ES 1 VENERDI 7 FEB

Nell'esercizio proposto, sono state utilizzate diverse tecniche di programmazione orientata agli oggetti (OOP) per modellare un sistema di unità militari e il loro controllo. Di seguito, una descrizione dettagliata delle tecniche di programmazione utilizzate:

**1. Uso delle classi (OOP)**

* **Concetto di classe**: L'oggetto principale della programmazione orientata agli oggetti è la *classe*. È stato creato un insieme di classi per rappresentare le diverse unità militari e il sistema di controllo.
* La classe base **UnitaMilitare** è stata progettata per essere generica, con attributi comuni a tutte le unità (come nome e numero\_soldati) e metodi generici per azioni comuni (come muovi(), attacca() e ritira()).
* Le classi derivate (ad esempio **Fanteria**, **Artiglieria**, **Cavalleria**, **SupportoLogistico**, **Ricognizione**) estendono la classe base **UnitaMilitare**, aggiungendo funzionalità specifiche per ciascun tipo di unità.

**Tecnica utilizzata**:

* Ogni **classe** rappresenta un'entità (unità militare o sistema di controllo).
* Le classi sono utilizzate per organizzare il codice in modo che ogni unità possa avere attributi e comportamenti specifici.

**2. Costruttori**

* Ogni classe, sia la classe base **UnitaMilitare** che le classi derivate, ha un *costruttore* (\_\_init\_\_() in Python) che viene chiamato al momento della creazione di un oggetto.
* **Costruttore in UnitaMilitare**: Il costruttore della classe base inizializza gli attributi nome e numero\_soldati per ogni unità militare.
* def \_\_init\_\_(self, nome, numero\_soldati):
* self.nome = nome
* self.numero\_soldati = numero\_soldati
  + Questo è il metodo che viene utilizzato per inizializzare le variabili d'istanza (ad esempio, nome e numero\_soldati) quando un oggetto della classe viene creato.
* **Costruttore nelle classi derivate**: Le classi derivate chiamano il costruttore della classe base attraverso il comando super().\_\_init\_\_(nome, numero\_soldati) per garantire che gli attributi di base siano inizializzati correttamente.

**Tecnica utilizzata**:

* **Costruttore** per inizializzare gli oggetti con i parametri richiesti.

**3. Ereditarietà**

* L'ereditarietà è una delle caratteristiche fondamentali della programmazione orientata agli oggetti. Permette di definire nuove classi (derivate) basate su classi esistenti (base).
* Le **classi derivate** come **Fanteria**, **Artiglieria**, **Cavalleria**, **SupportoLogistico** e **Ricognizione** ereditano dalla classe base **UnitaMilitare**. Questo consente di riutilizzare il codice definito nella classe base e di aggiungere o modificare il comportamento specifico per ogni tipo di unità.
* Ad esempio, la classe **Fanteria** ha il metodo aggiuntivo costruisci\_trincea(), che è specifico per le unità di fanteria.

def costruisci\_trincea(self):

print(f"L'unità {self.nome} sta costruendo trincee.")

**Tecnica utilizzata**:

* **Ereditarietà** per creare classi specializzate basate su una classe base comune.

**4. Polimorfismo**

* Il polimorfismo si riferisce alla capacità di utilizzare lo stesso nome di metodo in diverse classi, ma con comportamenti diversi.
* In questo caso, ogni classe derivata definisce o estende metodi generici (come muovi(), attacca(), ritira()) della classe base. Ad esempio, la \*\*classe **Artiglieria** non ha bisogno di una nuova implementazione del metodo muovi() perché eredita direttamente il comportamento dalla classe base. Tuttavia, ogni classe derivata aggiunge metodi specifici, come calibra\_artiglieria() per l'artiglieria o esplora\_terreno() per la cavalleria.

**Tecnica utilizzata**:

* **Polimorfismo** attraverso la possibilità di definire metodi generici nella classe base e specifici nelle classi derivate.

**5. Incapsulamento**

* L'incapsulamento è un concetto di OOP che si riferisce al nascondere i dettagli interni dell'oggetto e fornire un'interfaccia pubblica per interagire con l'oggetto.
* In questo caso, i dettagli interni delle unità (ad esempio, la gestione del numero di soldati e del nome) sono nascosti e accessibili solo tramite i metodi pubblici della classe (ad esempio muovi(), attacca(), ritira()).

**Tecnica utilizzata**:

* **Incapsulamento** per nascondere la logica interna dell'oggetto e fornire metodi pubblici per interagire con l'oggetto.

**6. Gestione delle collezioni di oggetti**

* La classe **ControlloMilitare** è responsabile di gestire le unità create. Usa una lista, **unita\_registrate**, per tenere traccia delle unità registrate.
* I metodi della classe **ControlloMilitare** come registra\_unita(), mostra\_unita(), e dettagli\_unita() manipolano questa collezione di unità per aggiungere, elencare e visualizzare informazioni sugli oggetti unità.
* La gestione di una collezione (in questo caso una lista) di oggetti è un aspetto importante della programmazione, poiché consente di organizzare e tracciare dinamicamente le unità create durante l'esecuzione del programma.

def registra\_unita(self, unita):

self.unita\_registrate.append(unita)

**Tecnica utilizzata**:

* **Gestione delle collezioni di oggetti** attraverso una lista per tenere traccia di più oggetti e permettere interazioni con essi.

**7. Comunicazione tramite messaggi (output)**

* Le classi utilizzano il metodo print() per comunicare l'esecuzione delle azioni. Ad esempio, quando un'unità si muove, viene stampato un messaggio specifico, come:

print(f"L'unità {self.nome} si sta muovendo.")

* Questo tipo di comunicazione rende visibili le azioni che vengono eseguite, utile per il debugging e per la visualizzazione dei risultati.

**Tecnica utilizzata**:

* **Comunicazione tramite output** per visualizzare i cambiamenti e le azioni eseguite dalle unità.

**8. Ricerca e Visualizzazione dei Dati**

* Il metodo **dettagli\_unita()** nella classe **ControlloMilitare** permette di cercare una unità registrata in base al nome e visualizzare i dettagli. Si usa un ciclo per iterare attraverso la lista di unità e confrontare il nome:

for unita in self.unita\_registrate:

if unita.nome == nome:

print(f"Dettagli dell'unità {unita.nome}:")

print(f"Numero soldati: {unita.numero\_soldati}")

return

**Tecnica utilizzata**:

* **Ricerca e visualizzazione dei dati** in una collezione di oggetti tramite cicli e condizioni.

**Conclusione:**

L'esercizio fa ampio uso dei principi della **programmazione orientata agli oggetti (OOP)**, tra cui **ereditarietà**, **polimorfismo**, **incapsulamento**, e **astrazione**, per costruire una struttura flessibile e scalabile che modella unità militari e un sistema di controllo. Ogni tecnica è stata implementata per garantire un'alta coesione tra gli oggetti e una bassa dipendenza tra le classi, facilitando l'espansione e la manutenzione del codice.