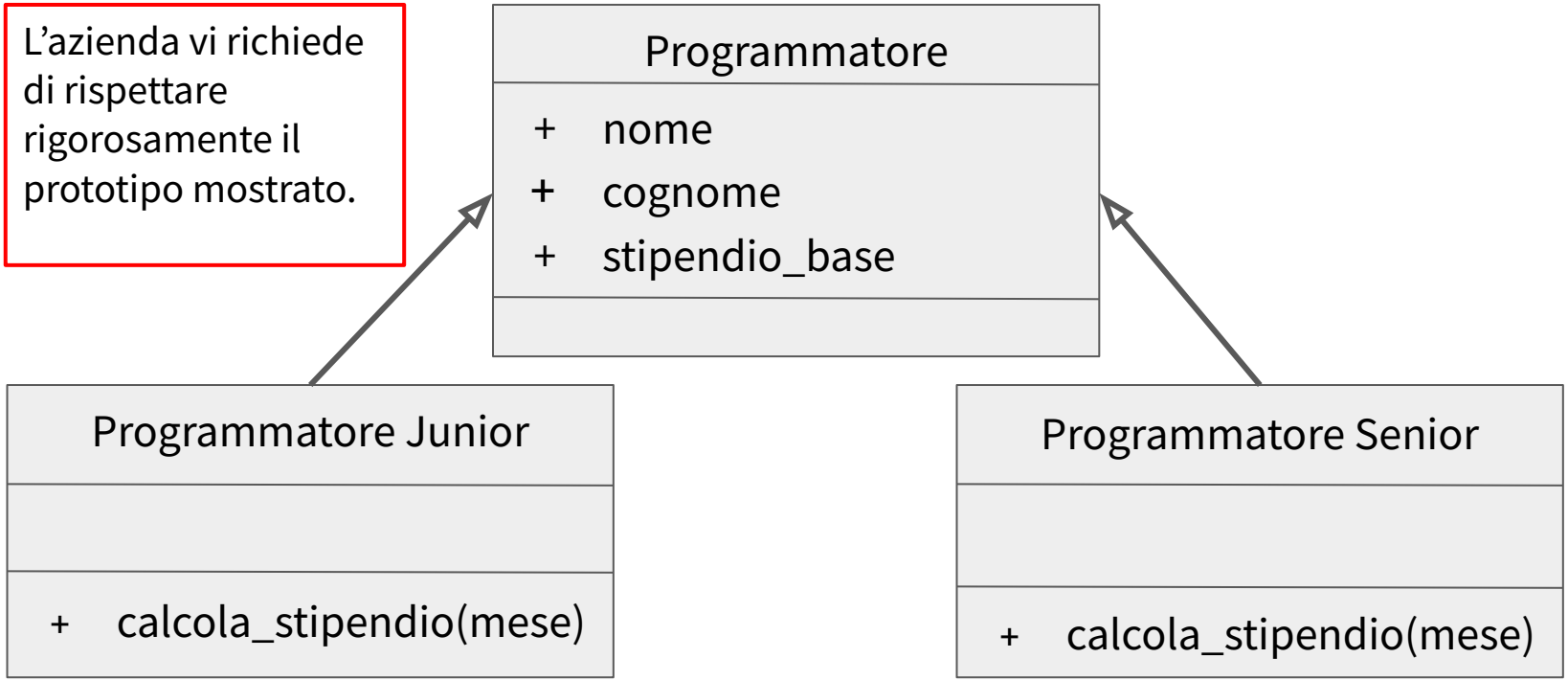
A dicembre però, gli viene data una premialità differente a seconda della loro categoria; lo stipendio viene aumentato di:

- 500 euro, per i programmatori junior
- 1000 euro, per i programmatori senior

Definire gli oggetti secondo il diagramma delle classi mostrato nella slide successiva e stampare a schermo lo stipendio di dicembre dei programmatori.

Esercizio sui Metodi di Classe

L'azienda vi richiede di rispettare rigorosamente il prototipo mostrato.



Esercizio sui Metodi di Classe

```
class CProgrammatore:
  stipendio_base = 1500

  def __init__(self, nome, cognome):
    self.nome = nome
    self.cognome = cognome
```

Esercizio sui Metodi di Classe

```
class CProgrammatoreJunior(CProgrammatore):
  @classmethod
  def calcola_stipendio(cls, mese):
    if mese == 12:
      return cls.stipendio_base + 500
    else:
      return cls.stipendio_base
```

Esercizio sui Metodi di Classe

```
class CProgrammatoreSenior(CProgrammatore):
  @classmethod
  def calcola_stipendio(cls, mese):
    if mese == 12:
      return cls.stipendio_base + 1000
    else:
      return cls.stipendio_base
```

Esercizio sui Metodi di Classe

```
print ("Lo stipendio di dicembre di un programmatore junior è :",
    CProgrammatoreJunior.calcola_stipendio(12))

print ("Lo stipendio di dicembre di un programmatore senior è :",
    CProgrammatoreSenior.calcola_stipendio(12))
```

Metodi di Classe e Costruttori Alternativi

Per poter invocare un metodo di istanza è necessario aver creato un’istanza dell’oggetto.

