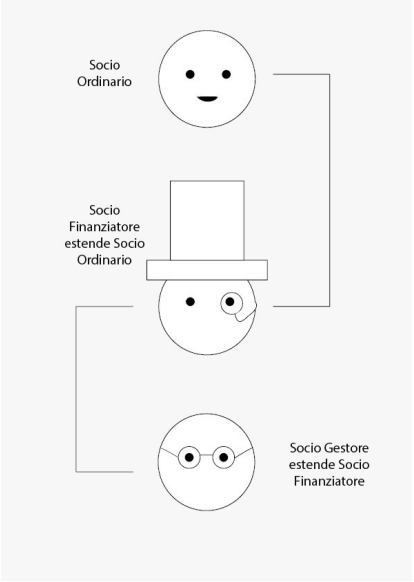
Il programma, pensato per la gestione da riga di comando dei soci e del budget di una cooperativa, consta di quattro classi. *Socio, Finanziatore* e *Gestore* verranno definite meglio nel paragrafo successivo, mentre ***UsaCooperativa*** **presenta il metodo main del programma.** Le quattro classi sono contenute nel package *Coop.*

All'inizio del metodo *main* troviamo le variabili *QUOTA_ORDINARIO*, *QUOTA_FINANZIATORE* e *QUOTA_GESTORE,* facilmente accessibili qualora si decidesse di modificare il programma stesso per aumentare o diminuire le quote d'associazione. **Queste variabili hanno l'attributo *final* per non essere modificate o sovrascritte all'interno del programma.**

A sua volta, **il *main* si appoggia a vari metodi ausiliari per le sue varie funzioni.** Sono qui degni di nota i **metodi *salva()* e *carica(),*** per il salvataggio e il caricamento della lista dei soci e dello storico delle transazioni attraverso serializzazione. **In *UsaCooperativa* è anche presente il metodo *aggiornaStorico()***: esso non viene usato dal *main,* ma dai metodi interni ai vari oggetti. Si è tuttavia deciso di includerlo nella stessa classe del *main* in quanto modifica lo storico delle transizioni definito, e a più riprese evocato, nel *main* stesso.

Infine, poiché **il programma permette all'utente stesso di terminarne l'esecuzione**, è stato giudicato ridondante chiudere i vari di scanner di input.

1. Concettualizzazione delle classi



Il primo problema affrontato è stata la *concettualizzazione delle diverse classi.* Ci siamo trovati di fronte a due possibilità: **rendere i *Gestori* una sottoclasse di *Finanziatori* e i *Finanziatori* una sottoclasse di *Socio***, o rendere le tre classi citate delle sottoclassi di un'ipotetica classe astratta *Socio Generico.* La prima ipotesi è parsa, da un punto di visto progettistico e computazionale, più immediata, **proponendo una diretta ereditarietà fra le varie classi.**

All'interno delle diverse classi troviamo i vari metodi utilizzati dagli oggetti di quella classe (e delle sue sottoclassi), che insieme ai metodi presenti nel *main* definiscono la logica del programma.

2. Array, ArrayList e Vettori

Ci si è interrogati su come definire la lista dei soci, se come *Array*, *ArrayList* o *Vettore.*

Poiché è l'utente stesso che aggiunge man mano i vari *Soci*, sicuramente non può essere un *Array* (non siamo a conoscenza *a priori* del numero massimo di *Soci*). Al tempo stesso, **il programma non prevede di effettuare operazioni particolari sulla lista** e dunque **definirla come *Vettore* avrebbe costituito uno un'inutile dispendio di memoria e risorse computazionali**. La lista dei *Soci* (e similmente la lista dello storico delle operazioni) è stata pertanto definita come *ArrayList.*

***3*. Salvataggio**

Altre particolari considerazioni hanno riguardato le liste dei soci e dello storico in fase di salvataggio. Era richiesto che il programma dovesse poterle salvare e caricare entrambe, ma contemporaneamente o no?

Si è deciso di rendere possibile salvare e caricare le lista solo "a coppia", immaginando possibile utilizzi reali del programma. Se, ad esempio, fosse possibile caricare uno storico diverso dalla lista dei soci, potrebbero figurare nello storico dei soci non presenti nel database.

4. Il metodo *termina()*

Sia nel *main* che nelle varie classi, sono stati implementati dei *try* e *catch* per la gestione delle eccezioni. Si è deciso che ad ogni eccezione si sarebbe stampato un breve messaggio d'errore e si sarebbe lanciato il metodo *termina()*, che si trova nel main. Il metodo chiede all'utente se desidera salvare lo stato del programma prima di terminarlo.

In questo modo **si è permesso di effettuare un salvataggio in caso di eccezioni al *runtime* del programma.**