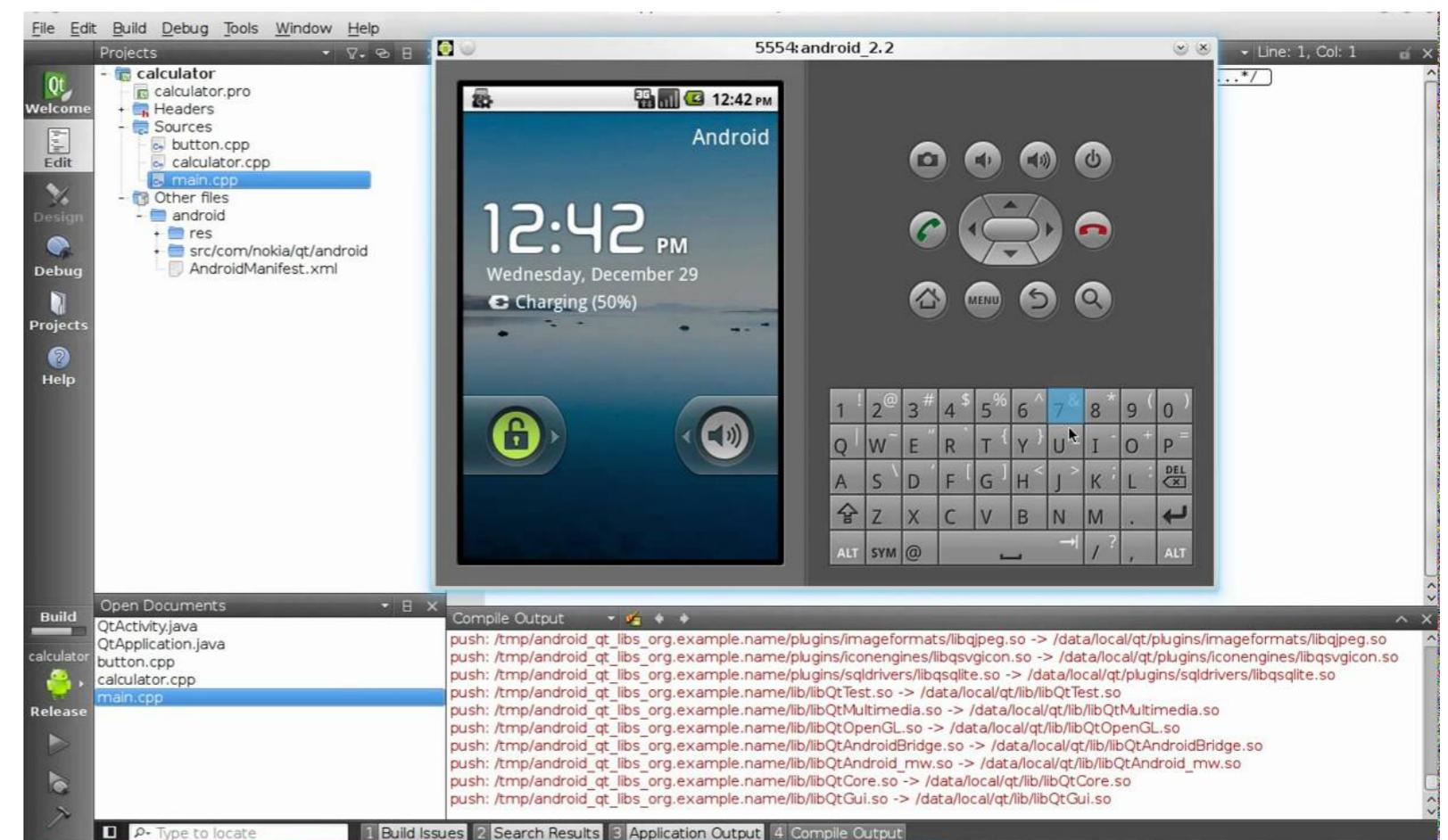
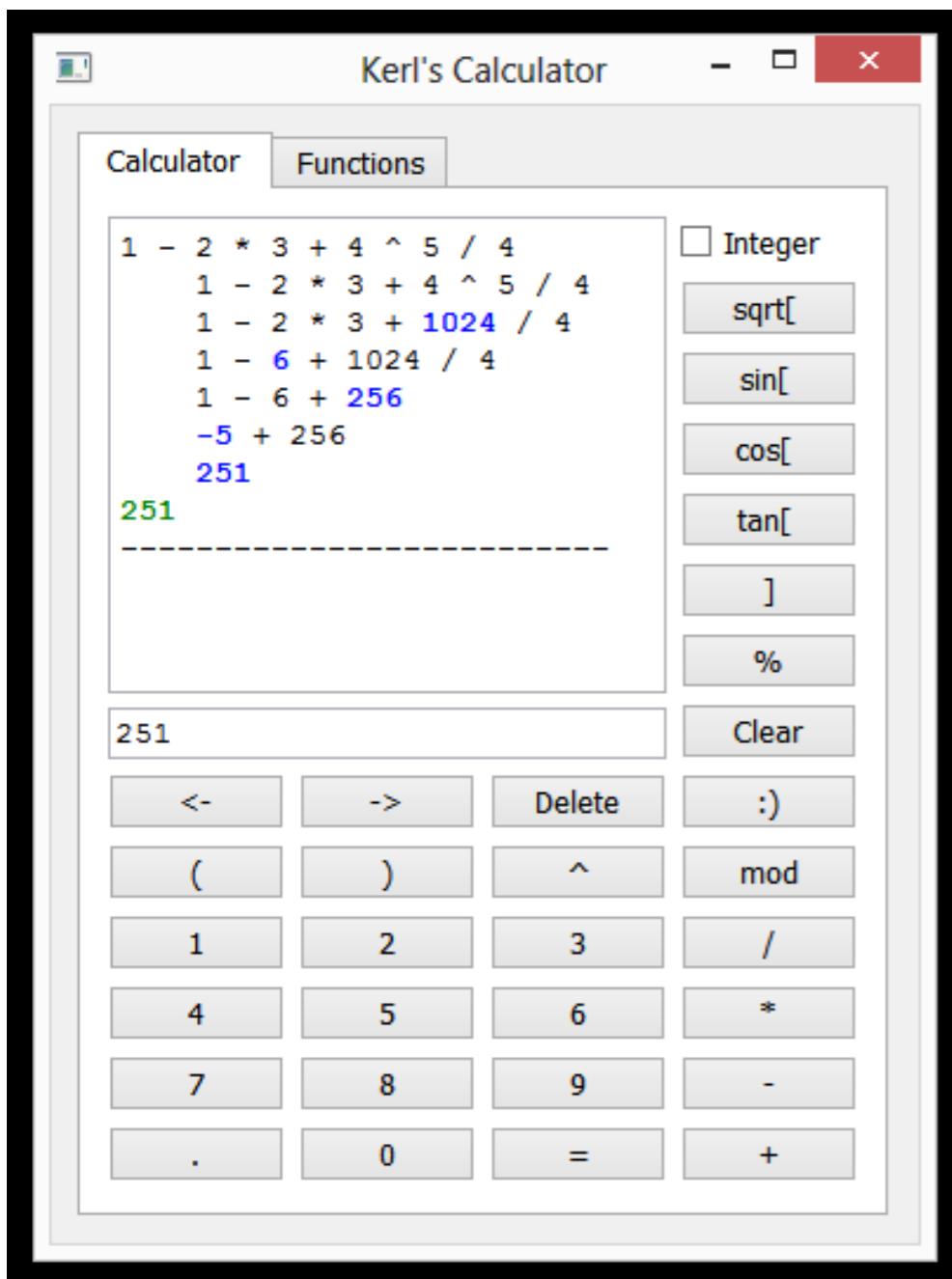




