Relazione Progetto Kalk

Relazione di Barzon Giacomo(1143164)

Proggetto sviluppato in Coppia con

Giacomo Greggio

**Descrizione Gerarchia di tipi**

La calcolatrice in questione si pone l’obbiettivo di eseguire calcoli tra vari tipi di formato colore come RGB e CMYK. A seguito sarà presente una dettagliata descrizione dell’intera gerarchia di tipi.

* **Color**: classe astratta che definisce i parametri generali che deve possedere un qualsiasi tipo di colore. Color infatti descrive i seguenti metodi virtuali puri, che dovranno quindi essere implementati da tutte le classi derivate:
  + Color\* Complementary(): operazione che coinvolge un unico colore e ritorna il colore complementare di quest’ultimo
  + Color\* Blend(const Color&): Operazione che coinvolge due colori e ritorna un colore che ha come valore dei suoi parametri la media tra i corrispettivi parametri dei colori coinvolti dall’operazione.
  + Color\* Operator+(const Color&): Operazione che coinvolge due colori e ritorna un colore che ha come valore dei suoi parametri la somma tra i corrispettivi parametri coinvolti dall’operazione.
  + Color\* Operator-(const Color&): Operazione che coinvolge due colori e ritorna un colore che ha come valore dei suoi parametri la differenza tra i corrispettivi parametri coinvolti dall’operazione.
  + Color\* Operator\*(const Color&): Operazione che coinvolge due colori e ritorna un colore che ha come valore dei suoi parametri la moltiplicazione tra i corrispettivi parametri coinvolti dall’operazione.
  + Color\* Operator/ (const Color&): Operazione che coinvolge due colori e ritorna un colore che ha come valore dei suoi parametri la divisione tra i corrispettivi parametri coinvolti dall’operazione. Nel caso in cui almeno uno dei parametri del secondo operando abbia valore zero verrà sollevata l’eccezione **DivisionByZero**.
  + Print: Funzione di stampa virtuale.
* **ColorRGB:** classe astratta che deriva da **Color**.
  + Ogni ColorRGB possiede tre attributi privati Red, Green e Blue corrispondenti alle tonalità dei colori primari dell’omonimo standard.

I valori di Red, Green e Blue non vengono limitati all’interno di alcun intervallo in quanto dovranno essere gestiti più approffonditamente all’interno dei successivi sottotipi.

