

# Relazione PAO

**FinanceCalculator**

Panozzo Stefano - Matricola: 1097068 - Anno 2017/2018

## Scopo del progetto:

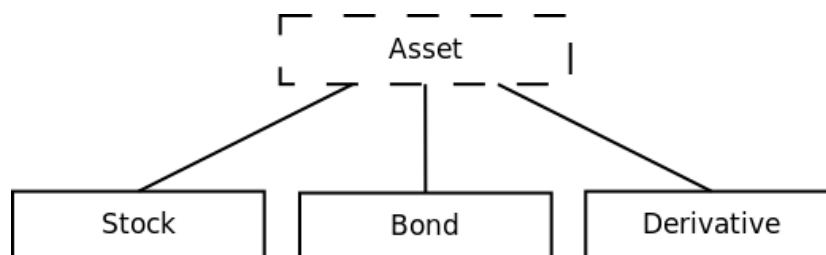
Il progetto si prefigge lo scopo di realizzare una calcolatrice con funzionalità utili alla gestione di un portafoglio di strumenti finanziari vari. Gli strumenti finanziari presi in considerazione sono tre:

- Stock: ovvero i normali titoli azionari
- Bond: obbligazioni che possono essere di Stato (enti pubblici) o private
- Derivative: ovvero strumenti derivati che possono essere di vario genere ma su cui sono possibili alcune operazioni comuni

## Compilazione ed esecuzione:

Per compilare il progetto è necessario eseguire il comando `qmake progetto.pro`, contenuto all'interno della directory dei file inerenti al C++. Tale file permetterà la generazione automatica tramite `qmake` del Makefile, sul quale successivamente sarà possibile lanciare il comando `make`. Per l'esecuzione invece il comando è `./FinanceCalculator`.

## Descrizione della gerarchia e delle classi:



### **Classe base astratta Asset:**

Tale gerarchia costituisce l'insieme degli assets su cui è possibile eseguire le operazioni della calcolatrice. I campi dati sono:

- `name`: il nome dell'asset
- `price`: il prezzo d'acquisto dell'asset
- `sellPrice`: il prezzo di vendita dell'asset
- `taxOn`: definisce se nei calcoli bisogna o meno tenere in considerazione la tassazione
- `tassa`: il valore della tassazione; corrispondente, salvo casi eccezionali, al 26% delle rendite finanziarie
- `unit`: il numero di unità dell'asset possedute

Si è scelto volutamente di non identificare univocamente un articolo tramite il nome, in quanto si dà la possibilità di aggiungere un altro asset analogo con altri dati ed eseguire alcune operazioni che possono risultare utili ad un confronto fra i due.

### **Polimorfismo:**

L'astrazione del concetto di asset è data dai seguenti metodi virtuali puri:

- `virtual double gain(unsigned int) const =0`: è il metodo per il calcolo del gain dato dalla vendita di `n` unità dell'asset. Differisce in base alla tassazione (alcuni Bond possono avere una tassazione agevolata) e all'utilizzo o meno della leva (disponibile solo per asset di tipo Derivative);
- `virtual double totalGain() const =0`: è il metodo per il calcolo del gain più eventuali altre rendite dell'asset, che possono essere il dividendo (Stock) o la cedola (Bond);
- `virtual bool operator==(const Asset* &) const =0`: ridefinizione dell'operatore `==`. Avendo i tre sottotipi campi dati differenti l'uno dall'altro, bisogna ridefinire per ognuno di essi questo operatore;
- `virtual const Asset* unionAsset(const Asset* &) const =0`: è il metodo per il calcolo di un nuovo asset dato dall'unione dell'asset di invocazione con il parametro formale, se essi sono compatibili.

Sono inoltre disponibili altri due metodi virtuali:

- `virtual double roi() const`: calcola il rendimento % del guadagno;
- `virtual double diffgain(const Asset* &) const`: restituisce il gain/loss rispetto ad un altro investimento  
(es. compro 100 unità di stock A a 1 e vendo a 2, compro stock B a 1 e vendo a 3  
→ `A.gain() = 100`, `B.gain() = 200` → `A.diffgain(B) = -50` perché A rende il 50% in meno di B  
→ `B.diffgain(A) = 100` perché B rende il 100% in più di A)

Oltre all'uso del polimorfismo, nella classe base `Asset` è stato reso virtuale il distruttore per non creare memory leak. Tale metodo è fondamentale affinché non venga lasciato garbage nella memoria.

