Object Oriented Programming

Prof. Ing. Loris Penserini, PhD

https://orcid.org/0009-0008-6157-0396

Scienza Informatica

L'informatica è la scienza che studia i principi e le tecniche per rappresentare, elaborare e trasmettere l'informazione. Essa comprende sia la modellazione del mondo reale tramite strumenti formali e computazionali, sia la progettazione di sistemi e procedure capaci di elaborare informazione in modo autonomo.

Attraverso specifici linguaggi di programmazione, si possono scrivere algoritmi (sequenze di istruzioni) che un calcolatore elettronico è in grado di elaborare in modo automatico al fine di fornire delle informazioni utili all'utente partendo da dati iniziali.

Ciascun paradigma di programmazione fornisce allo sviluppatore di software un differente approccio concettuale per implementare il pensiero computazionale, cioè la definizione della strategia algoritmica utile per digitalizzare e **risolvere un problema del mondo reale**.

L'evoluzione delle tecnologie digitali nei settori della Robotica e dell'AI, spingono i ricercatori a sviluppare nuovi paradigmi di programmazione…

Scienza Informatica e Mondo Reale

Storicamente e pragmaticamente, anche gli aspetti più teorici dell'informatica (come i limiti del calcolo, la complessità o la teoria degli automi) hanno avuto senso perché collegati all'esigenza di modellare e automatizzare compiti del mondo reale.

La complessità computazionale: serve a capire quali problemi del mondo reale sono trattabili in pratica (es. pianificazione, ottimizzazione, crittografia) e quali invece rimangono troppo costosi.

La teoria degli automi e dei linguaggi formali: nasce per modellare linguaggi naturali o di programmazione, cioè modi per descrivere compiti reali da affidare a una macchina.

I limiti del calcolo (problema della fermata, indecidibilità, ecc.): hanno un impatto diretto sulla consapevolezza di quali aspetti del mondo reale possono essere automatizzati e quali resteranno sempre fuori portata.

Scienza Informatica è al servizio dell'uomo

Immagina l'automobile: Prima di essa, l'uomo percorreva distanze a piedi o a cavallo. L'auto non ha inventato il viaggio: lo ha reso più rapido, efficiente, accessibile. Per costruirla, è stato necessario fondere insieme meccanica, termodinamica, elettronica, materiali: discipline diverse, tutte finalizzate a un unico scopo concreto — muoversi dal punto A al punto B in minor tempo.

Così è l'informatica: Non nasce dal nulla, ma dall'esigenza di rappresentare e trattare informazione per raggiungere obiettivi del mondo reale. Anche qui confluiscono logica, matematica, elettronica, linguistica, intelligenza artificiale: saperi diversi che cooperano per un fine comune — automatizzare o ampliare compiti tipicamente umani.

L'informatica, quindi, non è un gioco teorico fine a se stesso. È il motore che, modellando il mondo reale, ci permette di percorrerlo in modi nuovi, più veloci e più potenti, proprio come l'automobile ha trasformato il nostro modo di muoverci nello spazio.

Allora ecco una definizione più consona...

L'informatica è al sapere ciò che l'automobile è al movimento: integra discipline diverse per modellare il mondo reale e trasformare compiti umani in procedure automatiche, riducendo tempi e limiti delle nostre capacità, proprio come l'auto ha reso più rapido ed efficiente lo spostamento da un luogo all'altro.

Programmazione

La maggior parte dei linguaggi di programmazione si basano sul «**paradigma imperativo**», in cui il programma consiste in un insieme di istruzioni dette anche «**direttive**» o «**comandi**».

La programmazione imperativa viene generalmente contrapposta a quella dichiarativa, in cui un programma consiste in un insieme di «affermazioni» (non «comandi») che l'esecutore è tenuto a considerare vere e/o rendere vere. Un esempio di **paradigma dichiarativo** è la programmazione logica (es. Prolog, LTL, CTL, ecc.).