Progetto PAO-2017/2018

Kalk



1 Scopo

Lo scopo del progetto KALK è lo sviluppo in C++ di una calcolatrice estendibile dotata di una interfaccia utente grafica (GUI) sviluppata in Qt. La calcolatrice dovrà essere in grado di gestire i calcoli di diversi tipi di dati (non necessariamente di natura numerica). Questi tipi di dati devono essere implementati anche in Java. Il progetto dovrà soddisfare alcuni requisiti obbligatori. Il progetto può essere sviluppato da una coppia di studenti. Il progetto deve richiedere approssimativamente 50 ore di lavoro complessivo per persona.

1.1 Requisiti Obbligatori

La calcolatrice KALK deve soddisfare i seguenti requisiti obbligatori:

1. Separazione tra il codice del modello logico di KALK ed il codice della GUI. Il modello logico deve consistere di un insieme di classi senza alcuna relazione con il codice della GUI. Idealmente il modello logico di KALK dovrebbe poter essere usato anche da una calcolatrice a riga di comando, quindi non supportata da una GUI.
2. Il modello logico di KALK deve includere **almeno tre** tipi di dato non banali (cioè che non siano semplici wrapper attorno a tipi primitivi del C++ come `int` o `double`) che rappresentino tre diversi tipi su cui KALK offre delle funzionalità di calcolo. Se il progetto è sviluppato da una coppia di studenti allora i diversi tipi di dato di calcolo devono essere **almeno quattro**. Non è necessario definire dei tipi di natura numerica. Alcuni possibili (e generici) esempi di tipi di calcolo: matrici, stringhe, liste, vettori di uno spazio vettoriale (ad esempio i numeri complessi), intervalli di numeri, polinomi, date, orari, numeri in rappresentazione binaria, sottoinsiemi di un dato insieme, partizioni di un dato insieme, maps, etc.
3. I tipi di calcolo di KALK (che, come richiesto dal punto 2, devono essere almeno tre o quattro in caso di progetto svolto da una coppia) devono includere almeno due tipi C_1 e C_2 tali che C_1 e C_2 sono sottotipi di un qualche tipo B , che può essere o meno un tipo di calcolo di KALK. Inoltre, i tipi C_1 e C_2 devono offrire delle funzionalità di calcolo aggiuntive rispetto a quelle offerte dal supertipo B . Ad esempio, se `Intero` è un sottotipo di `Reale` allora `Intero` deve offrire delle operazioni che possono essere applicate solamente ad oggetti di `Intero`, ad esempio una operazione di calcolo del fattoriale.
4. **Requisito Java:** I tipi di calcolo di KALK devono essere implementati anche in Java, offrendo le stesse funzionalità della loro implementazione in C++. Questa implementazione Java deve essere corredata da un esempio di uso di tutte le funzionalità offerte da questi tipi di calcolo, quindi da una classe `Use` contenente un metodo `main` d'uso di tutte le funzionalità.
5. La GUI di KALK deve permettere l'uso di tutte le operazioni dei tipi di calcolo supportati. In ogni momento, KALK opera su un tipo di calcolo corrente, scelto dall'utente tra tutti i tipi di calcolo supportati da KALK. Inoltre, qualche tipo C potrebbe fornire delle operazioni che ritornano valori di un diverso tipo di calcolo D : ad esempio, potrebbe essere definita la radice quadrata di un numero intero che ritorna un numero reale. Se il tipo corrente di KALK è C e l'utente utilizza una tale operazione che trasforma un tipo di input C in un tipo di output D allora, coerentemente, il tipo di calcolo corrente di KALK cambia da C a D .
6. KALK deve essere supportata da una completa gestione degli errori. Ogni tipo di errore (ad esempio: divisioni per zero, overflow, underflow, radici quadrate di numeri negativi, etc) deve riportare un chiaro messaggio di errore all'utente, senza corrompere lo stato della calcolatrice o peggio provocarne un errore run-time del programma.