I metodi di classe possono quindi essere utilizzati per creare dei costruttori alternativi.

Metodi di Classe e Costruttori Alternativi

Abbiamo visto che è possibile far si che il metodo __init__, quando l'oggetto viene creato riceve i valori degli attributi presi in input dall'utente prima della creazione dell'oggetto.

```
class CStudente:

  def __init__(self, nome, cognome, matricola):
     self.nome = nome
     self.cognome = cognome
     self.matricola = matricola
```

Metodi di Classe e Costruttori Alternativi

Abbiamo visto che è anche possibile far si che il metodo __init__ non riceva i valori degli attributi ma che questo metodo a farli inserire in input all'utente.

```
class CStudente:

  def __init__(self):
     self.nome = input("Inserisci il nome dello studente ")
     self.cognome = input("Inserisci il cognome dello studente ")
     self.matricola = input("Inserisci la matricola dello studente ")
```

Metodi di Classe e Costruttori Alternativi

Supponiamo di voler rendere possibili entrambe le cose…

Vogliamo che, se i valori degli attributi sono stati acquisiti sia possibile passarli al metodo __init__.

Vogliamo però anche un metodo (costruttore alternativo) che si occupa di acquisire i valori e creare l'oggetto in modo che se i valori degli attributi non sono ancora stati acquisiti sia possibile richiamare questo metodo per creare l'oggetto.

Metodi di Classe e Costruttori Alternativi

```
class CStudente:
 def __init__(self, nome, cognome, matricola):
    self.nome = nome
    self.cognome = cognome
    self.matricola = matricola

 @classmethod
 def crea_oggetto_studente(cls):
     nome = input("inserisci il nome dello studente ")
     cognome = input("inserisci il cognome dello studente ")
     matricola = input("inserisci la matricola dello studente")
     return cls(nome, cognome, matricola)
```

Metodi di Classe e Costruttori Alternativi

In questo modo è possibile definire i valori degli attributi utilizzando il metodo *__init__* o il costruttore alternativo (il metodo *crea_oggetto_studente*).

```
# definizione degli attributi utilizzando il metodo (__init__)
nome = input("Inserisci il nome dello studente ")
cognome = input("Inserisci il cognome dello studente ")
matricola = input("Inserisci la matricola dello studente ")
obj_studente = CStudente(nome, cognome, matricola)

# creazione usando il costruttore alternativo (crea_oggetto_studente)
oggetto_studente = CStudente.crea_oggetto_studente()
```

Metodi Statici

I metodi statici sono dei metodi che non hanno bisogno di ricevere nè una classe nè una sua istanza.

Non hanno quindi **nessun parametro al quale viene assegnato automaticamente un argomento**.

Metodi Statici

La sintassi per definire un metodo statico è:

```
class <nome_classe>:
  ..
  @staticmethod
  def <nome_metodo>(<parametro1> .. <parametron>):
      ...
```

Metodi Statici

Esempio di definizione di un metodo statico:

```
class CAnalista(CDipendente):
  stipendio_base = 1800

  @staticmethod
  def info_assunzioni():
      print("Per essere assunto come analista il candidato deve aver già "
          "lavorato come analista o avere almeno 5 anni di documentata "
          "esperienza come programmatore.")
```

Metodi Statici

I metodi statici possono essere invocati utilizzando indifferentemente la sintassi:

<oggetto>.<metodo_statico>()

Oppure:

<classe>.<metodo_statico>()

Quest'ultima ci permette di utilizzare un metodo statico senza bisogno di creare un'istanza dell'oggetto.

Esercizio sui Metodi Statici

Considerando l'ultimo esercizio svolto, aggiungere alle classi Programmatore Junior e Programmatore senior un metodo pubblico chiamato *info*.