Paradigmi

- **Storicamente i principali Paradigmi di programmazione**
- **Imperativo**
  - Il programma è una sequenza di istruzioni che modificano lo stato.
  - Esempi: C, Fortran, Assembly.
- **Procedurale**
  - Variante dell'imperativo, organizza il codice in procedure o funzioni.
  - Esempi: C, Pascal.
- **Dichiarativo**
  - Si specifica **cosa** deve essere calcolato, non **come**.
  - Esempi: SQL, Prolog.

La contestazione dello «spaghetti code»

Negli anni ’60 e ’70, lo sviluppo software cresceva in complessità.

I programmi erano spesso scritti in Fortran, COBOL o in Assembly, e facevano ampio uso del comando **goto**, che saltava arbitrariamente da un punto all’altro del codice.

Il risultato erano programmi difficili da leggere, mantenere e verificare: il cosiddetto “**spaghetti code**”, per l’intreccio caotico del flusso di controllo.

- **1965 – Corrado Böhm e Giuseppe Jacopini**
  Pubblicano il teorema **Böhm–Jacopini**, dimostrando che ogni algoritmo può essere espresso usando solo tre strutture di controllo:
- **sequenza**,
- **selezione** (if/then/else),
- **iterazione** (while/for).

Questo fu il fondamento teorico della ***programmazione strutturata***.

Modellare il Mondo Reale in OOP

Ciascun paradigma di programmazione fornisce allo sviluppatore un differente approccio concettuale per implementare il pensiero computazionale, cioè la definizione della strategia algoritmica utile per affrontare e risolvere un problema del mondo reale.

La OOP è attualmente il paradigma di sviluppo del SW più utilizzato, per questo molti linguaggi che in passato nascevano non ad oggetti ora lo sono diventati, come il C → C++, il PHP dopo la ver. 5, mentre altri sono nati direttamente OO come JAVA (JSP e Servlet).

Perché OOP per lo sviluppo di SW

Ecco alcuni vantaggi di pensare ad un algoritmo in termini di Oggetti:

- **Astrarre dalla complessità del codice** per concentrarsi maggiormente sulle similitudini tra gli oggetti software e gli oggetti del mondo reale che si vogliono modellare.
- Pensare al comportamento (**behavior**) di un oggetto software come al suo analogo reale: causalità e relativi effetti.
- Notevole aumento della possibilità di **riuso del codice**, con conseguente aumento della produttività di qualità:
  - Maggiore stabilità delle applicazioni;
  - Facilità nell'aggiornare mantenere il codice

La «classe»

Nella OOP, la **classe** è il modulo autocontenuto che definisce il progetto (parziale o intero) dell'algoritmo che si vuole realizzare. E' caratterizzata da:

- **Proprietà:** variabili e metodi
- **Funzioni o metodi**, che contengono la logica dell'algoritmo
- Il **nome della classe** può coincidere con quella del file (in Java è obbligatorio)
- Una «classe» determina lo **scope o campo d'azione** delle sue proprietà
- Lo stato di una classe dipende dai valori di inizializzazione delle sue proprietà nel momento in cui viene caricata in memoria («**istanza**»)

La classe assomiglia al progetto della casa, ma non è la casa!

Esempio concettuale di «classe» in OOP

//classe
Persona
- nome
- cognome
- età
- Interessi
- pagina di saluto

Buongiorno sono «nome» + «cognome»

specializzare

generalizzare

//sottoclassi
Studente
- nome
- cognome
- età
- interessi
- indirizzo_studi
- pagina di saluto_stu

Buongiorno sono «nome» + «cognome», e frequento «indirizzo_studi»

Insegnante
- nome
- cognome
- età
- interessi
- materia
- pagina di saluto_ins

Buongiorno sono «nome» + «cognome» e insegno «materia»

Cosa è un «oggetto»?