Su github (ed in generale sul web) si trovano numerosissimi esempi di calcolatrici per diversi tipi di dato (stringhe, matrici, polinomi, etc), anche implementate in C++, da cui poter trarre qualche ispirazione per il progetto KALK.

The screenshot shows a GitHub search interface for the query "calculator". The top navigation bar includes links for Pull requests, Issues, Marketplace, and Explore, along with user and repository icons. Below the bar, metrics for repositories (105K), code (10M), commits (487K), issues (50K), wikis (8K), and users (139) are displayed, with an Advanced search link. The main search results section shows 105,538 repository results. The first result is "ahfarmer/calculator", described as a "Simple calculator built with React", updated on May 16, and tagged with "calculator", "react", "minimal", and "example-project". It is written in JavaScript and has 118 stars. The second result is "saucelabs/calculator", described as a "Sample Calculator App for Android", updated 27 days ago, and tagged with "calculator", "react", "minimal", and "example-project". It is written in Java and has 13 stars. To the right of the results, a sidebar titled "Languages" lists the most used languages in these repositories, with Java at the top followed by JavaScript, Swift, HTML, C#, Python, C++, Objective-C, Ruby, and CSS.

Language	Count
Java	24,048
JavaScript	15,835
Swift	8,249
HTML	7,771
C#	6,251
Python	6,070
C++	3,904
Objective-C	3,128
Ruby	3,002
CSS	2,531

Interfaccia Grafica

La GUI può naturalmente trarre ispirazione dalle innumerevoli calcolatrici disponibili (applicazioni per sistemi operativi desktop oppure app per sistemi mobili). Si potra` aderire al design pattern Model-View-Controller o Model-View per la progettazione architettonale della GUI, dove il Model includera` il modello logico di KALK. Qt include un insieme di classi di “view” che usano una architettura “model/view” per gestire la relazione tra i dati logici della GUI ed il modo in cui essi sono presentati all’utente della GUI (si veda <http://qt-project.org/doc/qt-5/model-view-programming.html>). Come noto, la libreria Qt e` dotata di una documentazione completa e precisa che sara` la principale guida di riferimento nello sviluppo della GUI, oltre ad offrire l’IDE QtCreator ed il tool QtDesigner. La libreria Qt offre una moltitudine di classi e metodi per lo sviluppo di GUI dettagliate e user-friendly.

2 Valutazione del Progetto

Un buon progetto dovrà essere sviluppato seguendo i principi fondamentali della programmazione orientata agli oggetti, anche per quanto concerne lo sviluppo dell’interfaccia grafica. La valutazione del progetto prenderà in considerazione i seguenti criteri:

1. Correttezza (peso 10%): il progetto deve:

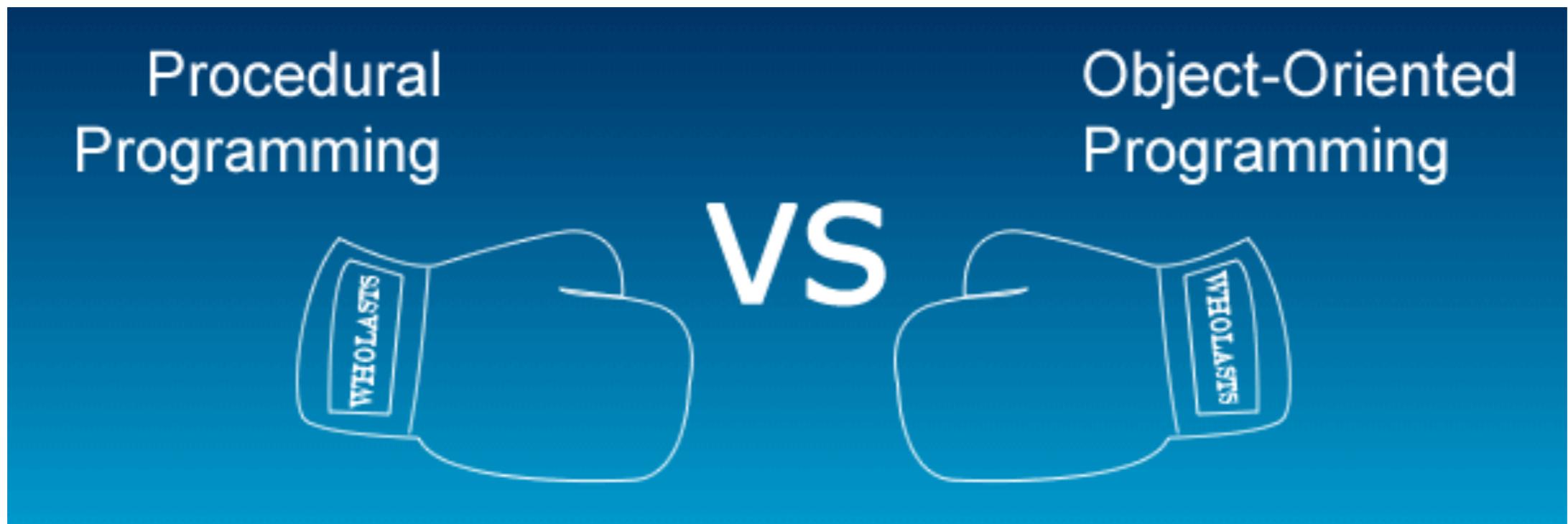
- (a) compilare ed eseguire correttamente nella macchina `ssh.studenti.math.unipd.it`, di cui è stata fornita una immagine da usare come macchina virtuale in VirtualBox. **NB: si tratta di una condizione necessaria per la valutazione del progetto.** La compilazione dovrà essere possibile invocando la sequenza di comandi: `qmake -project` ⇒ `qmake` ⇒ `make`. Se la compilazione del progetto necessita di un project file (`.pro`) per `qmake` diverso da quello ottenibile tramite l’invocazione di `qmake -project` allora deve anche essere consegnato un file `progetto.pro` che permetta la generazione automatica tramite `qmake` del `Makefile`, e ciò dovrà essere esplicitamente documentato nella relazione.
- (b) soddisfare pienamente e correttamente tutti i requisiti obbligatori