### **Classe derivata Stock:**

Definisce un comune titolo azionario, rappresentativo di una quota della proprietà di una società per azioni.

Il campo dati aggiuntivo di questa classe rappresenta il dividendo, ovvero quella parte di utile che viene distribuita da una società ai suoi azionisti.

I principali metodi di questa classe sono:

- `double gainDividendo() const`: calcola il gain dato dalla distribuzione del dividendo da parte della società di cui si possiedono delle azioni;
- `double gain(unsigned int) const`: ridefinizione del metodo virtuale puro di `Asset`;
- `double totalGain() const`: ridefinizione del metodo virtuale puro di `Asset`. Calcola il gain dato dalla vendita di tutte le azioni e dal guadagno dato dal dividendo;
- `bool operator==(const Asset* &) const`: se tutti i campi dati dei due `Stock` sono equivalenti, essi sono uguali e quindi ritorno `true`;
- `virtual const Asset* unionAsset(const Asset* &) const`: ridefinizione del metodo virtuale puro di `Asset` per oggetti di tipo `Stock`.

### **Classe derivata Bond:**

Definisce una comune obbligazione, ovvero un titolo di debito che può essere emesso da una società o ente pubblico e che attribuisce al suo possessore, alla scadenza, il diritto di rimborso del capitale investito (maggiorato con un possibile guadagno), più un interesse. L'interesse periodico che caratterizza le obbligazioni è la cosiddetta cedola (o coupon). Inoltre, le obbligazioni emesse da alcuni enti pubblici hanno una tassazione agevolata al 12,5% anziché il comune 26%.

La classe `Bond` rende quindi disponibile i seguenti metodi:

- `double gainCedola() const`: calcola il gain dato dalla cedola corrispondente all'obbligazione;
- `double gain(unsigned int) const`: ridefinizione del metodo virtuale puro di `Asset`. Tiene conto della tassazione agevolata se essa è attivata e disponibile;
- `double totalGain() const`: ridefinizione del metodo virtuale puro di `Asset`. Calcola il gain dato dalla vendita di tutte le obbligazioni (solitamente il rimborso delle obbligazioni non è maggiorato ma corrisponde semplicemente al rimborso del capitale inizialmente investito) e dal guadagno dato dalla cedola;
- `bool operator==(const Asset* &) const`: se tutti i campi dati dei due `Bond` sono equivalenti, essi sono uguali e quindi ritorno `true`;
- `virtual const Asset* unionAsset(const Asset* &) const`: ridefinizione del metodo virtuale puro di `Asset` per oggetti di tipo `Bond`.

### Classe derivata Derivative:

Definisce uno strumento derivato, ovvero un “contratto” il cui valore è stabilito e varia in base ad un’altra entità, detta sottostante, la cui natura può essere varia. Essendo lo strumento derivato un oggetto più “astratto” del suo sottostante e contrattato da disparate istituzioni finanziarie, esso si presta all’utilizzo della leva finanziaria, ossia è possibile investire una somma maggiore di quella a nostra disposizione (l’istituzione finanziaria “presta” il capitale mancante). Inoltre, sempre per la sua natura “astratta” è possibile anche fare previsioni al ribasso (dette posizioni short), e quindi guadagnare quando il prezzo del sottostante diminuisce, in contrapposizione alla perdita data da qualsiasi altro strumento finanziario.

La classe Derivative rende quindi disponibile i seguenti metodi:

- `double gain(unsigned int) const`: ridefinizione del metodo virtuale puro di Asset. Tiene conto della leva e della posizione (se short o meno) per il calcolo del gain;
- `double totalGain() const`: ridefinizione del metodo virtuale puro di Asset. Corrisponde al gain su tutte le unità possedute, non avendo i derivati altri metodi di guadagno oltre alla sola vendita dello strumento;
- `bool operator==(const Asset* &) const`: se tutti i campi dati dei due Derivative sono equivalenti, essi sono uguali e quindi ritorno true;
- `virtual const Asset* unionAsset(const Asset* &) const`: ridefinizione del metodo virtuale puro di Asset per oggetti di tipo Derivative.