Nella OOP il concetto di «**oggetto**» è :

- Un processo in esecuzione in memoria
- Una applicazione che fa uso di funzioni/metodi appartenenti a una o più classi (es. librerie diverse)
- Un'applicazione alla quale si può dare uno stato iniziale che ne determina il comportamento (behavior) iniziale.
- Una applicazione che interagisce con il mondo esterno determinando il comportamento che più si adatta per quello scenario

Differenza tra «classe» e «oggetto»

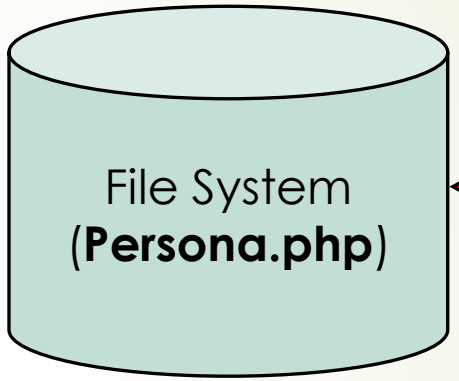
DESIGN time

//classe: è un file
Persona
- nome
- cognome
- età
- Interessi
- pagina di benvenuto

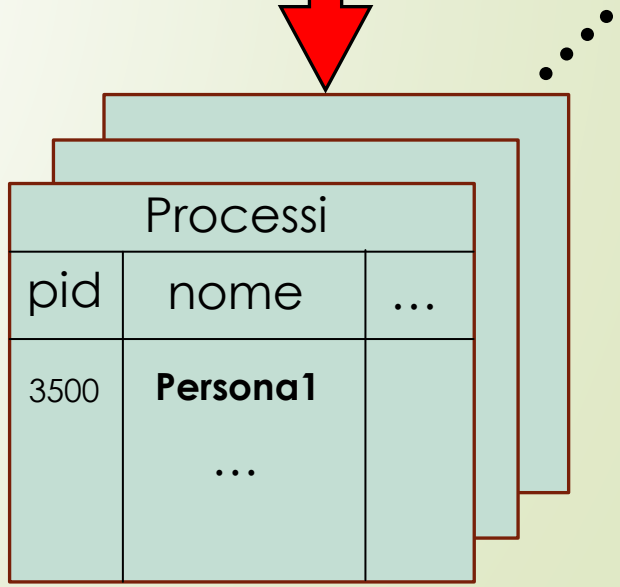
//oggetto: è un processo
Persona1
- nome → Mario
- Cognome → Rossi
- Età → 50
- Interessi → robotica
- pagina di benvenuto → Ciao Mario Rossi

RUN time

Memoria di massa



RAM



Il «costruttore» in Python

In Python, il metodo costruttore è quello chiamato __init__().

Si esegue automaticamente ogni volta che si crea un oggetto della classe.

E' utile scrivere un costruttore solo se occorre:

- inizializzare attributi all'interno dell'oggetto;
- imporre parametri obbligatori;
- eseguire codice ogni volta che un oggetto viene creato.

Se nella classe non si definisce esplicitamente __init__, Python usa un costruttore di default (fornito dal sistema). Questo funziona perché Python eredita automaticamente un __init__ "vuoto" da object, la classe base di tutte le classi.

Proprietà fondamentali in OOP

Tutti i linguaggi OOP forniscono sempre queste tre proprietà:

- **Ereditarietà**
- **Incapsulamento**
- **Polimorfismo**

EREDITARIETA'

Ereditarietà

In Python una classe può ereditare da più classi «padre», cioè applica l'**eredità multipla**.

In PHP (come in Java) **non** è prevista l'ereditarietà multipla come invece è possibile nel C++, per cui in PHP una classe può al più ereditare da una sola altra classe.

Tuttavia, per bravi programmatori OOP, l'ereditarietà singola non è considerata una limitazione ma un vantaggio per progettare SW efficiente e modulare.

La classe Persona

Python

OOP > oop_1 > Persona.py > Persona

```python
class Persona:
    # Costruttore: viene chiamato quando si crea un nuovo oggetto
    def __init__(self, nome, cognome, eta, interessi):
        # Attributi dell'oggetto (proprietà)
        self.nome = nome
        self.cognome = cognome
        self.eta = eta
        self.interessi = interessi
        self.saluto = f"Buongiorno, sono {nome} {cognome}"

    # Metodo che restituisce la frase di saluto
    def get_pag_benvenuto(self):
        self.saluto = f"Buongiorno, sono {self.nome} {self.cognome}"
        return self.saluto
```