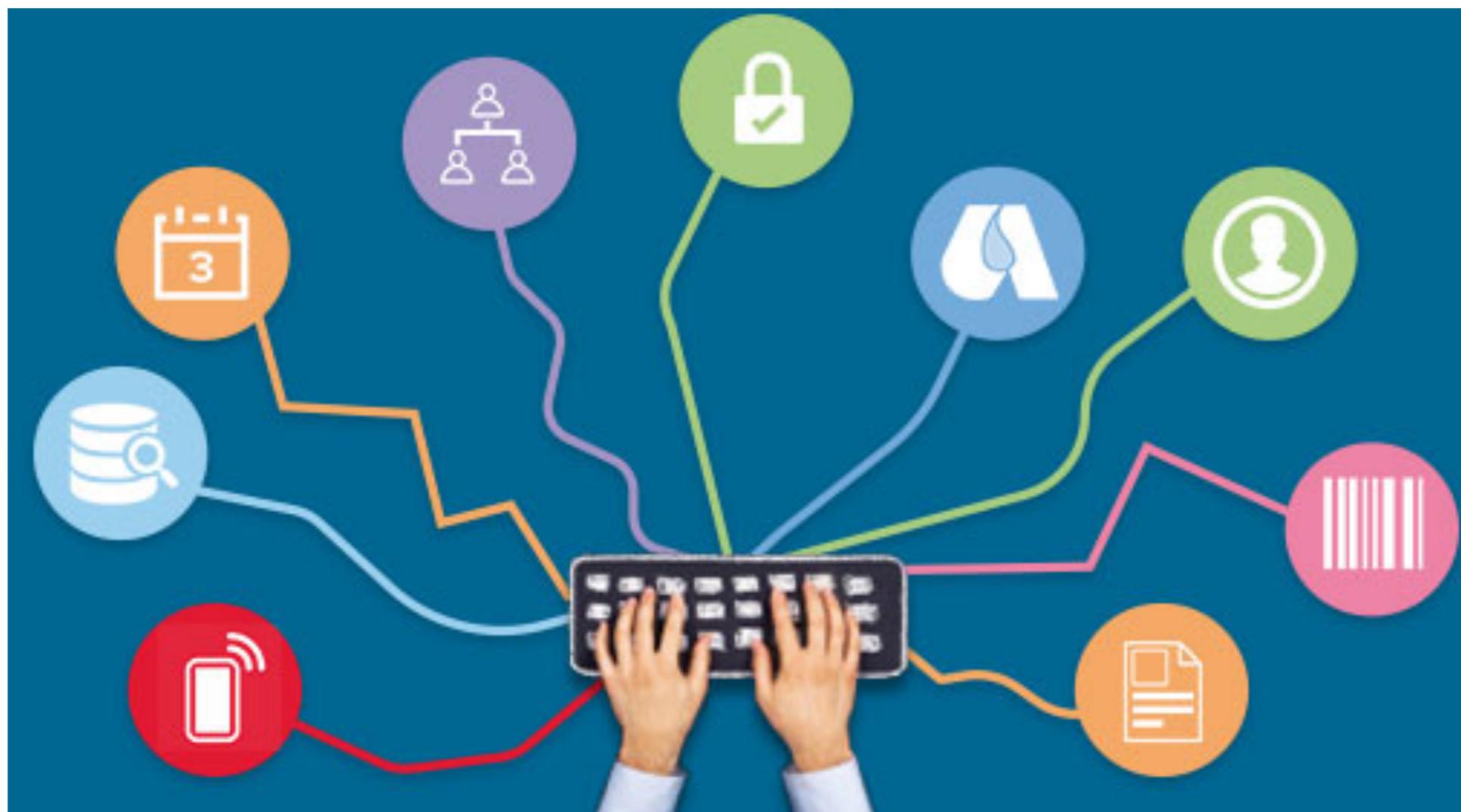
correttezza

2. Orientazione agli oggetti (peso 30%): qualità desiderabili del codice prodotto:

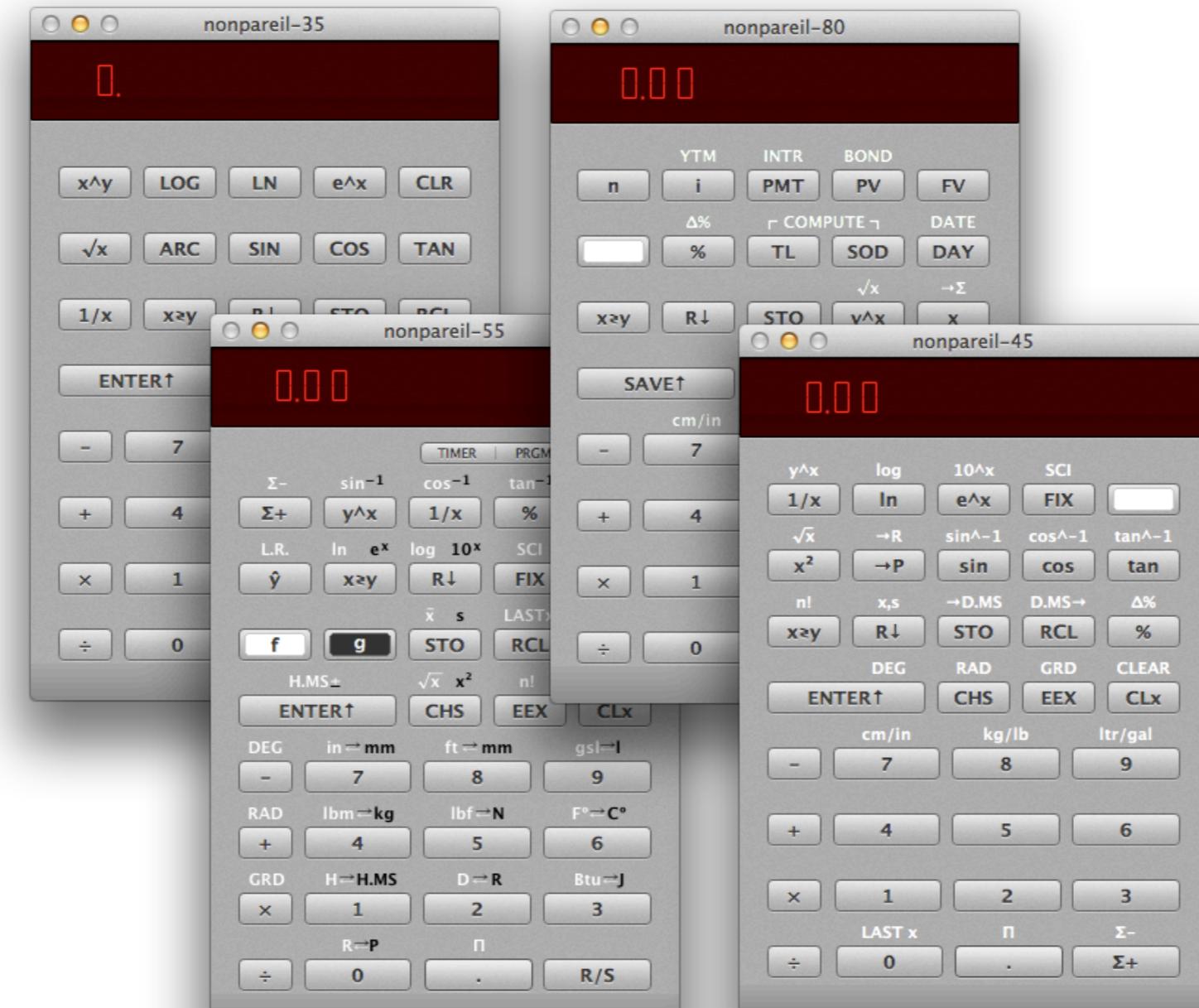
- (a) encapsulamento
- (b) modularità (in particolare, massima separazione tra parte logica e grafica (GUI) del codice)
- (c) estensibilità ed evolvibilità, in particolare mediante polimorfismo
- (d) efficienza e robustezza



3. Funzionalità (peso 20%): quante e quali funzionalità il progetto rende disponibili, e la loro qualità.



4. GUI (peso 20%): utilizzo corretto della libreria Qt; qualità, usabilità e robustezza della GUI.



5. Relazione (peso 20%): chiarezza e qualità della relazione, che verterà sui seguenti aspetti:

- (a) descrizione di tutte le gerarchie di tipi usate
- (b) descrizione **obbligatoria** di tutti gli eventuali usi di codice polimorfo, oppure motivazione **obbligatoria** per l'assenza di codice polimorfo
- (c) se l'applicazione lo richiede, manuale utente della GUI
- (d) se necessario, indicazione di eventuali istruzioni di compilazione ed esecuzione
- (e) indicazione delle **ore effettivamente richieste** dalle diverse fasi progettuali per ogni studente (quindi con suddivisione tra i due studenti in caso di progetto svolto da una coppia): analisi preliminare del problema, progettazione modello e GUI, apprendimento libreria Qt, codifica modello e GUI, debugging, testing. In caso di superamento del previsto monte di 50 ore di lavoro complessivo per persona, giustificazione per le ore in eccesso
- (f) **Per progetti svolti da una coppia di studenti S_1 e S_2 :** dovranno essere prodotte **due diverse e distinte relazioni individuali**. Ogni studente della coppia dovrà quindi scrivere la propria relazione sull'**intero progetto**. Non saranno accettate relazioni identiche per una coppia di studenti. Inoltre, ogni relazione dovrà contenere una esplicita sezione “Suddivisione del lavoro progettuale” che descriva almeno approssimativamente le diverse responsabilità progettuali dei due studenti S_1 e S_2 . Naturalmente dovrà essere esplicitamente dichiarata la coppia di studenti responsabile del progetto consegnato.