* + All’interno di ColorRGB inoltre vengono dichiarati tre metodi virtuali
    - ColorRGB\* pickRed(const ColorRGB&): Operazione che coinvolge due colorRGB c1 e c2 e ritorna un colorRGB che ha Green e Blue inizializzati con i corrispettivi valori di c1 e Red inizializzato con il corrispettivo valore di c2.
    - ColorRGB\* pickGreen(const ColorRGB&): Operazione che coinvolge due colorRGB c1 e c2 e ritorna un colorRGB che ha Red e Blue inizializzati con i corrispettivi valori di c1 e Green inizializzato con il corrispettivo valore di c2.
    - ColorRGB\* pickBlue(const ColorRGB&): Operazione che coinvolge due colorRGB c1 e c2 e ritorna un colorRGB che ha Red e Green inizializzati con i corrispettivi valori di c1 e Blue inizializzato con il corrispettivo valore di c2.
* **ColorSRGB:** classe concreta che deriva da **ColorRGB**.
  + Versione standard di ColorRGB. I valori di Red, Green, Blue devono essere compresi nell’intervallo [0,255]. Il tentativo di creazione di un ColorSRGB con valori dei parametri non compresi nel suddetto intervallo provocherà il lancio di una tra le eccezioni:
    - **RedValueNotBetweenBounds**
    - **GreenValueNotBetweenBounds**
    - **BlueValueNotBetweenBounds**
  + ColorSRGB in quanto classe concreta definisce tutti i metodi virtuali puri dichiarati nelle classi **Color** e **ColorRGB**
  + Viene utilizzato un metodo privato statico, chiamato come ReduceToBounds, per ridurre i valori dei parametri dell’oggetto ottenuto dalle operazioni all’interno dell’intervallo [0,255]
    - Se il valore è minore di 0 allora esso verrà posto a zero
    - Se al contrario è maggiore di 255 allora sarà equivalente al resto della divisione per 256
* **ColorSCRGB:** classe concreta che deriva da **ColorRGB**.
  + Versione alternativa di ColorRGB che accetta anche valori negativi di Red, Green e Blue(standard realmente esistente sviluppato da Microsoft ed Hp).
  + I valori di Red, Green, Blue devono essere compresi nell’intervallo [-255,255]. Il tentativo di creazione di un ColorSCRGB con valori dei parametri non compresi nel suddetto intervallo provocherà il lancio di una tra le eccezioni:
    - **RedValueNotBetweenBounds**
    - **GreenValueNotBetweenBounds**
    - **BlueValueNotBetweenBounds**
  + ColorSCRGB in quanto classe concreta definisce tutti i metodi virtuali puri dichiarati nelle classi **Color** e **ColorRGB**
  + Viene utilizzato un metodo privato statico, chiamato come ReduceToBounds, per ridurre i valori dei parametri dell’oggetto ottenuto dalle operazioni all’interno dell’intervallo [-255,255]
    - Sia v il valore da ridurre
    - Inizialmente viene calcolato il valore assoluto di v=|v|
    - Successivamente su |v| viene calcolato |v|%256
    - E come ultimo passo viene il valore ottenuto viene moltiplicato per -1 se il valore di partenza era negativo
* **ColorSRGBA:** classe concreta che deriva da **ColorSRGB**
  + Colore SRGB che possiede un’opacità variabile
  + Aggiunge alle caratteristiche ereditate da **ColorSRGB** un campo opacità che ammette solo ed esclusivamente valori compresi nell’intervallo [0,100]. Nel caso in cui si tenti di inizializzare un ColorSRGBA con valore di opacity non compreso all’interno dell’intervallo accettato verra’ lanciata l’eccezione **OpacityValueNotBetweenBounds**.
  + Viene inoltre implementato un metodo aggiuntivo ColorSRGBA\* pickOpacity(const ColorRGB&)che coinvolge due colorRGB c1 e c2 e ritorna un colorRGB che ha Red, Green e Blue inizializzati con i corrispettivi valori di c1 ed opacity inizializzato con il corrispettivo valore di c2.
  + Le operazioni di ColorSRGBA accettano un qualsiasi sottotipo di ColorRGB. Nel caso in cui venga passato come secondo operando un oggetto che non sia sottotipo di ColorSRGBA e che quindi non possieda un opacità variabile verrà preso come valore di default dell’opacità il valore massimo 100.
* **ColorCMYK:** classe concreta che deriva da **Color**.
  + Ogni ColorCMYK possiede quattro attributi privati Cyan, Magenta, Yellow and KeyBlack corrispondenti alle tonalità dei colori primari dell’omonimo standard.
  + I valori di Cyan, Magenta, Yellow e KeyBlack devono essere compresi nell’intervallo [0,100]. Il tentativo di creazione di un ColorCMYK con valori dei parametri non compresi nel suddetto intervallo provocherà il lancio di una tra le eccezioni:
    - **CyanValueNotBetweenBounds**
    - **MagentaValueNotBetweenBounds**
    - **YellowValueNotBetweenBounds**
    - **KeyBlackValueNotBetweenBounds**
  + All’interno di ColorCMYK inoltre vengono dichiarati quattro metodi virtuali
    - pickCyan: Operazione che coinvolge due ColorCMYK c1 e c2 e ritorna un ColorCMYK che ha Magenta, Yellow e KeyBlack inizializzati con i corrispettivi valori di c1 e Cyan inizializzato con il corrispettivo valore di c2.
    - pickMagenta: Operazione che coinvolge due ColorCMYK c1 e c2 e ritorna un ColorCMYK che ha Cyan, Yellow e KeyBlack inizializzati con i corrispettivi valori di c1 e Magenta inizializzato con il corrispettivo valore di c2.
    - pickYellow: Operazione che coinvolge due ColorCMYK c1 e c2 e ritorna un ColorCMYK che ha Cyan, Magenta e KeyBlack inizializzati con i corrispettivi valori di c1 e Yellow inizializzato con il corrispettivo valore di c2.
    - pickKeyBlack: Operazione che coinvolge due ColorCMYK c1 e c2 e ritorna un ColorCMYK che ha Cyan, Magenta e Yellow inizializzati con i corrispettivi valori di c1 e KeyBlack inizializzato con il corrispettivo valore di c2.
  + Viene utilizzato un metodo privato statico, chiamato come ReduceToBounds, per ridurre i valori dei parametri dell’oggetto ottenuto dalle operazioni all’interno dell’intervallo [0,100]
    - Se il valore è minore di 0 allora esso verrà posto a zero
    - Se al contrario è maggiore di 100 allora verrà sarà equivalente al resto della divisione per 101

E’ importante sottolineare come:

* qualsiasi sottotipo di **ColorRGB** accetti come secondo operando delle loro operazioni solo altri sottotipi di **ColorRGB**
* qualsiasi sottotipo di **ColorCMYK** accetti come secondo operando delle loro operazioni solo altri sottotipi di **ColorCMYK**

Non è quindi possibile eseguire operazioni tra **ColorRGB** e **ColorCMYK** in quanto totalmente incompatibili tra loro.

