

Progetto PAO-2015/2016



Freedom

Progetto PAO-2015/2016

1 Scopo

Lo scopo del progetto è lo sviluppo in C++/Qt di una applicazione a libera scelta che soddisfi alcuni vincoli generali obbligatori e che richieda approssimativamente 50-60 ore di lavoro complessivo. Ogni progetto dovrà essere dotato di un nome, qui genericamente indicato con Freedom.



2 Vincoli Obbligatori

I **vincoli obbligatori** per il progetto Freedom sono i seguenti:

1. Definizione ed utilizzo di una gerarchia G di tipi di altezza ≥ 1 e larghezza ≥ 1 .
2. Definizione di un opportuno contenitore C , con relativi iteratori, che permetta inserimenti, rimozioni, modifiche.
3. Utilizzo del contenitore C per memorizzare oggetti polimorfi della gerarchia G .
4. Il front-end dell'applicazione deve essere una GUI sviluppata nel framework Qt.



2.1 Interfaccia Grafica

Si valuti l'opportunità di aderire al design pattern Model-View-Controller per la progettazione architettonale della GUI. 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 è dotata di una documentazione completa e precisa che sarà 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 curate, dettagliate e user-friendly.



3 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 Freedom prenderà in considerazione i seguenti criteri:

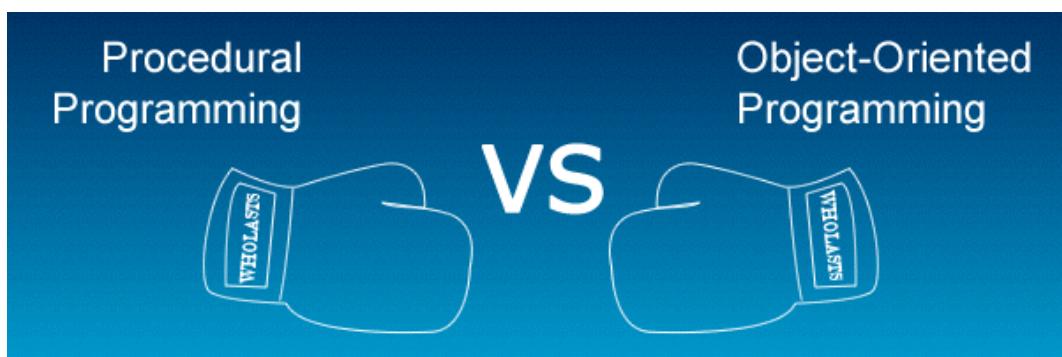
1. **Correttezza** (peso 10%): il progetto deve:

- (a) compilare ed eseguire correttamente (**NB: condizione necessaria** per la valutazione del progetto)
- (b) soddisfare pienamente i vincoli obbligatori
- (c) raggiungere correttamente gli scopi che si prefigge



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

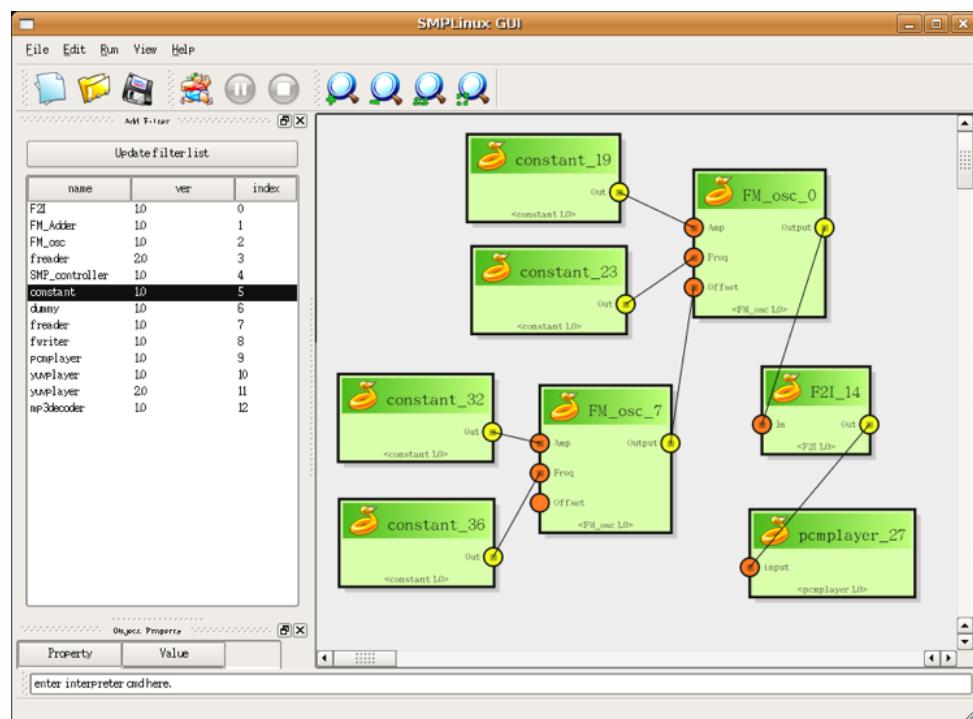
- (a) encapsulamento
- (b) modularità (in particolare, massima separazione tra parte logica e grafica (GUI) del codice)
- (c) estensibilità ed evolvibilità, in particolare mediante polimorfismo
- (d) efficienza e robustezza



