POO

Sumar
OO

Cuprins

- Cum proiectam o aplicatie OO
  - arhitectura MVC revizuita
- Studiu de caz: interfata grafica utilizand FLTK
- O trecere revista a conceptelor si principiilor OO

Proiectarea arhitecturii aplicatiei

(Pattern-Oriented Software Architecture For Dummies, by Robert Hanmer)

1. “Select a component to be refined.
2. Identify the requirements of that component and the requirements for its interactions.
   - For each class or component, write a CRC card.
   - Use the scenarios in your use cases to guide you through the CRC cards.”

CRC card

Nume:	Colaboratori:
Responsabilitati:	

Proiectarea arhitecturii aplicatiei (cont)

3. “Search for an existing architectural style or pattern that fits the requirements and interactions that you identified in Step 2.
4. Use the pattern that you’ve matched to your problem to guide the arrangement of your classes and components.
5. Iterate through the components, repeating Steps 2–4 for each one.”

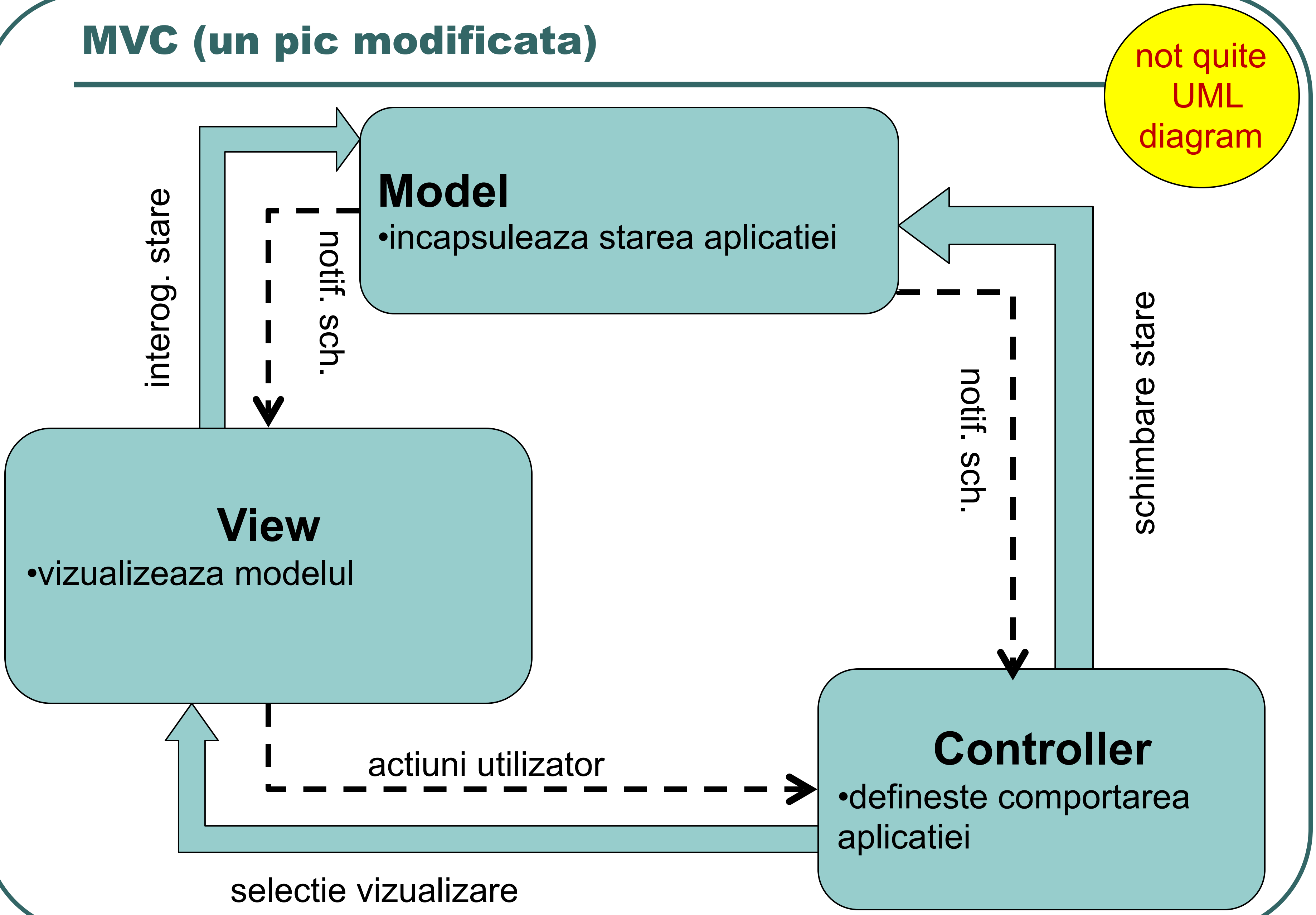
Documentarea muncii

- “Take some notes while you’re defining the architecture. Keep a record of the key decisions.
- If you make any assumptions while you’re developing your architecture, keep track of them so that you can validate them with your customer or client.
- When you make decisions after trading off one alternative with another, make notes to help you remember why you selected one alternative over another.
- If you reject some ideas because they won’t work for this problem, write down the ones you rejected and the reasons why you rejected them. You’ll be surprised how often someone will second-guess the decision.”

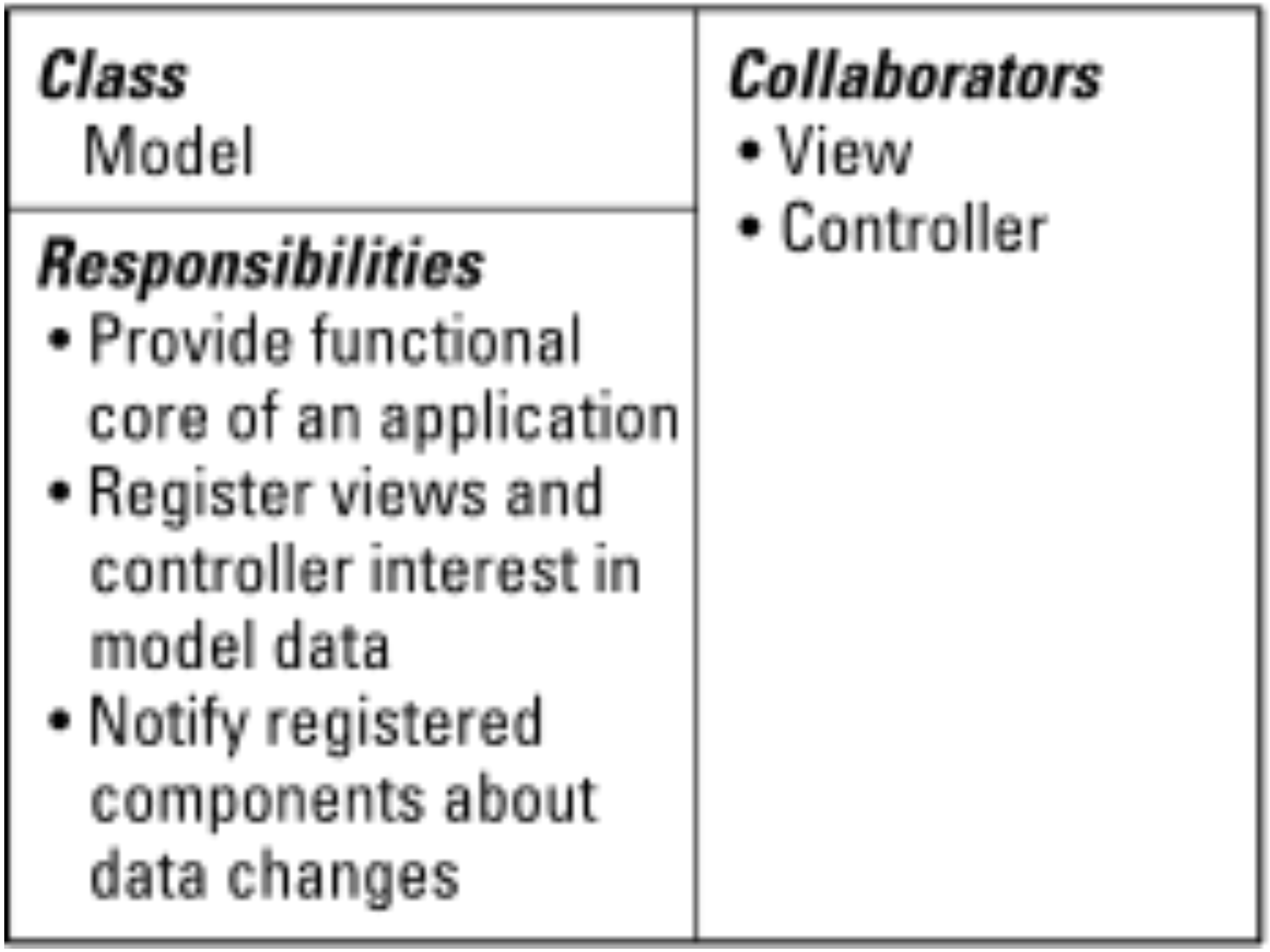
POO

MVC
revizuit

MVC (un pic modificata)