La classe Persona

PHP

```php
class Persona {
    //Proprietà
    public $nome = "";
    public $cognome = "";
    public $eta = "";
    public $interessi = "";
    public $saluto = "";

    //costruttore
    public function __construct($nome,$cognome,$eta,$interessi) {
        //inizializzazione
        $this->nome = $nome;
        $this->cognome = $cognome;
        $this->eta = $eta;
        $this->interessi = $interessi;
        $this->saluto = "Buongiorno sono ".$nome." ".$cognome;
    }

    //metodo che restituisce la pagina di saluto
    public function getPagBenvenuto() {
        $this->saluto = "Buongiorno sono ".$this->nome." ". $this->cognome;
        return $this->saluto;
    }
}
```

La classe Studente

Python

OOP > oop_1 > Studente.py > ...

```python
from Persona import Persona

class Studente(Persona):
    def setIndStudio(self,ind_studio):
        self.ind_studio = ind_studio

    def getPagBenvenutoStud(self):
        self.saluto_persona = self.get_pag_benvenuto()
        return self.saluto_persona + ", e frequento " + self.ind_studio

```

La classe Studente

L'operatore «extends»

PHP

```php
class Studente extends Persona {
    public $ind_studio = "";

    //metodo per inserire info specifiche per lo studente
    public function setIndStudio($ind_studio) {
        $this->ind_studio = $ind_studio;
    }

    public function getPagBenvenutoStud() {
        $saluto_persona = $this->getPagBenvenuto();
        return $saluto_persona.", e frequento ".$this->ind_studio;
    }
}
```

Le Istanze di classi sono Oggetti

Python (main.py)

Persona.py | Studente.py | main.py

OOP > oop_1 > main.py > ...

```python
from Persona import Persona
from Studente import Studente

nome = "Loris"
cognome = "Penserini"
eta = "50"
interessi = "Droni"

Persona1 = Persona(nome,cognome,eta,interessi)
print("\033c", end="") # ripulisce il terminale
print("SALUTO DI PERSONA_1:")
print(Persona1.get_pag_benvenuto())
print()

Studente1 = Studente(nome,cognome,eta,interessi)
Studente1.setIndStudio("Sitemi Informativi Aziendali")
print("SALUTO DI STUDENTE_1:")
print(Studente1.getPagBenvenutoStud())

```

output

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
SALUTO DI PERSONA_1:
Buongiorno, sono Loris Penserini

SALUTO DI STUDENTE_1:
Buongiorno, sono Loris Penserini, e frequento Sitemi Informativi Aziendali
```

Le Istanze di classi sono Oggetti

PHP

```php
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <?php
        include 'Persona.php';
        //require_once 'Persona.php';
        include 'Studente.php';

        $nome = "Loris";
        $cognome = "Penserini";
        $eta = "50";
        $interessi = "DRONI";

        //CREO L'OGGETTO "Personal"
        $Personal = new Persona($nome, $cognome, $eta, $interessi);
        $Studente1 = new Studente($nome, $cognome, $eta, $interessi);
        $Studente1->setIndStudio("Sistemi Informativi Aziendali");

        echo "SALUTO DI PERSONA_1: <br>".$Personal->getPagBenvenuto();
        echo "<br><br>SALUTO DI STUDENTE_1: <br>".$Studente1->getPagBenvenutoStud();

        ?>
    </body>
</html>
```

Costruttori di classe

Ogni classe dovrebbe avere il metodo costruttore, poiché definisce lo stato iniziale dell'oggetto associato alla classe. Cioè il metodo costruttore crea un punto di partenza nell'esecuzione del codice dell'oggetto.

Allora nella classe Studente da quale blocco di codice si inizia?

Project work 1

Riutilizzate il codice del progetto appena presentato. E con modifiche minimali, aggiungere la classe «Dirigente» (utilizzando come guida la classe Studente), poi dal **main.py** lanciare un'istanza e accodare a video il relativo saluto:

output

```
SALUTO DI PERSONA_1:
Buongiorno sono Loris Penserini

SALUTO DI STUDENTE_1:
Buongiorno sono Mario Rossi, e frequento Sistemi Informativi Aziendali

SALUTO DI DIRIGENTE_1:
Buongiorno sono Giovanna Rossini, e dirigo la scuola IIS POLO3 FANO
```

PW1: Soluzione PHP

La classe Dirigente (come Studente) eredita dalla classe Persona, ovviamente al contrario di Studente i metodi di Dirigente si specializzano per questo ruolo.