3. Funzionalità (peso 25%): 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 15%): chiarezza e qualità della relazione sui seguenti aspetti:

- (a) scopo del progetto
- (b) descrizione della gerarchia G : utilità e ruolo dei tipi di G
- (c) descrizione dell'uso di codice polimorfo
- (d) manuale utente della GUI, se l'applicazione lo richiede

Il progetto dovrà quindi essere obbligatoriamente accompagnato da una relazione scritta, 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.



- | | |
|--------------------------------------|------------|
| 1. Correttezza: | 10% |
| 2. Orientazione agli oggetti: | 30% |
| 3. Funzionalità: | 25% |
| 4. GUI: | 20% |
| 5. Relazione: | 15% |



5.1 Singleness

Il progetto dovrà essere realizzato da ogni singolo studente in modo **indipendente** da terze persone.



4 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 Freedom entro la scadenza stabilita, che verrà sempre comunicata nel gruppo Facebook del corso
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. 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. Si ricorda inoltre che all'eventuale esame orale lo studente dovrà saper motivare **ogni** scelta progettuale e dovrà dimostrare la **piena conoscenza** di ogni parte del progetto.

5.5 Scadenze di consegna

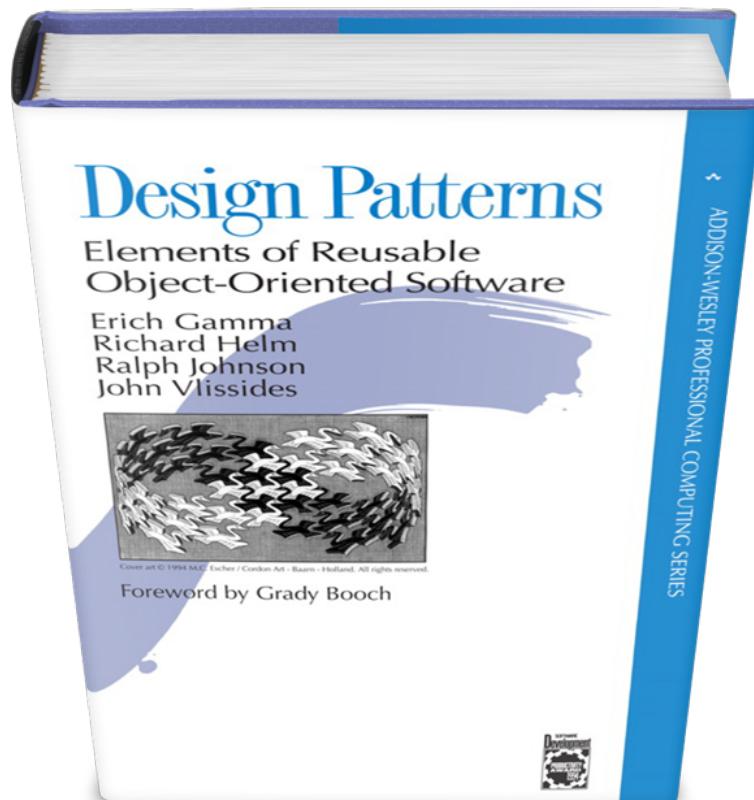
Il progetto dovrà essere consegnato rispettando **tassativamente** le scadenze **ufficiali** (data e ora) previste che verranno rese note tramite le liste Uniweb di iscrizione agli esami scritti ed orali e tramite il gruppo Facebook del corso <https://www.facebook.com/groups/pao15.16>. Approssimativamente la scadenza sarà circa 8-10 giorni prima dell'esame orale.