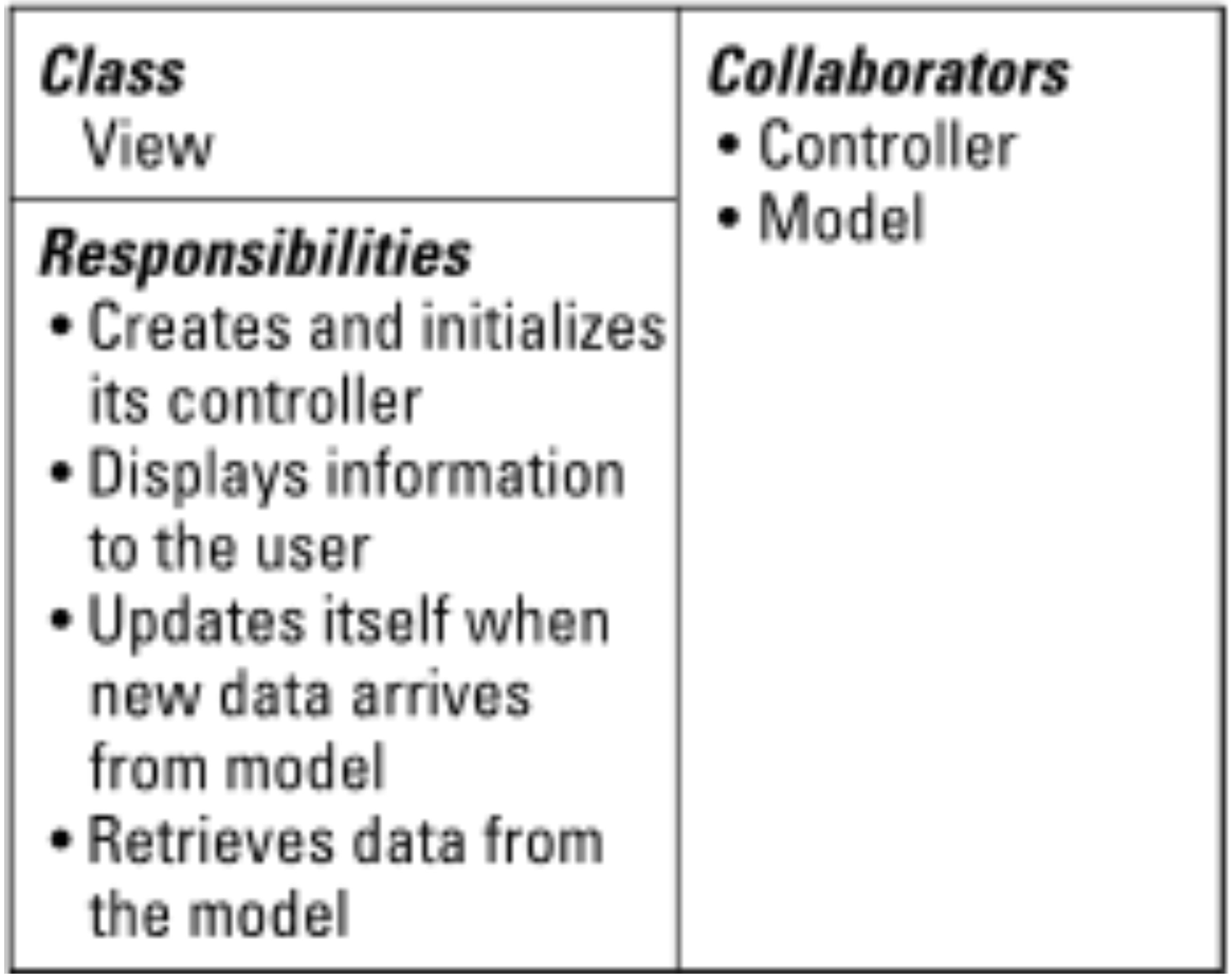


Componenta Model



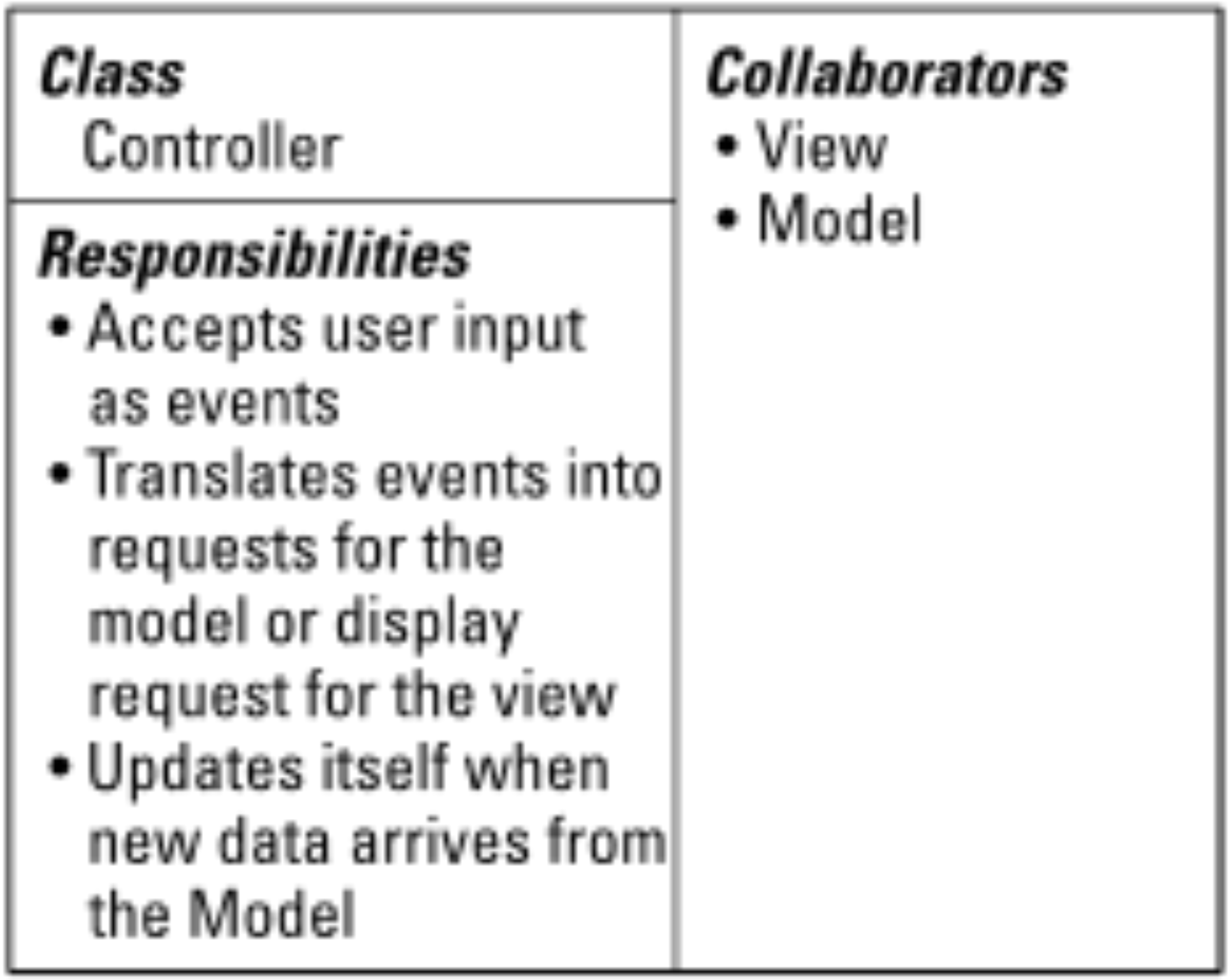
Class Model	***Collaborators***
Responsibilities • Provide functional core of an application • Register views and controller interest in model data • Notify registered components about data changes	• View • Controller

Componenta View



Class View	***Collaborators***
Responsibilities • Creates and initializes its controller • Displays information to the user • Updates itself when new data arrives from model • Retrieves data from the model	• Controller • Model

Componenta Controller



Class
Controller

Responsibilities
- Accepts user input as events
- Translates events into requests for the model or display request for the view
- Updates itself when new data arrives from the Model