Quindi, il progetto dovrà essere **obbligatoriamente** accompagnato da una relazione scritta **individuale**, di **massimo 8 pagine in formato 10pt**. La relazione deve essere presentata come un file PDF di nome (preciso) `relazione.pdf`. La relazione deve anche specificare il sistema operativo di sviluppo e le versioni precise del compilatore e della libreria Qt.



Valutazione

- | | |
|-------------------------------|------------|
| 1. Correttezza: | 10% |
| 2. Orientazione agli oggetti: | 30% |
| 3. Funzionalità: | 20% |
| 4. GUI: | 20% |
| 5. Relazione: | 20% |





3 Esame Orale e Registrazione Voto

La partecipazione all'esame orale è possibile solo dopo:

1. avere superato con successo (cioè, con voto $\geq 18/30$) l'esame scritto
2. avere consegnato il progetto entro la scadenza stabilita, che verrà sempre comunicata nel gruppo Facebook del corso. In caso di progetti svolti in coppia, **ogni studente dovrà fare una consegna distinta del progetto**: quindi il codice sarà in comune alla coppia, mentre ogni consegna sarà caratterizzata dalla propria relazione individuale
3. essersi iscritti alla lista Uniweb dell'esame orale

Il giorno dell'esame orale (nel luogo ed all'orario stabiliti) verrà comunicato l'esito della valutazione del progetto (non vi saranno altre modalità di comunicazione della valutazione del progetto) assieme ad un sintetico **feedback** sui punti deboli riscontrati nella valutazione del progetto. Il feedback per progetti svolti in coppia naturalmente sarà lo stesso, eccetto per la relazione individuale. Tre esiti saranno possibili:

- (A) Valutazione positiva del progetto con registrazione del voto complessivo proposto **con esenzione dell'esame orale**. Nel caso in cui il voto proposto non sia ritenuto soddisfacente dallo studente, sarà possibile rifiutare il voto oppure richiedere l'esame orale, che potrà portare a variazioni in positivo o negativo del voto proposto.
- (B) Valutazione del progetto da completarsi con un **esame orale obbligatorio**. Al termine dell'esame orale, o verrà proposto un voto complessivo sufficiente oppure si dovrà riconsegnare il progetto per un successivo esame orale.
- (C) Valutazione negativa del progetto che comporta quindi la **riconsegna del progetto** per un successivo esame orale (il voto dell'esame scritto rimane valido).