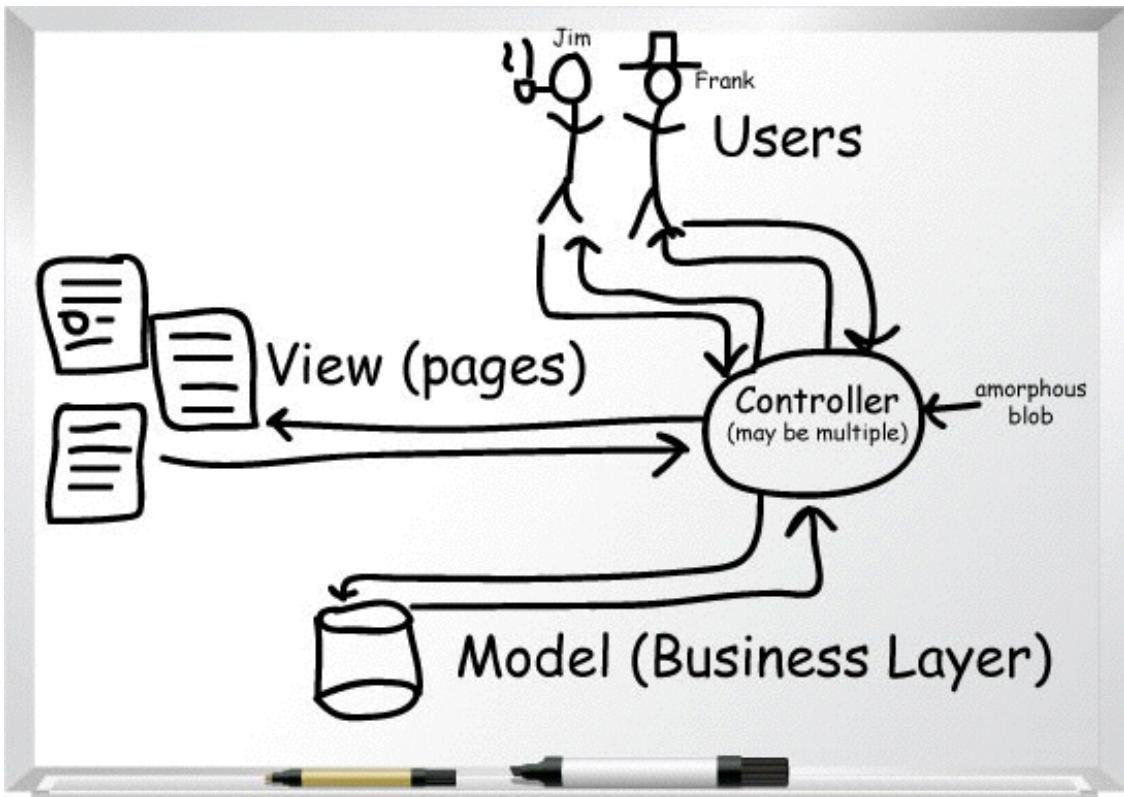
Per i progetti ritenuti insufficienti, lo studente dovrà consegnare una nuova versione del progetto per un successivo appello orale.

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: martedì 16 febbraio 2016, scadenza di consegna: domenica 7 febbraio 2016 ore 23:59

Secondo orale: venerdì 26 febbraio 2016, scadenza di consegna: mercoledì 17 febbraio 2016 ore 23:59





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.