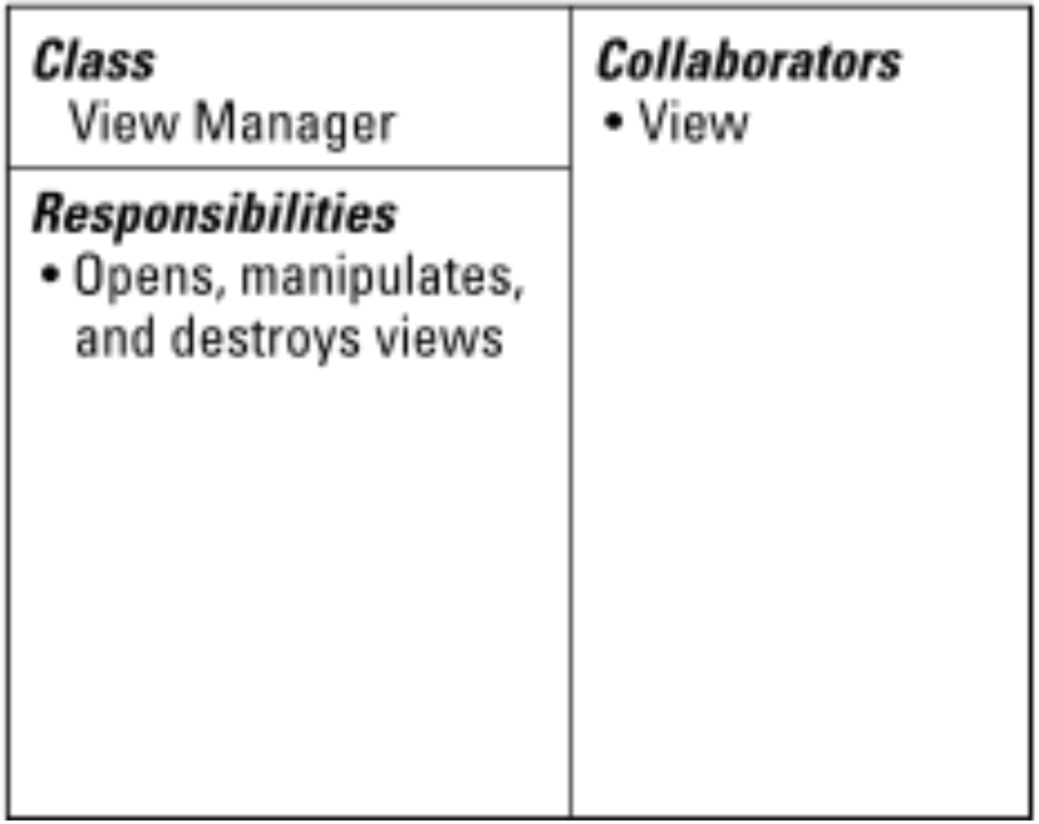
Collaborators
- View
- Model

Implementarea MVCului

- “Step 1: Separate the core functionality from the UI behavior
  - Analyze the application domain of the problem you’re solving, and answer the following questions:
    - ✓ What are the core data parts?
    - ✓ What computational functions are performed on the data?
    - ✓ What is the system’s desired input?
    - ✓ What is the system’s desired output?
- Step 2: Build the change-propagation mechanism
- Step 3: Design and implement the views
- Step 4: Design and build the controllers
- Step 5: Build the relationship between the view and controller”

Implementarea MVCului

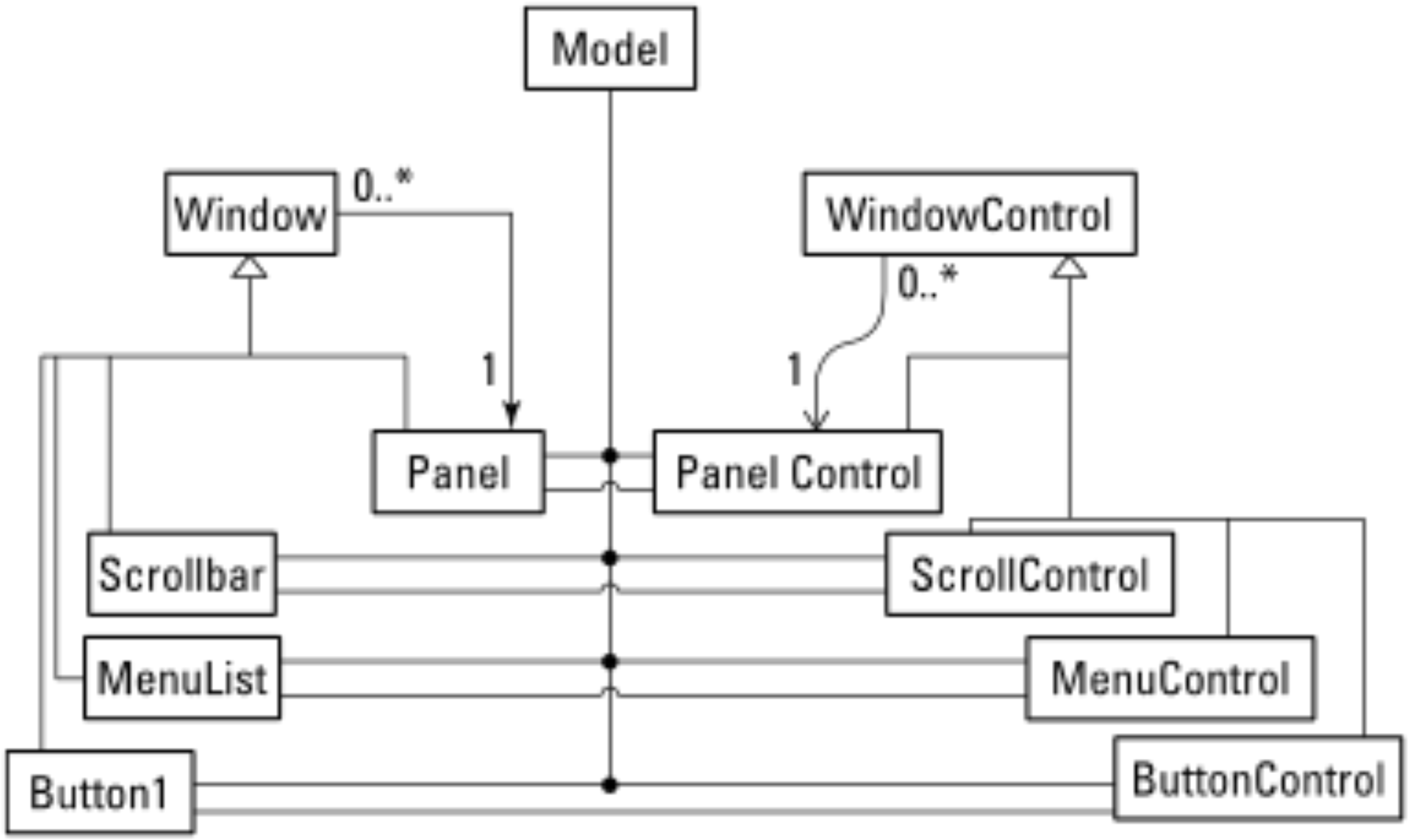
- “Step 6: Get the MVC started
- Step 7: Create dynamic views
- Step 8: Create changeable controllers”



Class View Manager	***Collaborators***
Responsibilities • Opens, manipulates, and destroys views	• View

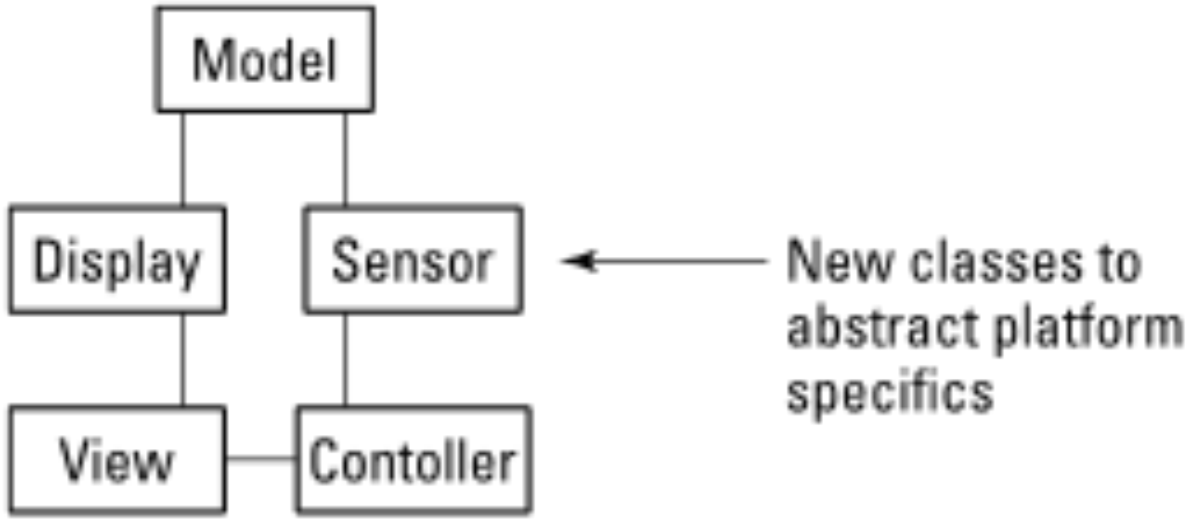
Implementarea MVCului

- “Step 9: Design the infrastructure for hierarchical views and controllers”



Implementarea MVCului

- “Step 10: Remove system dependencies”



POO

- Cum implementam interfata de utilizare grafica

(Bjarne Stroustrup www.stroustrup.com/Programming)

Alternative I/O

- “Use console input and output
  - A strong contender for technical/professional work
  - Command line interface
  - Menu driven interface
- Graphic User Interface
  - Use a GUI Library
  - To match the “feel” of Windows/Mac applications
  - When you need drag and drop, WYSIWYG
  - Event driven program design
  - A web browser – this is a GUI library application
    - HTML / a scripting language
    - For remote access (and more)”

Common GUI tasks

- “Titles / Text
  - Names
  - Prompts
  - User instructions
- Fields / Dialog boxes
  - Input
  - Output
- Buttons
  - Let the user initiate actions
  - Let the user select among a set of alternatives
  - e.g. yes/no, blue/green/red, etc.”