Lo studente che decida di rifiutare il voto finale proposto, con o senza orale, dovrà riconsegnare il progetto per un successivo orale (tranne al quinto orale), cercando quindi di porre rimedio ai punti deboli segnalati nel feedback di valutazione. Il voto sufficiente dell'esame scritto rimane comunque valido. All'eventuale esame orale lo studente dovrà saper motivare **ogni** scelta progettuale e dovrà dimostrare la **piena conoscenza** di ogni parte del progetto, oppure delle parti di propria responsabilità in caso di progetti svolti in coppia.

4.4 Scadenze di consegna

Il progetto dovrà essere consegnato rispettando **tassativamente** le scadenze **ufficiali** (data e ora) previste che verranno rese note tramite le liste Uniweb di iscrizione agli esami scritti ed orali e tramite il gruppo Facebook del corso <https://www.facebook.com/groups/pao17.18>. Approssimativamente la scadenza sarà circa 8-12 giorni prima dell'esame orale.

Per i progetti ritenuti insufficienti, lo studente dovrà consegnare una nuova versione del progetto per un successivo appello orale.

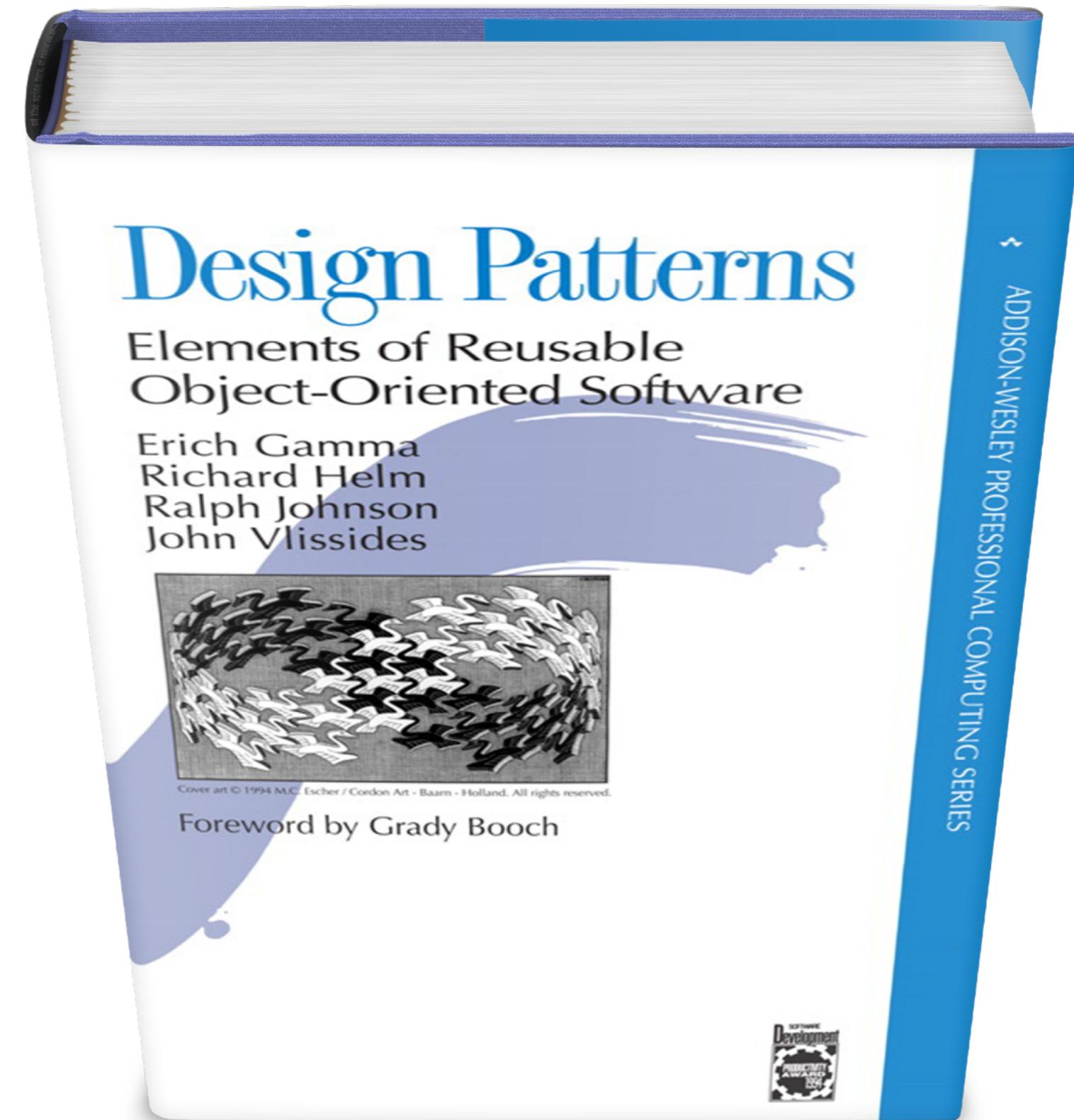
Prima sessione regolare di esami orali: Le date degli esami orali della sessione regolare con relative scadenze tassative di consegna del progetto sono le seguenti:

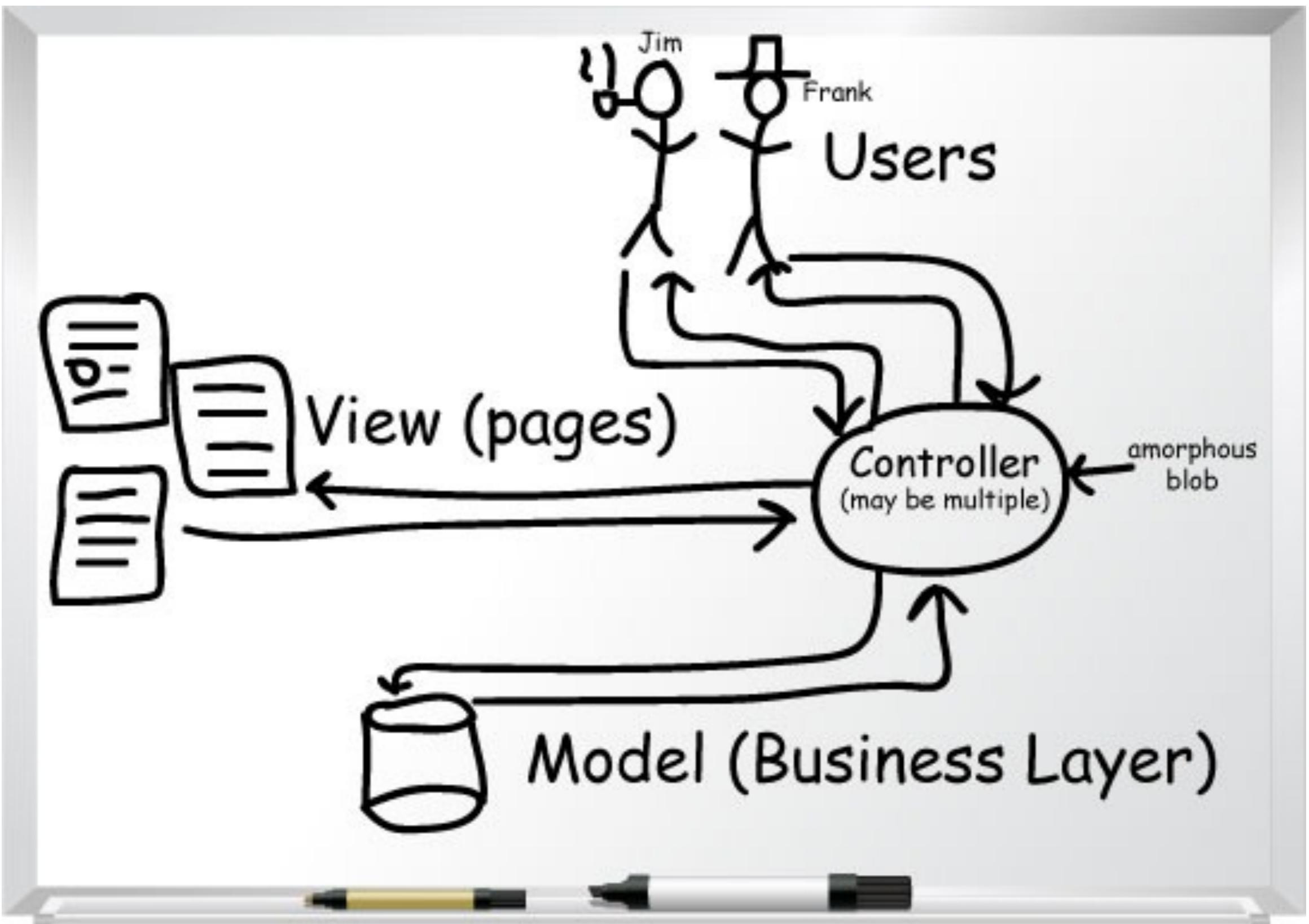
Primo orale: lunedì 12 febbraio 2018, scadenza di consegna: domenica 4 febbraio 2018 ore 23:59

Secondo orale: venerdì 23 febbraio 2018, scadenza di consegna: martedì 13 febbraio 2018 ore 23:59



GANG OF FOUR





Model–view–controller

From Wikipedia, the free encyclopedia

Model–view–controller (MVC) is a [software pattern](#) for implementing [user interfaces](#). It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.^{[1][2]} The central component, the *model*, consists of application data, business rules, logic, and functions. A *view* can be any output representation of information, such as a chart or a diagram. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants. The third part, the *controller*, accepts input and converts it to commands for the model or view.^[3]

Contents [hide]

- [1 Component interactions](#)
- [2 Use in web applications](#)
- [3 History](#)
- [4 See also](#)
- [5 References](#)
- [6 External links](#)

Model-View-Controller (MVC) [edit]

A pattern often used by applications that need the ability to maintain multiple views of the same data. The model-view-controller pattern was until recently a very common pattern especially for graphic user interface programming, it splits the code in 3 pieces. The model, the view, and the controller.

The Model is the actual data representation (for example, Array vs Linked List) or other objects representing a database. The View is an interface to reading the model or a fat client GUI. The Controller provides the interface of changing or modifying the data, and then selecting the "Next Best View" (NBV).

Newcomers will probably see this "MVC" model as wasteful, mainly because you are working with many extra objects at runtime, when it seems like one giant object will do. But the secret to the MVC pattern is not writing the code, but in maintaining it, and allowing people to modify the code without changing much else. Also, keep in mind, that different developers have different strengths and weaknesses, so team building around MVC is easier. Imagine a View Team that is responsible for great views, a Model Team that knows a lot about data, and a Controller Team that see the big picture of application flow, handing requests, working with the model, and selecting the most appropriate next view for that client.