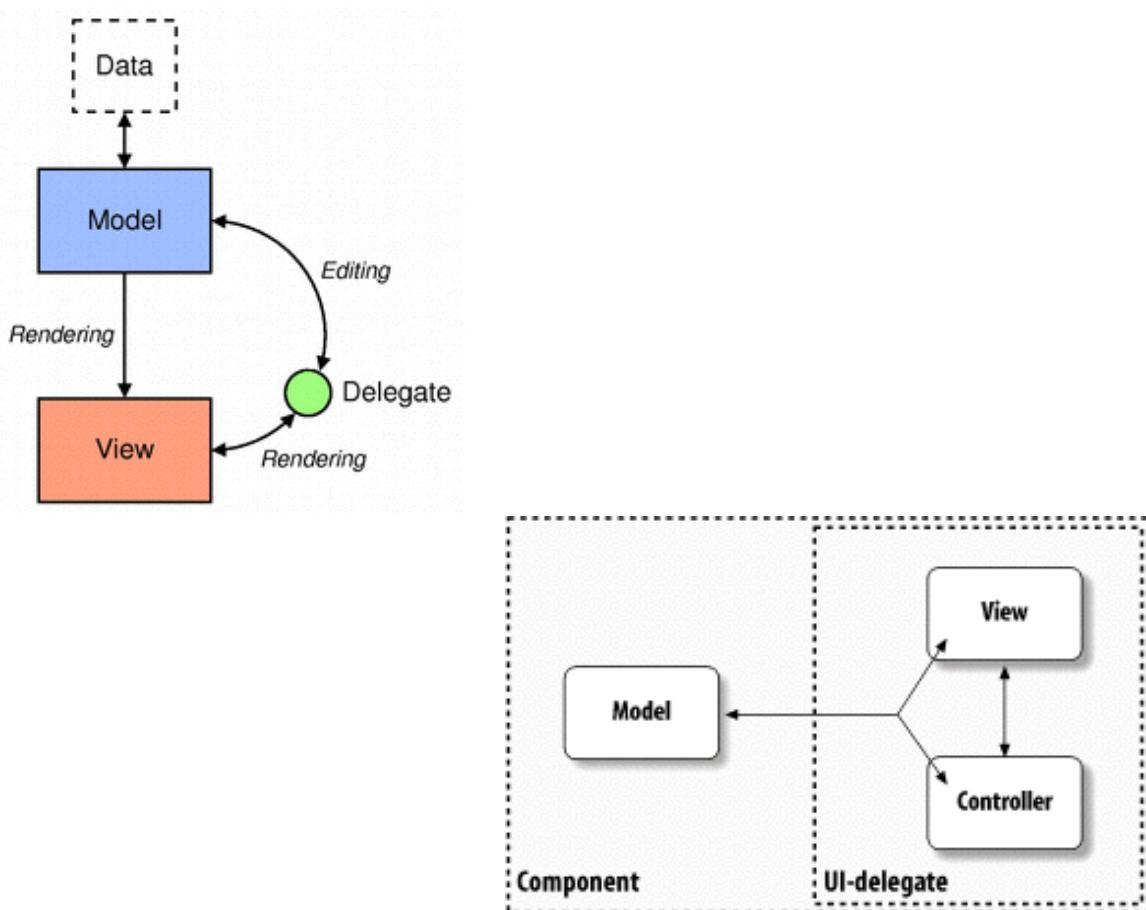
The Model is the actual data representation (for example, Array vs Linked List) or other objects representing a database. The View is an interface to reading the model or a fat client GUI. The Controller provides the interface of changing or modifying the data, and then selecting the "Next Best View"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

Introduction to Model/View Programming

Qt contains a set of item view classes that use a model/view architecture to manage the relationship between data and the way it is presented to the user. The separation of functionality introduced by this architecture gives developers greater flexibility to customize the presentation of items, and provides a standard model interface to allow a wide range of data sources to be used with existing item views. In this document, we give a brief introduction to the model/view paradigm, outline the concepts involved, and describe the architecture of the item view system. Each of the components in the architecture is explained, and examples are given that show how to use the classes provided.