Common GUI tasks

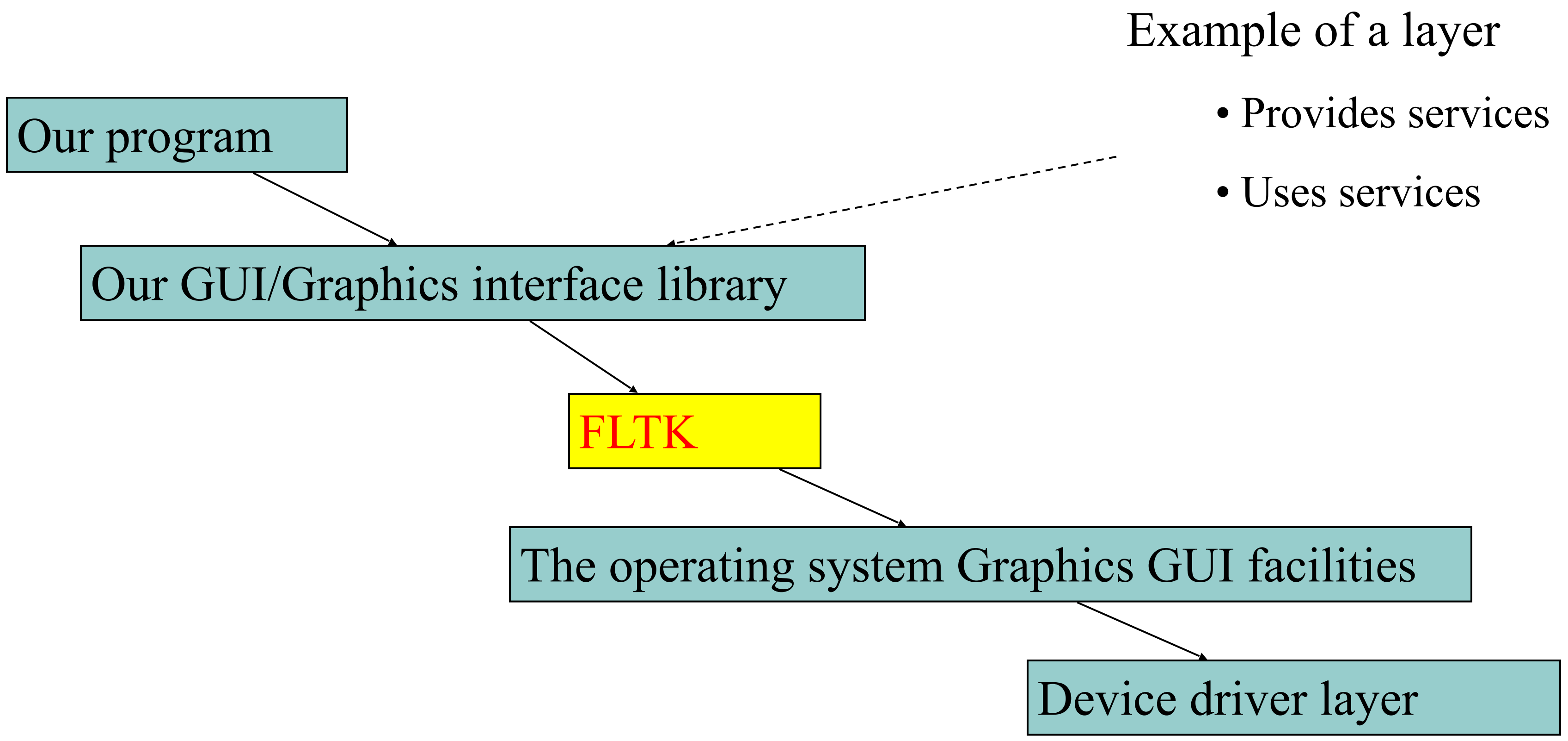
- “Display results
  - Shapes
  - Text and numbers
- Make a window “look right”
  - Style and color
  - Note: our windows look different (and appropriate) on different systems
- More advanced
  - Tracking the mouse
  - Dragging and dropping
  - Free-hand drawing”

GUI

- “From a programming point of view GUI is based on two techniques
  - Object-oriented programming
    - For organizing program parts with common interfaces and common actions
  - Events
    - For connecting an event (like a mouse click) with a program action”

Layers of software

- “When we build software, we usually build upon existing code”



Ce este FLTK

Din manual:

- “The Fast Light Tool Kit (”FLTK”, pronounced ”fulltick”) is a cross-platform C++ GUI toolkit for UNIX ®/Linux® (X11), Microsoft® Windows®, and Apple® OS X®.
- FLTK provides modern GUI functionality without the bloat and supports 3D graphics via OpenGL ® and its built-in GLUT emulation.
- It was originally developed by Mr. Bill Spitzak and is currently maintained by a small group of developers across the world with a central repository in the US.”

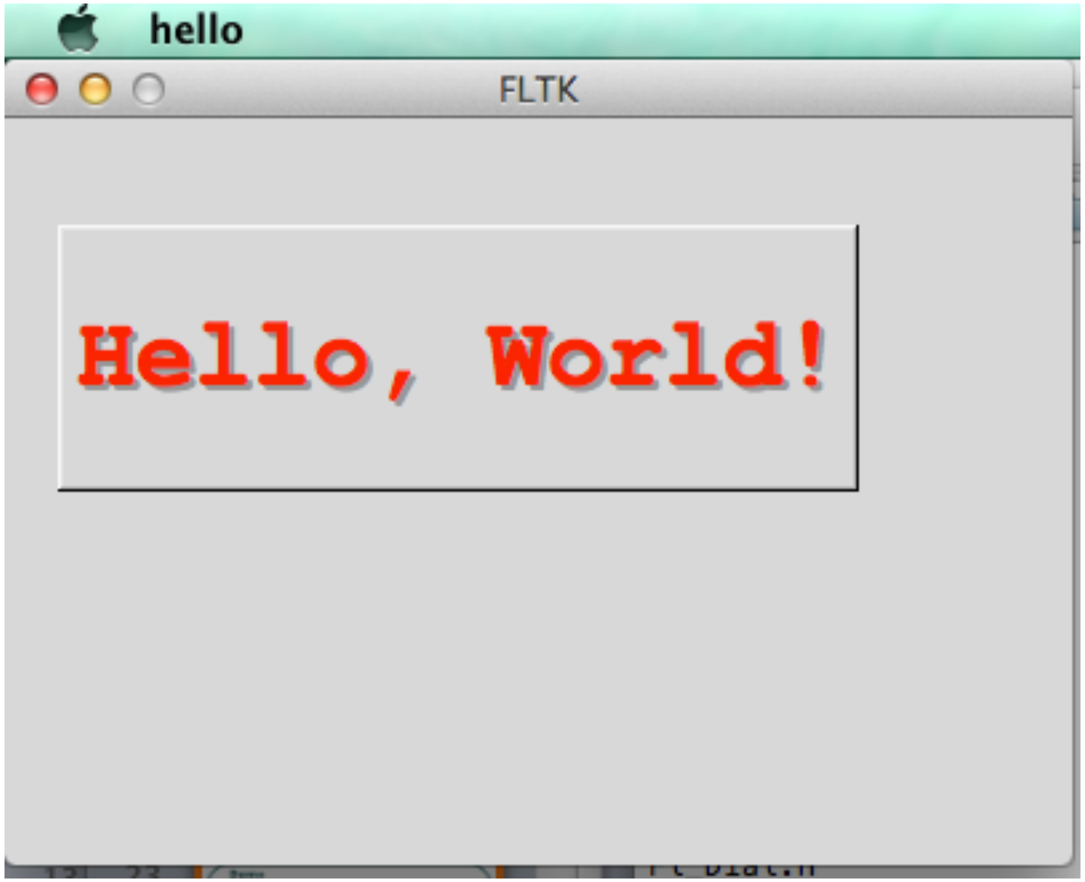
FLTK - instalare

- se descarca aplicatia de al adresa
  http://www.fltk.org/software.php
- se urmeaza instructiunile din fisierul RERADME corespunzator platfoermei
  - Linux/Unix: README.Unix.txt
  - Windows: README.MSWindows.txt
  - Mac OS: README.OSX.txt
  - CMake
- documentatie
  http://www.fltk.org/doc-1.3/index.html
- … sau se instaleaza local urmand instructiunile din documentation/README