Model/View Programming

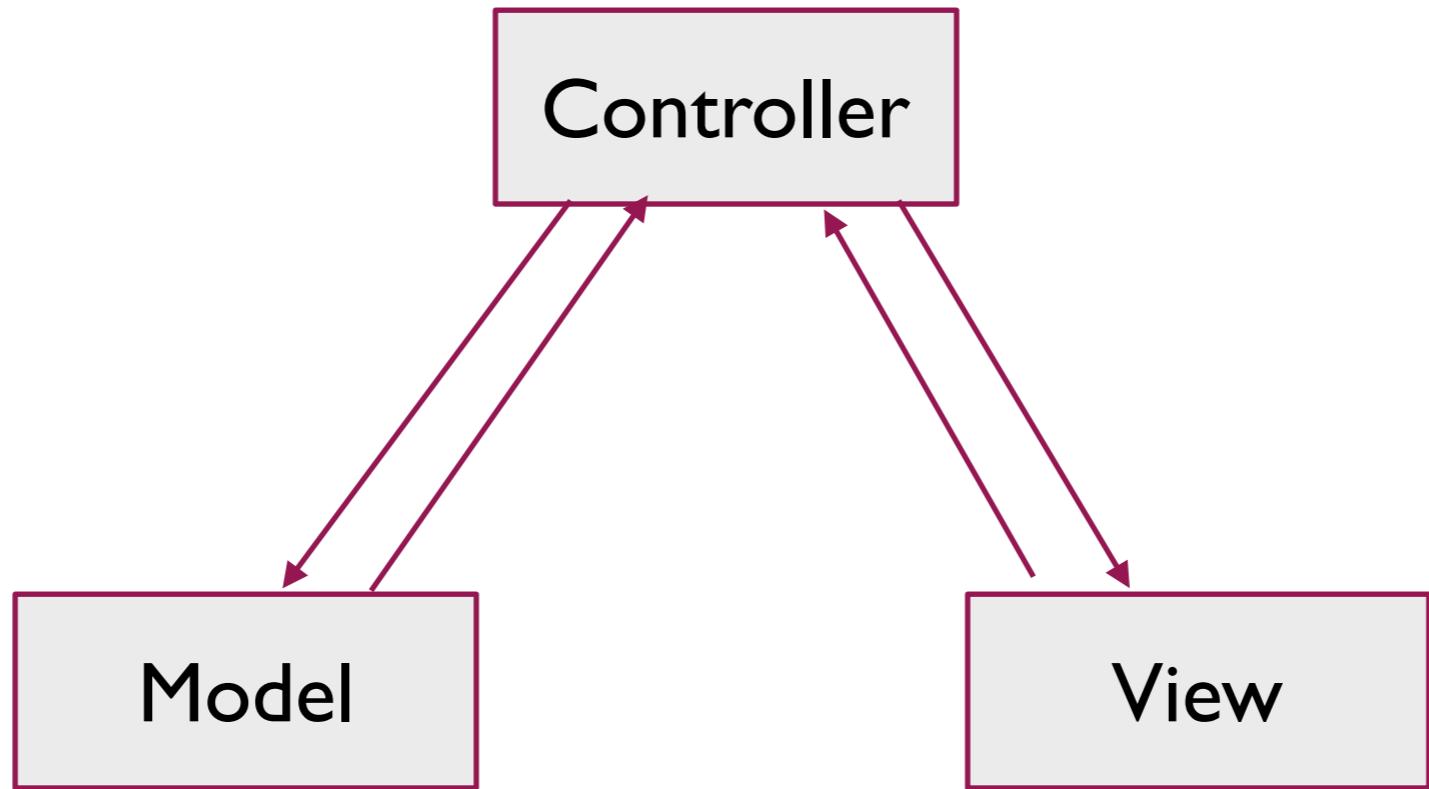
The model/view architecture

Model-View-Controller (MVC) is a design pattern originating from Smalltalk that is often used when building user interfaces. In [Design Patterns](#), Gamma et al. write:

MVC consists of three kinds of objects. The Model is the application object, the View is its screen presentation, and the Controller defines the way the user interface reacts to user input. Before MVC, user interface designs tended to lump these objects together. MVC decouples them to increase flexibility and reuse.

If the view and the controller objects are combined, the result is the [model/view architecture](#). This still separates the way that data is stored from the way that it is presented to the user, but provides a simpler framework based on the same principles. This separation makes it possible to display the same data in several different views, and to implement new types of views, without changing the underlying data structures. To allow flexible handling of user input, we introduce the concept of the *delegate*. The advantage of having a delegate in this framework is that it allows the way items of data are rendered and edited to be customized.

Mini esempio di MVC



Model

```
class GraphModel {  
private:  
    int number;  
  
public:  
    GraphModel(): number(1) {}  
  
    void increaseNumber() { number += 10; }  
  
    int getNumber() const { return number; }  
};
```

View

```
class GraphView {  
private:  
    Button*          button;  
    GraphController* controller;  
  
public:  
    // costruisce view e controller  
    GraphView():  
        button(new Button("Click Me")),  
        controller(new GraphController(this)) {}  
  
    ~GraphView() {  
        delete button;  
        delete controller;  
    }  
  
    // definisce il gestore del button click  
    void setClickHandler(ButtonHandler* bh){  
        button->setHandler(bh);  
    }  
  
    void drawGraph() {  
        // Ottieni i dati dal model per disegnare il grafo  
        int data = controller->getDataDrawing();  
        // Disegna il grafo su data  
        // ...  
    }  
};
```

Controller

```
class GraphController {  
private:  
    GraphModel*    model;  
    GraphView*     view;  
  
public:  
    // non alloca memoria, sola copia di puntatori  
    GraphController(GraphView* v): model(new GraphModel()), view(v) {  
        view->setClickHandler(&onButtonClicked);  
    }  
  
    // trasmette l'input dalla view al model e modifica il model  
    void onButtonClicked() {  
        model->increaseNumber();  
    }  
    // ottiene dati dal model  
    int getDataDrawing() const {  
        return model->getNumber();  
    }  
};
```

creativi(TÀ)

Open World Assumption

Precisione

GO FOR IT !



GOOD LUCK !

I will not write any more bad code
I will not write any more bad code