### Classe Portfolio:

Questa classe è un contenitore per tutti gli asset in possesso. È infatti costituita da un `QVector<const Asset*>` e rende disponibili alcuni metodi per la gestione e il calcolo sull’intero portfolio di strumenti. Questi metodi sono:

- `double valoreTot() const`: calcola il totale investito sugli assets nel portfolio;
- `double gainTot() const`: calcola il gain totale sugli assets nel portfolio;
- `double totalReturn() const`: calcola il capitale a disposizione dopo gli investimenti effettuati, ovvero calcola la somma dei due precedenti metodi;
- `void addAsset(const Asset*)`: aggiunge l’asset al portfolio;
- `void removeAsset(int)`: rimuove l’asset dal portfolio;
- `void setTax(bool)`: setta il valore bool di taxOn;
- `int addUnionAsset(int n, int k)`: effettua l’unione dei due assets in posizione n e k, se essi sono compatibili. Dopo aver aggiunto il nuovo asset, rimuove i precedenti su cui ha eseguito l’operazione.

### Classi della GUI:

La GUI è composta da tre classi, corrispondenti ai rispettivi form .ui creati tramite QtDesigner.

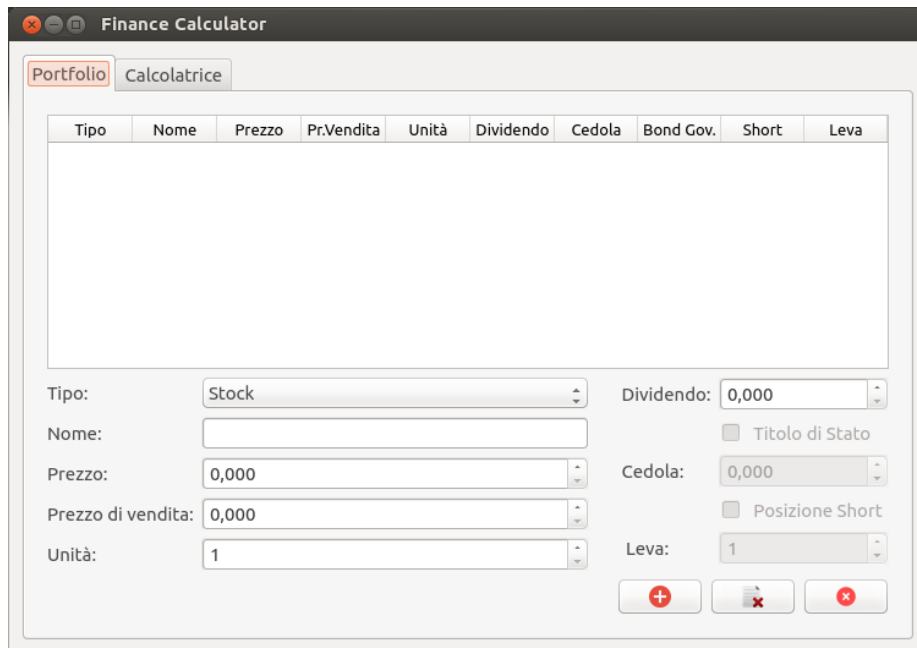
La classe `MainWindow` (`calculator.ui`) rappresenta lo “scheletro” della GUI, in cui sono dichiarati signal e slots per operazioni su entrambe le classi contenute in essa.

È utilizzato un `QTabWidget` che contiene i `QWidget` corrispondenti alle altre due classi: `PortfolioTab` (`tabPortfolio.ui`) e `CalculatorTab` (`tabCalculator.ui`) tramite i quali si esegue l’interazione con l’utente.

Ipoteticamente la GUI è ampliabile semplicemente aggiungendo un nuovo tab al `QTabWidget` di `calculator.ui` e creando un nuovo form per la nuova classe che si andrà poi ad implementare.

## Manuale utente GUI:

### Tab Portfolio:



Finance Calculator

Portfolio Calcolatrice

Tipo	Nome	Prezzo	Pr.Vendita	Unità	Dividendo	Cedola	Bond Gov.	Short	Leva
------	------	--------	------------	-------	-----------	--------	-----------	-------	------

Tipo: Stock

Nome:

Prezzo: 0,000

Prezzo di vendita: 0,000

Unità: 1

Dividendo: 0,000

Cedola: 0,000

Leva: 1

☐ Titolo di Stato

☐ Posizione Short

+ x x

- Tipo: il QComboBox corrispondente serve ad identificare il tipo dell'asset che si vuole aggiungere. In questo caso è selezionato Stock e di conseguenza, a fianco, è disponibile solo il campo 'Dividendo', questo perché i campi sulla destra sono specifici del tipo che si vuole aggiungere e di conseguenza la loro visibilità cambia in base al tipo selezionato;
- Nome: il nome dell'asset (max 7 caratteri; sono consentiti solo lettere e numeri);
- Prezzo: il prezzo di acquisto dell'asset;
- Prezzo di vendita: il prezzo di vendita dell'asset;
- Unità: unità di asset che si vogliono aggiungere;
- Dividendo: valore dividendo;
- Titolo di Stato: selezionare solo se il Bond che si vuole aggiungere è emesso da un ente pubblico e la cui tassazione è agevolata;
- Cedola: valore cedola, indicata come punti percentuali;
- Posizione Short: selezionare solo se sul Derivative che si vuole aggiungere si ha una posizione Short;
- Leva: valore della leva. Una leva 10 significa che il valore totale investito sarà 10 volte il capitale realmente investito dato dal prezzo\*unità acquistate (min 1: nessuna leva, max 400);
- Pulsante ADD (sinistra): aggiunge al portfolio l'asset con i dati appena immessi
- Pulsante RESET (centro): resetta i campi dati
- Pulsante REMOVE (destra): elimina l'asset selezionato nella tabella dal portfolio

**Tab Calcolatrice:**

**Finance Calculator**

Portfolio **Calcolatrice**

Tipo	Nome	Prezzo	Pr.Vendita	Unità	Dividendo	Cedola	Bond Gov.	Short	Leva

☐ Tax    Secondo Operando:     Unità:     Val. Portfolio    Gain Portfolio

+	Unione
Gain	Gain Totale
Difference Gain	ROI %
Gain Dividendo	Gain Cedola

- Tax: selezionare solo se si vuole considerare la tassazione nei calcoli;
- Secondo Operando: definisce il secondo operando su cui eseguire l'operazione se essa lo necessita (+, Unione e Difference Gain). Il numero del secondo operando è indicato dall'indice di fianco all'asset quando esso sarà aggiunto alla tabella;
- Unità: definisce il numero di unità sul quale eseguire l'operazione Gain. Di default si utilizza il totale delle unità a disposizione;
- Val. Portfolio: calcola il valore totale del Portfolio, dato dall'investimento e dal guadagno totale maturato su di esso;
- Gain Portfolio: calcola il guadagno maturato sul portfolio;
- +: calcola la somma del gain totale di due asset. Il primo operando sarà dato dall'asset selezionato nella tabella, il secondo da 'Secondo Operando'. È possibile eseguire più somme consecutive, mantenendo costante il primo asset selezionato e variando 'Secondo Operando';
- Unione: unisce due asset se essi sono compatibili (stesso tipo e, se Bond, valore Titolo di Stato uguale, se Derivative, valore Posizione Short e Leva uguali). Il primo operando sarà dato dall'asset selezionato nella tabella, il secondo da 'Secondo Operando';
- Gain: calcola il guadagno dato dalla vendita di un certo numero di 'Unità' dell'asset selezionato nella tabella;
- Gain Totale: calcola il guadagno totale dell'asset, dato dalla vendita di tutte le unità più possibili altri rendimenti;
- Difference Gain: calcola la differenza di guadagno % fra due asset. Il primo operando sarà dato dall'asset selezionato nella tabella, il secondo da 'Secondo Operando';
- ROI %: calcola il guadagno percentuale sull'asset selezionato nella tabella;
- Gain Dividendo: questa funzionalità è disponibile solo per asset di tipo Stock e calcola il guadagno del dividendo sull'asset selezionato nella tabella;
- Gain Cedola: questa funzionalità è disponibile solo per asset di tipo Bond e calcola il guadagno della cedola sull'asset selezionato nella tabella.

## **Indicazioni conclusive:**

### **Impegno temporale:**

Analisi preliminare: 2-3 h;

Progettazione Modello ed Interfaccia Grafica: 4-5 h;

Apprendimento libreria Qt: 4-5 h;

Codifica Modello ed Interfaccia Grafica: circa 40 h;

Debugging e testing: 3-4 h;

I valori sono stimati, soprattutto per quanto riguarda la codifica dell'interfaccia grafica e l'apprendimento della libreria Qt utilizzata per essa.

### **Specifiche tecniche:**

Sistema operativo: Windows 10 Home

Versione Qt: Qt Creator 4.5.0 based on Qt 5.10.0

Compilatore: MinGW 5.3.0 32bit