Il tentato passaggio di **ColorCMYK** in una qualsiasi operazione di un **ColorRGB** o viceversa provocherà l’eccezione **IncompatibleOperands**.

Tutte le operazioni tra i tipi della calcolatrice (blend, complementary, somma, sottrazione, pickRed, ecc... ) sono virtuali e quindi polimorfe.

**Descrizione View (CalculatorGui)**

* La view si compone di 3 parti principali
  + **Display**: componente che permette all’utente di visualizzare l’operazione attualmente in corso e l’eventuale risultato ottenuto da quest’’ultima
  + **ColorKeyPad:**Componente formata da vari slider che permette all’utente di scegliere i valori dei colori che verranno immessi nelle varie operazioni. Esistono vari tipi di ColorKeyPad uno per ogni tipo di colore supportato dalla calcolatrice, ed anch’essi sono struttura gerarchica simile a quella di quest’ultimi. Alla base della gerarchia e’ presente una classe astratta ColorKeyPad che descrive tre metodi virtuali puri che dovranno quindi essere sviluppati da tutte le classi derivate
    - Color\* getColor(): ritorna un colore inizializzato con i valori attuali degli slider del ColorKeyPad
    - void setColor(const Color&): setta i valori attuali degli slider con i valori dei parametri dell’oggetto Color passato
    - void reset(): setta i valori attuali degli slider nel loro stato iniziale
  + **OperationKeyPad**:Componente formata da vari pulsanti che permette all’utente di decidere quali operazioni eseguire. Anche esse sono organizzati in una struttura gerarchica simile a quella dei tipi di Colore della calcolatrice.Alla base della gerarchia e’ presente una classe astratta OperationKeyPad che definisce i pulsanti relativi alle operazioni base definite obbligatoriamente su ogni tipo di Colore, mentre le classi derivate di OperationKeyPad aggiungono i pulsanti relativi alle operazioni specifiche ad un determinato tipo di Colore.
* La view inoltre fornisce una serie di segnali, una per ogni operazione eseguibile, i quali passeranno anche l’oggetto Color ritornato dal metodo getColor della ColorKeyPad.

**Descrizione Model**

* La model contiene uno slot per ogni segnale definito sulla view il quale si occupera’ di effettuare le elaborazioni richieste.
* Il risultato delle operazioni verra’ ritrasmesso alla view tramite un apposito segnale void Result(const Color&).
* Eventuali errori incontrati durante l’esecuzione delle operazioni verrano indicati alla view tramite un apposito seganle void Error(std::string).

**Descrizione Controller**

* Il Controller si occupa soltanto di collegare i segnali inviati dalla view coi relativi slot messi a disposizione dal model.

**Descrizione Installazione**

* java(Linux): da riga di comando accedere alla cartella principale Kalk e digitare: **javac -cp “:.” \*.java** in seguito eseguire Use.java tramite: java Use .
* cpp(Linux): La compilazione del progetto necessita di un project file (.pro) per qmake diverso da quello ottenibile tramite l’invocazione di qmake -project. Utilizzare il file Kalk.pro presente all’interno della cartella principale Kalk e procedere con la normale compilazione qmake => make.

**Indicazione delle ore richieste dal progetto e Suddivisione del lavoro progettuale:**

Analisi preliminare del problema: Greggio: 4 ore / Barzon: 4 ore.

Progettazione modello e GUI: Greggio 2 ore / Barzon: 2 ore.

Apprendimento libreria QT: Greggio8 ore / Barzon: 8 ore.

Codifica modello: Greggio 15 ore / Barzon: 15 ore.

Codifica GUI: Barzon 13 ore.

Debugging: Greggio 3 ore / Barzon: 3 ore.

Testing: Greggio 3 ore / Barzon: 3 ore.

Java: Greggio 13 ore.

L’analisi preliminare del progetto è stata pensata e realizzata insieme quindi il tempo dedicato equivale.

La progettazione della parte logica è stata equamente divisa.

L’apprendimento della libreria Qt è stato fatto da entrambi anche se poi lo sviluppo della parte di Qt Model-Control-View e GUI è stata sviluppata da Barzon.

La gerarchia in Java è stata sviluppata da Greggio.

Debugging e Test sono stati fatti da entrambi su tutto il progetto.

**Descrizione specifiche software di sviluppo**

* Sistema Operativo: Windows 10 Home
* Versione QT: Qt 5.6.3 MinGW 32bit
* Versione Compilatore: MinGW-W64 v7.1.0
* Versione Java: 9.0.4