Primul Program FLTK

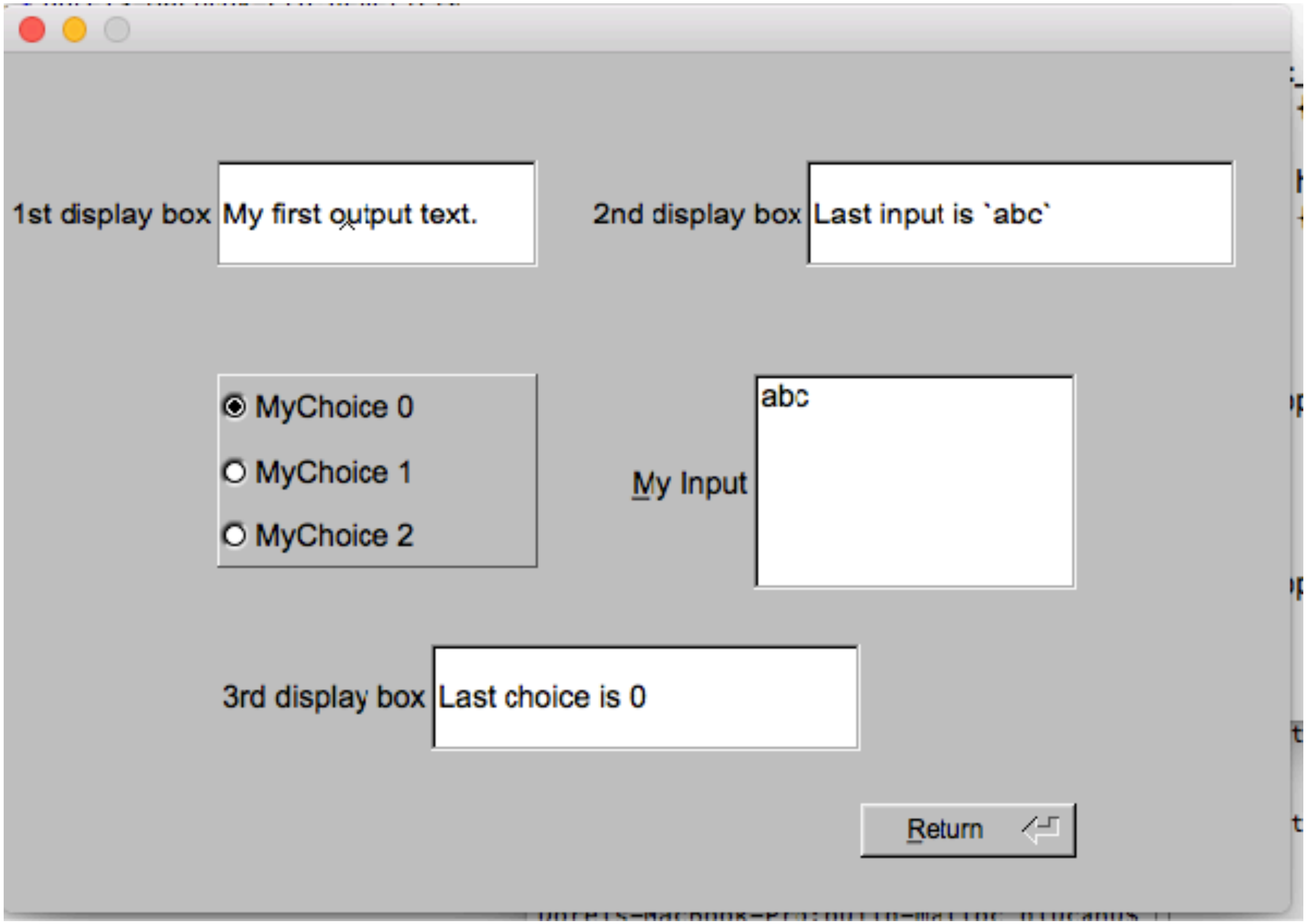
```
#include <FL/Fl.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Box.H>
int main(int argc, char **argv) {
  Fl_Window *window = new Fl_Window(400,280);
  Fl_Box *box = new Fl_Box(20,40,300,100,"Hello, World!");
  box->box(FL_UP_BOX);
  box->labelfont(FL_BOLD+FL_COURIER);
  box->labelsize(36);
  box->labelcolor(FL_RED);
  box->labeltype(FL_SHADOW_LABEL);
  window->end();
  window->show(argc, argv);
  return Fl::run();
}
```

Demo



```
$ g++ hello.cxx `fltk-config --use-forms --ldflags`
```

My GUI demo

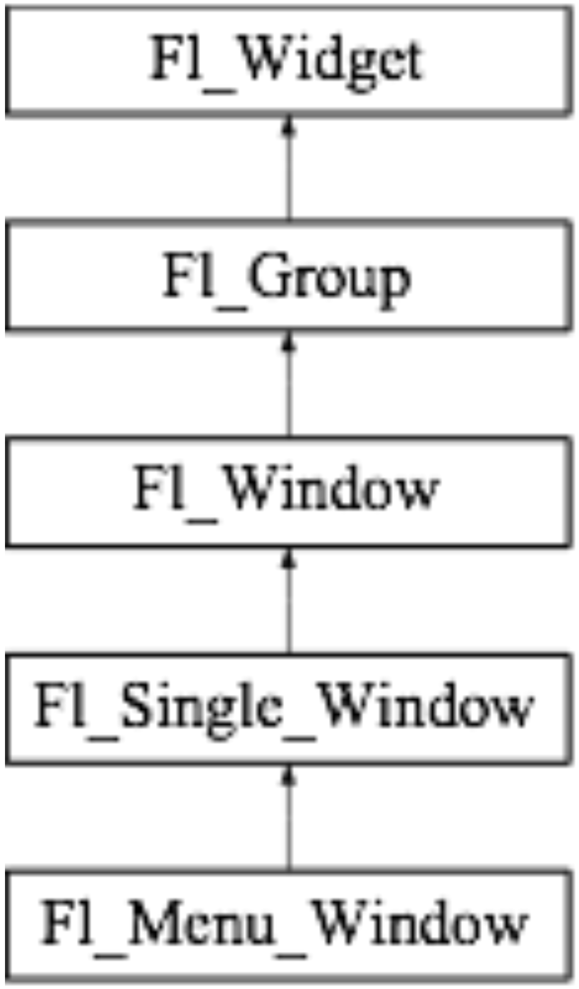


1st display box
My first output text.
2nd display box
Last input is `abc`
MyChoice 0
MyChoice 1
MyChoice 2
My Input
abc
3rd display box
Last choice is 0
Return

My GUI: How?

- We saw buttons, input boxes and an outbox in a window
  - How do we define a window?
  - How do we define buttons?
  - How do we define input and output boxes?
- Click on a button and something happens
  - How do we program that action?
  - How do we connect our code to the button?
- You type something into a input box
  - How do we get that value into our code?
- We saw output in the output box
  - How do we get the values there?