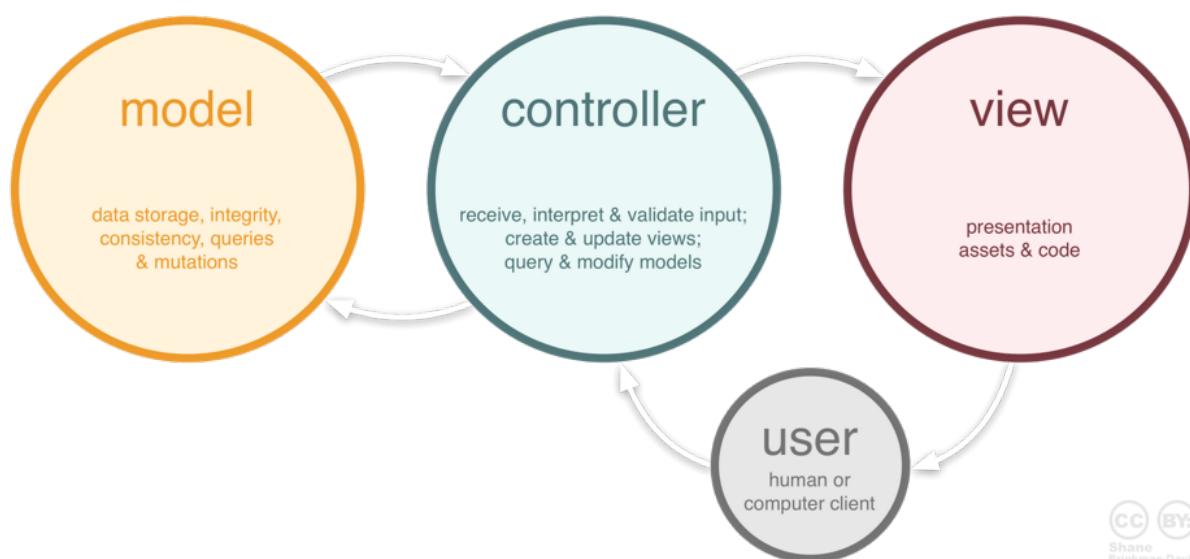


The model/view architecture

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.



Esempio concettuale di MVC

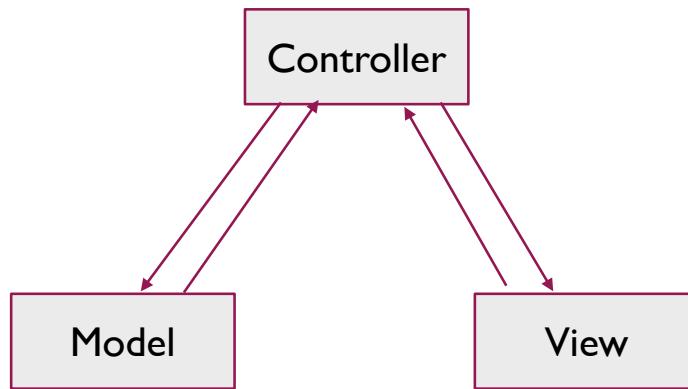
```
class StudentModel {  
private:  
    string matricola;  
    string nome;  
public:  
    string getMatricola() const { return matricola; }  
    void setMatricola(string m) { matricola = m; }  
  
    string getNome() const { return nome; }  
    void setNome(string n) { nome = n; }  
};
```

```
class StudentView {  
private:  
    ostream os;  
public:  
    StudentView(): os(std::cout) {}  
    // metodo di istanza con una view di invocazione  
    void printStudentDetails(StudentModel s) const {  
        os << "STUDENTE:" << std::endl;  
        os << "Nome: " << s.getNome() << std::endl;  
        os << "Matricola: " << s.getMatricola() << std::endl;  
    }  
};
```

```
class StudentController {  
private:  
    StudentModel model;  
    StudentView view;  
public:  
    StudentController(StudentModel m, StudentView v): model(m), view(v) {}  
  
    void setNomeStudente(string n){ model.setNome(n); }  
    string getNomeStudente() const { return model.getNome(); }  
  
    void setMatricolaStudente(string m){ model.setMatricola(m); }  
    string getMatricolaStudente() const { return model.getMatricola(); }  
  
    void updateView() const { view.printStudentDetails(model); }  
};
```

Esempio concettuale di MVC

```
class MVCPatternDemo {  
private:  
    static StudentModel retrieveStudentFromDB(string m) {  
        StudentModel student;  
        student.setMatricola(m);  
        student.setNome("Roberto");  
        return student;  
    }  
  
public:  
    MVCPatternDemo() {  
        // Crea un model recuperando uno studente dal DB  
        StudentModel model = retrieveStudentFromDB("999");  
        // Crea una view per l'output su cout dei dettagli dello studente  
        StudentView view;  
        // Crea un controller su model e view  
        StudentController controller(model, view);  
  
        controller.updateView();  
  
        controller.setNomeStudente("Paolo");  
        controller.updateView();  
    }  
};  
  
int main() {  
    MVCPatternDemo app;  
}
```



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;  
    GraphModel*       model;  
    GraphController* controller;  
  
public:  
    // costruisce model e controller  
    GraphView():  
        button(new Button("Click Me")),  
        model(new GraphModel()),  
        controller(new GraphController(model,this)) {}  
  
    ~GraphView() {  
        delete button;  
        delete model;  
        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(GraphModel* m, GraphView* v): model(m), 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(TA)

Open World Assumption

Precisione

IMMAGINARE

=

In Me Mago Agere

Lascio agire il mago che c'è in me

ENJOY!