Questo metodo dovrà stampare:

-Per la classe Programmatore Junior la stringa:

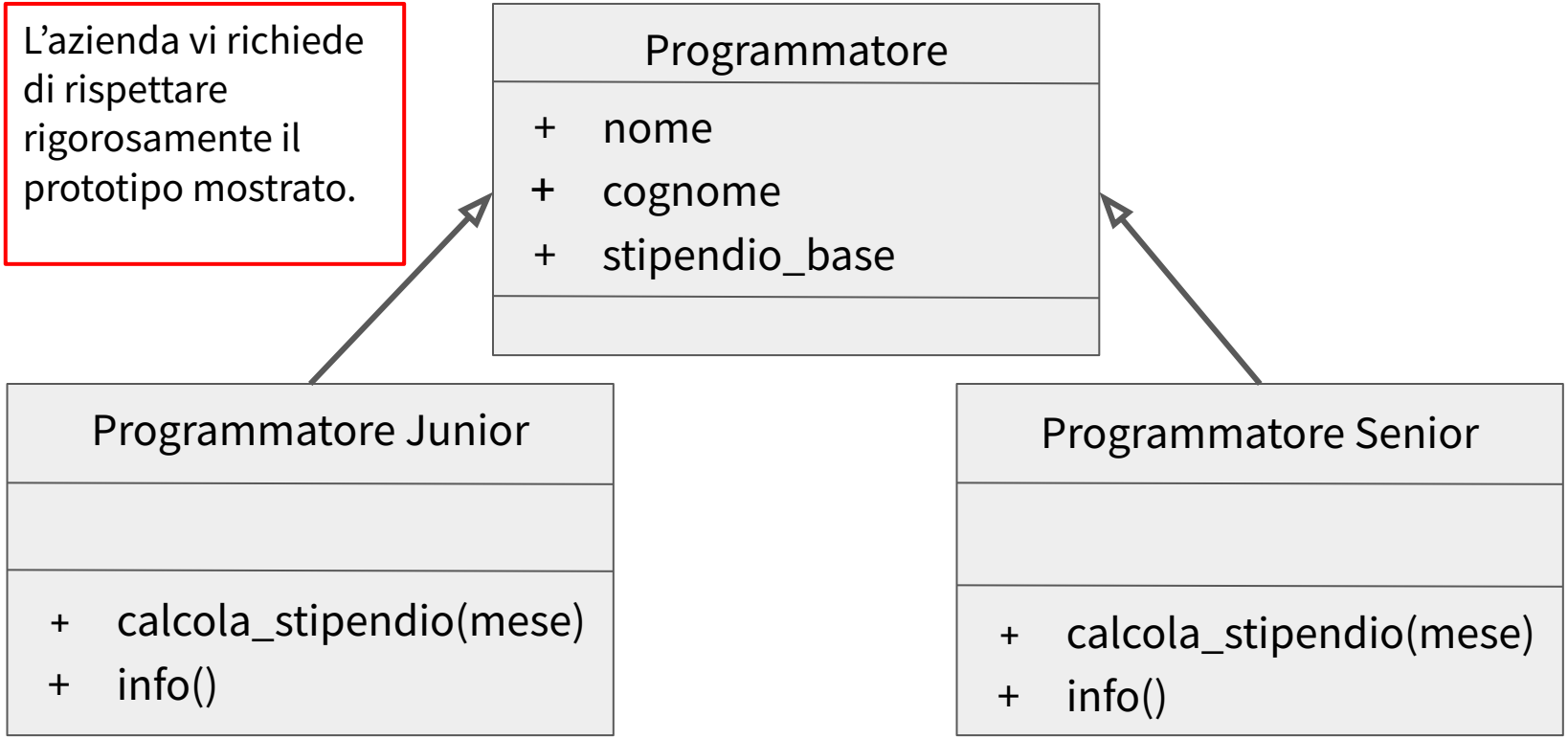
I programmatori junior sono programmatori con meno di 5 anni di esperienza.

-Per la classe Programmatore Senior la stringa:

I programmatori senior sono programmatori con più di 5 anni di esperienza.

Esercizio sui Metodi Statici

L'azienda vi richiede di rispettare rigorosamente il prototipo mostrato.



Esercizio sui Metodi Statici

Nella classe CProgrammatoreJunior aggiungeremo il metodo:

```
@staticmethod
def info():
    print("I programmatori junior sono programmatori con meno di 5 anni "
          "di esperienza")
```

Esercizio sui Metodi Statici

Nella classe CProgrammatoreSenior aggiungeremo il metodo:

```
@staticmethod
def info():
  print("I programmatori senior sono programmatori con più di 5 anni "
        "di esperienza")
```

Esercizio sui Metodi Statici

Per verificare se funzionano correttamente possiamo provare ad invocarli.

CProgrammatoreJunior.info()

CProgrammatoreSenior.info()

Stamperà:

I programmatori junior sono programmatori con meno di 5 anni di esperienza

I programmatori junior sono programmatori con meno di 5 anni di esperienza