FLTK Window

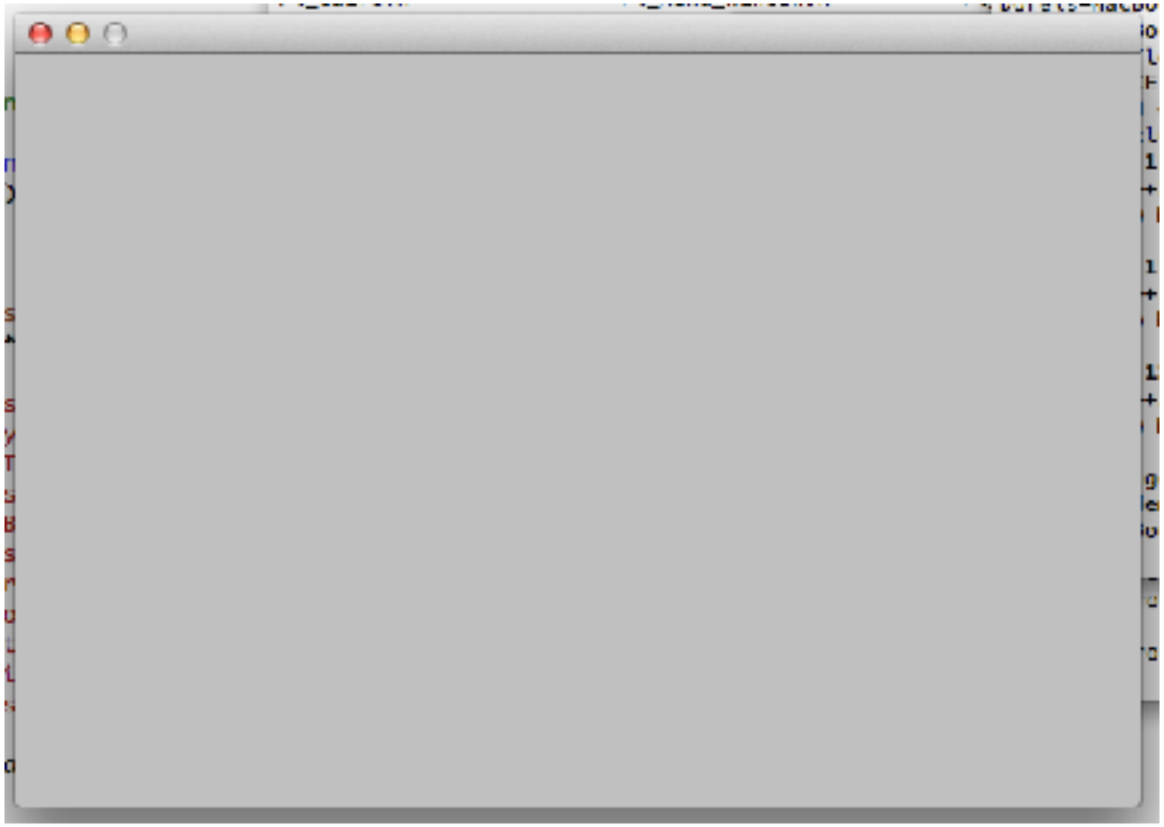


My Window

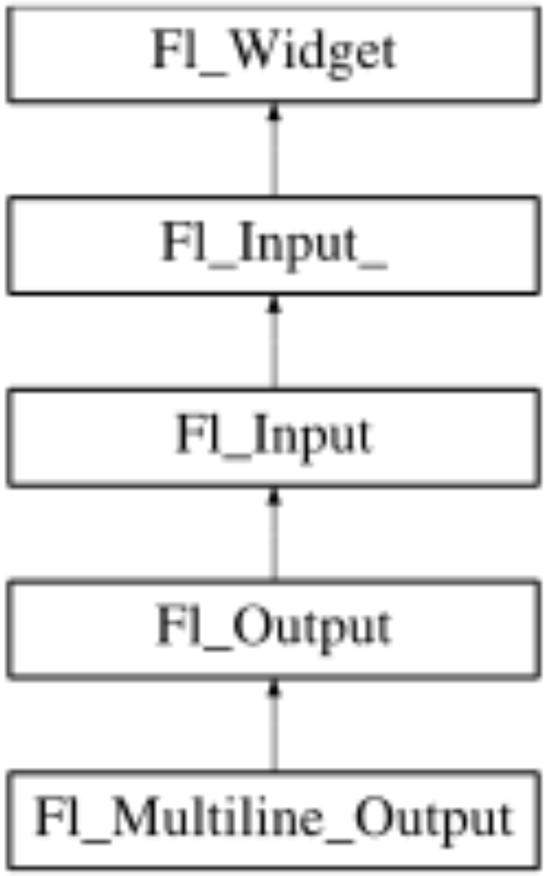
```
class MyWindow : Fl_Window {
 public:
 MyWindow(Point pos, int w, int h, const char *title = 0)
       : Fl_Window(pos.getX(), pos.getY(), w, h, title) {}
 MyWindow(int w, int h, const char *title = 0)
       : Fl_Window(w, h, title) {}

  int display() {
    this->end();     // inherited
    this->show();   // inherited
    return Fl::run();
  }
};
```

Demo

```
Point posMainWindow(100, 200);
 MyWindow* mainwindow =
         new MyWindow(posMainWindow, 600, 400);
```



FLTK Output

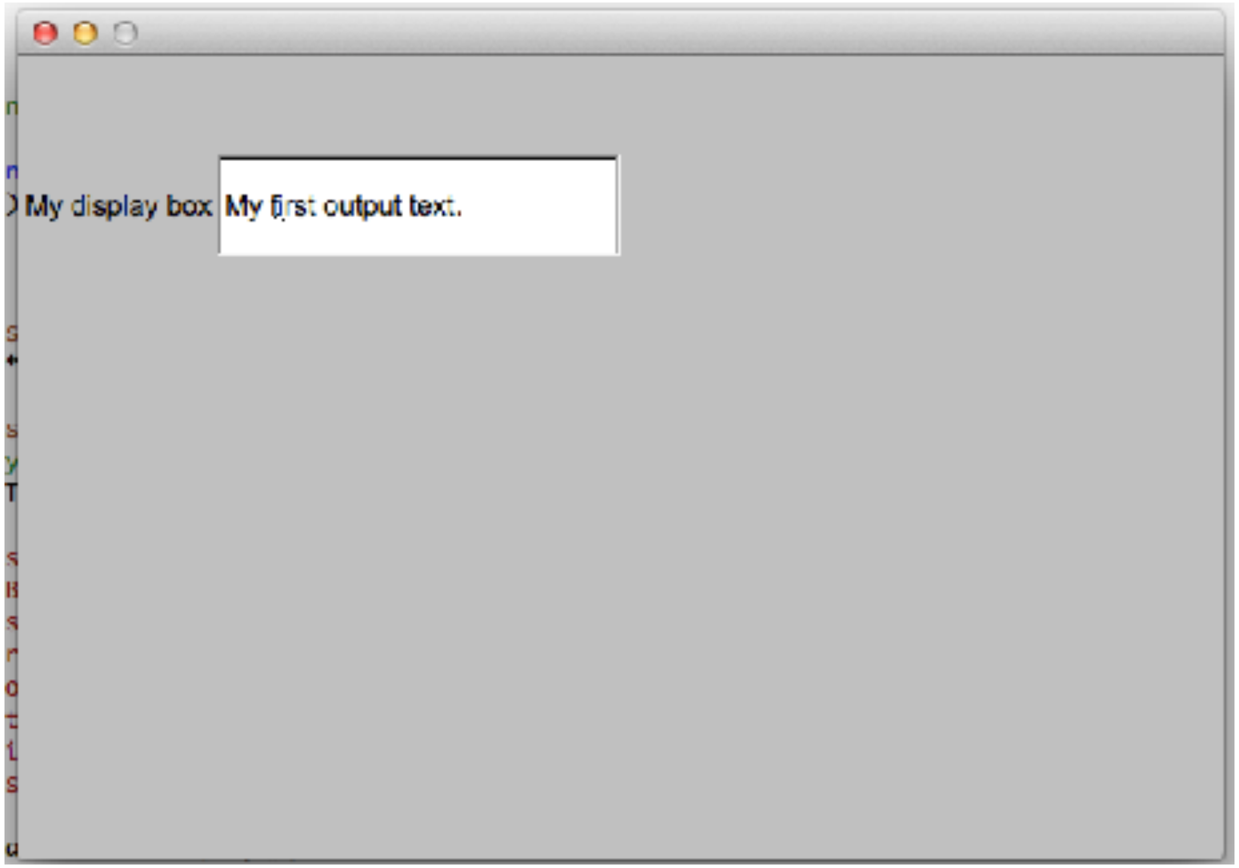


MyDisplayBox

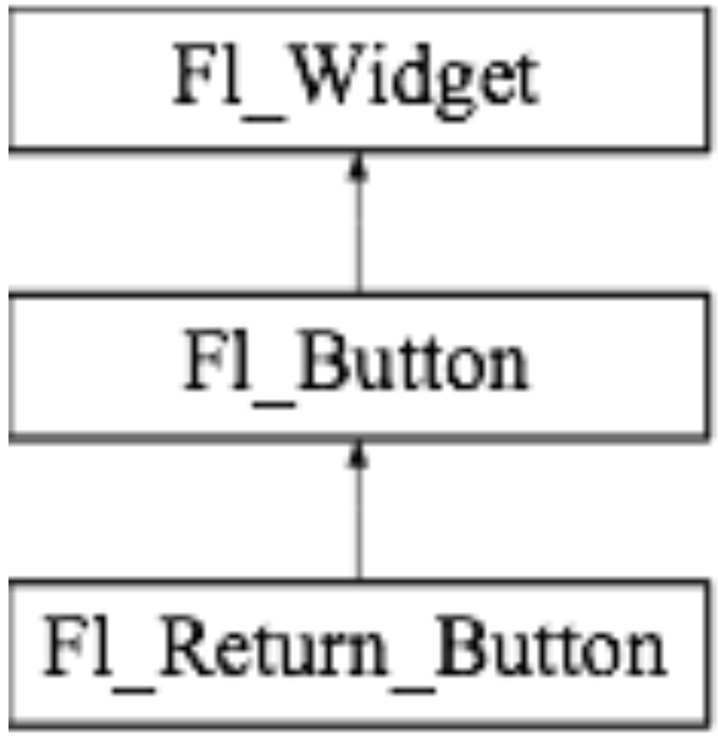
```
class MyDisplayBox : Fl_Output {
 public:
 MyDisplayBox(Point pos, int w, int h, const char *label = 0) :
Fl_Output(pos.getX(), pos.getY(), w, h, label) {}

  void setText(std::string txt) {
    this->value(txt.c_str());   // inherited
    this->redraw();     // inherited
  }
};
```

Demo

```
Point posFirstDB(100, 50);
MyDisplayBox *adb =
   new MyDisplayBox(posFirstDB, 200, 50, "My display box");
```


My display box My first output text.

FLTK Button



Callbacks

Din manual:

- Callbacks are functions that are called when the value of a widget changes.
- A callback function is sent a Fl Widget pointer of the widget that changed and a pointer to data that you provide:

```
void xyz_callback(Fl Widget *w, void *data) f {
  ...
}
```

- The callback() method sets the callback function for a widget. You can optionally pass a pointer to some data needed for the callback:

```
int xyz_data;
button->callback(xyz_callback, &xyz_data);
```

Callbacks

Din manual:

- Normally callbacks are performed only when the value of the widget changes. You can change this using the Fl_Widget::when() method:

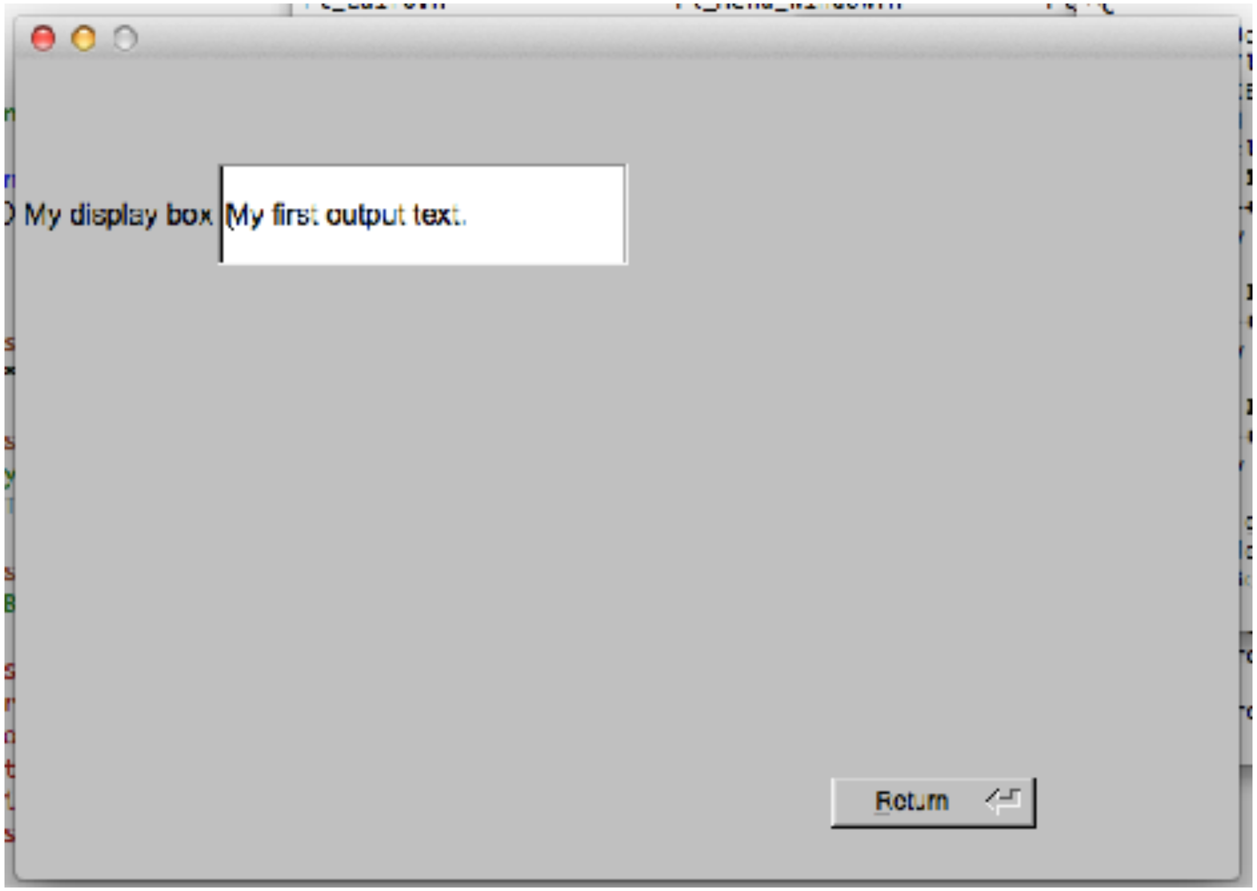
```
button->when(FL_WHEN_NEVER);
button->when(FL_WHEN_CHANGED);
button->when(FL_WHEN_RELEASE);
```

MyReturnButton

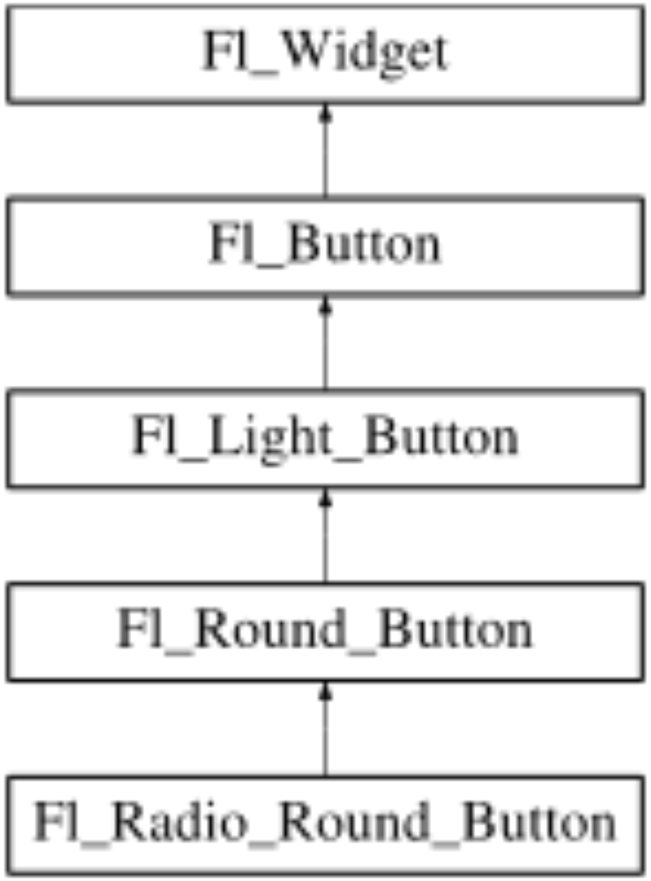
```
class MyReturnButton : Fl_Return_Button {
 public:
  MyReturnButton(Point pos, int w, int h, const char *label = 0)
             : Fl_Return_Button(pos.getX(), pos.getY(), w, h, label) {
    this->tooltip("Push Return button to exit");
    this->labelsize(12);
    this->callback((Fl_Callback*) ret_cb);
    this->redraw();
  }
 private:
  static void ret_cb(Fl_Button *b, void *) {
    exit(0);
  }
};
```

Demo

```
Point posRet(400, 350);
MyReturnButton *ret =
    new MyReturnButton(posRet, 100, 25, "&Return");
```



FLTK Round Button



MyRadioButton

```
class MyRadioButton : Fl_Radio_Round_Button {
 public:
  MyRadioButton(Point pos, int w, int h, std::string slabel) :
     Fl_Radio_Round_Button(pos.getX(), pos.getY(), w, h)
  {
    this->tooltip("Radio button, only one button is set at a time.");
    this->down_box(FL_ROUND_DOWN_BOX);
    this->callback((Fl_Callback*) radio_button_cb);
    this->when(FL_WHEN_CHANGED);
    sprintf(blabel, "%s", slabel.data());
    this->label(blabel);
    this->redraw();
  }
```

MyRadioButton

```
private:  char blabel[256];
  static void radio_button_cb(MyRadioButton *b, void *) { }
};
```

Grup de butoane radio

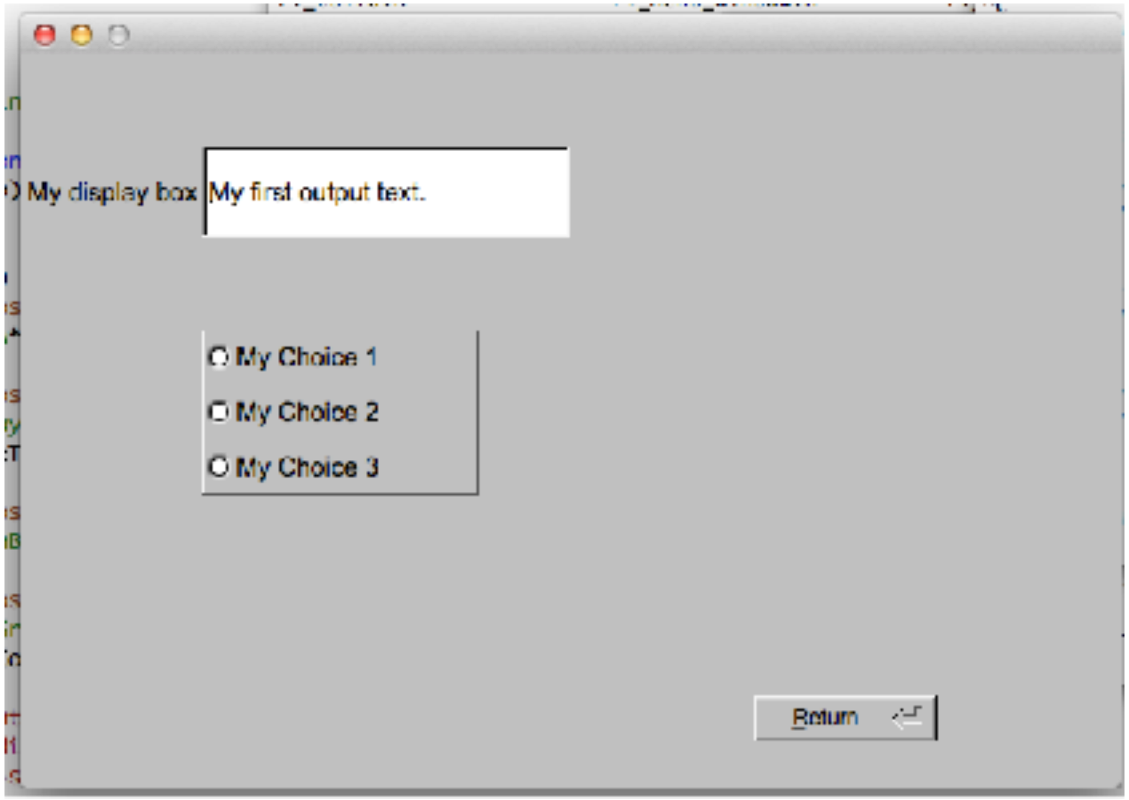
```
class MyRadioGroup : Fl_Group {
 public:
 MyRadioGroup(Point pos, int w, int h,  std::string slabel, int no) :
   Fl_Group(pos.getX(), pos.getY(), w, h)
 {
    this->box(FL_THIN_UP_FRAME);
    Point bpos = pos;
    MyRadioButton *b;
    for (int i = 0; i < no; ++i) {
      bpos.setY(pos.getY() + i*30);
      b = new MyRadioButton(bpos, 100, h/no, slabel.data()+std::to_string(i));
      elts.push_back(b);
    }
    noOfElts = no;
    this->end();
 }
```