```php
class Dirigente extends Persona {
    public $scuola = "";

    //metodo per inserire info specifiche per lo studente
    public function setScuola($scuola) {
        $this->scuola = $scuola;
    }

    public function getPagBenvenutoDir() {
        $saluto_persona = $this->getPagBenvenuto();
        return $saluto_persona.", e dirigo la scuola ".$this->scuola;
    }
}
```

PW1: Soluzione Python

La classe Dirigente (come Studente) eredita dalla classe Persona, ovviamente al contrario di Studente i metodi di Dirigente si specializzano per questo ruolo.

OOP > oop_pw1 > dirigente.py > ...

```python
from persona import Persona

class Dirigente(Persona):
    def setScuola(self,scuola):
        self.scuola = scuola

    def getPagBenvenutoDir(self):
        self.saluto_persona = self.get_pag_benvenuto()
        return self.saluto_persona + ", e dirigo la scuola " + self.scuola
```

PW1: Soluzione (main.py)

output

```
SALUTO DI PERSONA_1:
Buongiorno, sono Loris Penserini

SALUTO DI STUDENTE_1:
Buongiorno, sono Mario Rossi, e frequento Sitemi Informativi Aziendali

SALUTO DI DIRIGENTE_1:
Buongiorno, sono Giovanna Rossini, e dirigo la scuola IIS POLO3 FANO
```

Python

OOP > oop_pw1 > main.py > ...

```python
from persona import Persona
from studente import Studente
from dirigente import Dirigente

nome = "Loris"
cognome = "Penserini"
eta = "50"
interessi = "Droni"

Persona1 = Persona(nome,cognome,eta,interessi)
print("\033c", end="") # ripulisce il terminale
print("SALUTO DI PERSONA_1:")
print(Persona1.get_pag_benvenuto())
print()

nome = "Mario"
cognome = "Rossi"
eta = "55"
interessi = "Ciclismo"

Studente1 = Studente(nome,cognome,eta,interessi)
Studente1.setIndStudio("Sitemi Informativi Aziendali")
print("SALUTO DI STUDENTE_1:")
print(Studente1.getPagBenvenutoStud())
print()

nome = "Giovanna"
cognome = "Rossini"
eta = "35"
interessi = "Aerobica"

Dirigente1 = Dirigente(nome,cognome,eta,interessi)
Dirigente1.setScuola("IIS POLO3 FANO")
print("SALUTO DI DIRIGENTE_1:")
print(Dirigente1.getPagBenvenutoDir())
```

PW 2: Problema

Riutilizzate il codice del progetto appena presentato. Inserire nella classe «Studente» i metodi necessari per leggere le proprietà: nome, cognome, età e indirizzo di studio.

PW 2: Soluzione

OOP > oop_pw2 > studente.py > ...

```python
from persona import Persona

class Studente(Persona):
    def setIndStudio(self,ind_studio):
        self.ind_studio = ind_studio

    def getPagBenvenutoStud(self):
        self.saluto_persona = self.get_pag_benvenuto()
        return self.saluto_persona + ", e frequento " + self.ind_studio

    def getNomeStud(self):
        return self.nome

    def getCognomeStud(self):
        return self.cognome

    def getEtaStud(self):
        return self.eta

```

PW 2: Soluzione «main.py»

OOP > oop_pw2 > main.py > ...

```python
from studente import Studente

nome = "Mario"
cognome = "Rossi"
eta = "55"
interessi = "Ciclismo"

Studente1 = Studente(nome,cognome,eta,interessi)
Studente1.setIndStudio("Sitemi Informativi Aziendali")
print("Nome, Cognome e Età dello STUDENTE_1:")
print("Nome: " + Studente1.getNomeStud() + " " + Studente1.getCognomeStud() +
      " - Età: " + Studente1.getEtaStud() + " anni")
print()
```

output

```
Nome, Cognome e Età dello STUDENTE_1:
Nome: Mario Rossi - Età: 18 anni
```

GRAZIE!