Grup de butoane radio (cont.)

```
private:
  int noOfElts;
  std::vector<MyRadioButton *> elts;
};
```

Demo

```
Point posRG(100, 150);
MyRadioGroup *rg =
      new MyRadioGroup(posRG, 150, 90, "MyChoice", 3);
rg->setController(adb);
```



Punerea in miscare a mecanismului MVC

- deocamdata nu se intampla nimic la apasarea unui buton radio
- e timpul sa facem conexiunea cu modelul si controller-ul
- modelul: memoreaza ultimul buton radio apasat
- contoller-ul: va fi un “display box” (output box)

Clasa Model

- asociat cu componentele de vizualizare

```
class Model {
 public:
  void setLastChoice(int ch);
  int getLastChoice() const;
  void  setChView(MyDisplayBox *db);
  void notify() const;
 private:
  int lastChoice;
  MyDisplayBox *chView;
};
```

Clasa Model

```
void Model::notify() const {
  chView->setText(std::string("Last choice is ") +
                            std::to_string(lastChoice));
}
```

Controller

- asociat cu modelul

```
class Controller {
 public:
  void chControl(std::string aString);
  void setModel(Model *aModel);
 private:
  Model *model;
};
```

aplica actiunea din GUI peste model

Asocierea view-controller

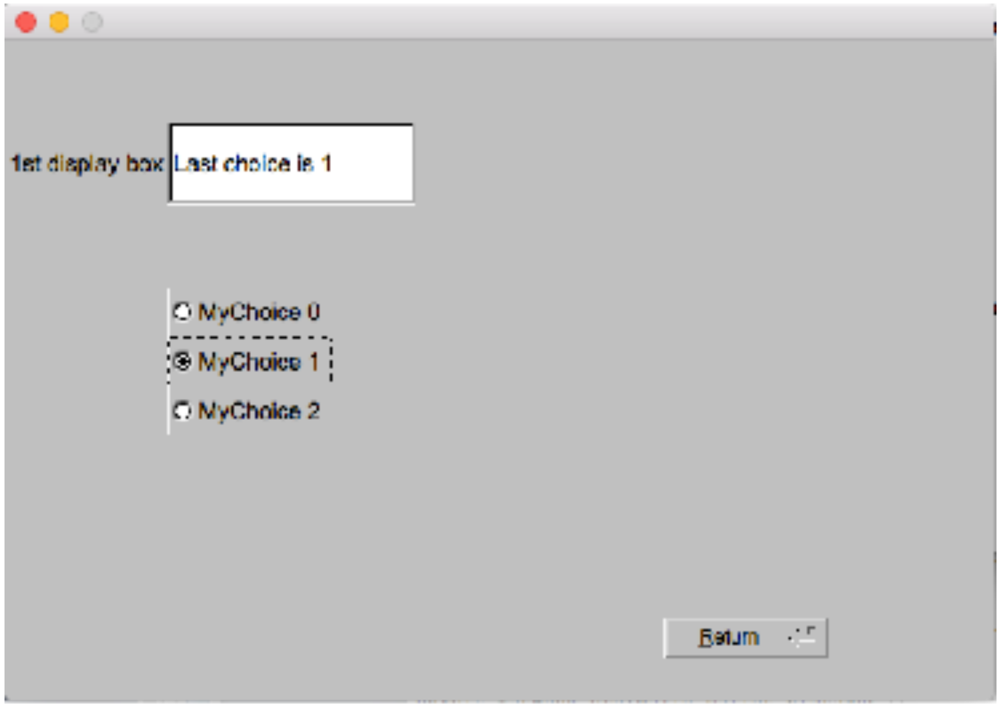
```
class MyRadioButton : Fl_Radio_Round_Button {
 public:
  …
  void setController(Controller *aCntrl);
 private:
  …
  Controller *controller;
};

class MyRadioGroup : Fl_Group {
 public:
   void setController(Controller *aCntrl);
  …
};
```

Asocierea view-controller

```
void MyRadioButton::radio_button_cb(MyRadioButton *b,
                                    void *) {
  b->controller->chControl(std::string(b->label()));
}


void MyRadioGroup::setController(Controller *aCntrl) {
    int i;
    for (i = 0; i < noOfElts; ++i)
      elts[i]->setController(aCntrl);
}
```

Demo

```
Model model(-1, "nothing");
 Controller chCntrl;
 model.setChView(adb);
 chCntrl.setModel(&model);
 rg->setController(&chCntrl);
```



My Input

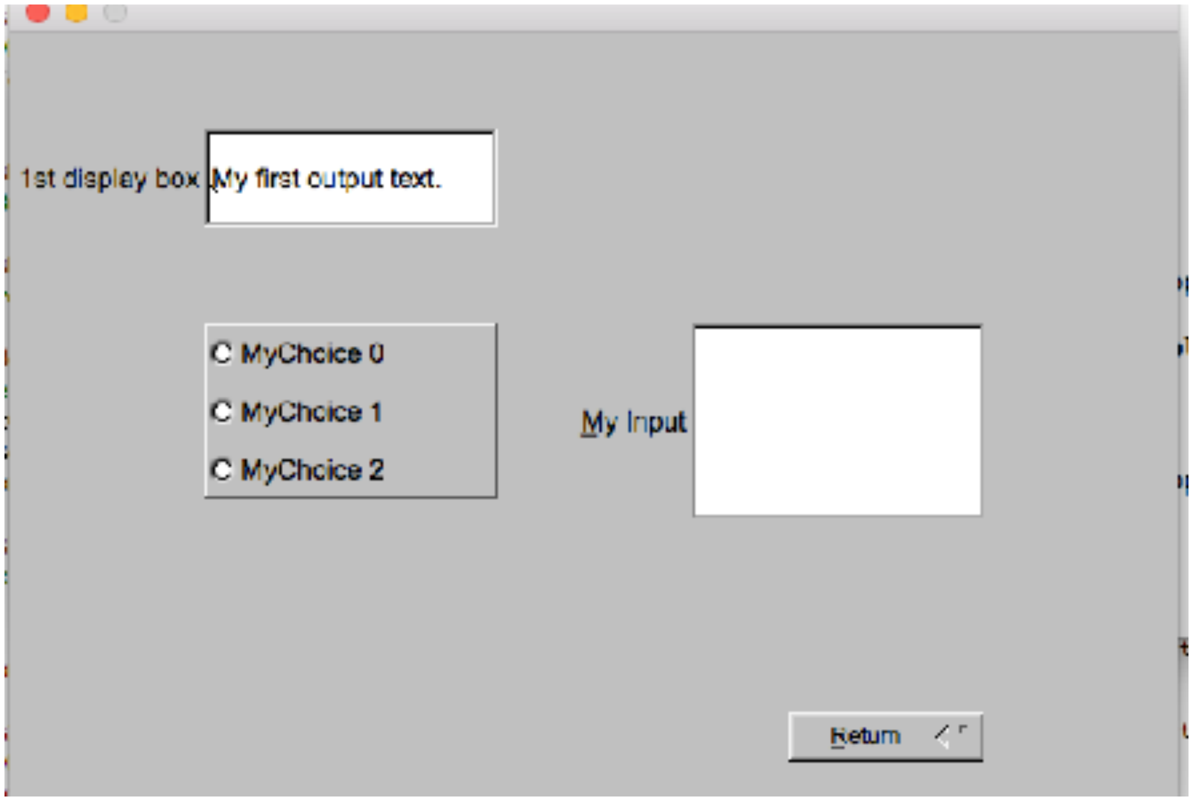
```
class MyEditBox : public Fl_Multiline_Input {
 public:
 MyEditBox(Point pos, int w, int h, char * label) :
Fl_Multiline_Input(pos.getX(), pos.getY(), w, h, label) {
    this->tooltip("Input field for short text with newlines.");
    this->wrap(1);
    //    this->when(0);
    this->when(FL_WHEN_RELEASE);
    this->callback((Fl_Callback*) input_cb);
    this->show();
  }
```

My Input

```
private:
  MyDisplayBox *controller;
  static void input_cb(MyEditBox *eb, void *) { }
};
```

Demo

```
Point posEB(350, 150);
 MyEditBox *eb = new MyEditBox(posEB, 150, 100,
                                        (char *) "&My Input");
```



Adaugarea unui model

- ca sa vedem cum poate fi utlizat textul introdus, extindem modelul cu
  - capabilitatea de a memora si ultimul text intrat in fereastra de input
- adaugam doua componente MyDisplayBox:
  - una pentru ultimul buton radio ales
  - una pentru text introdus

```
Point posSndDB(375, 50);
MyDisplayBox *snddb = new MyDisplayBox(…);
snddb->setText("My second output text.");

Point posTrdDB(200, 275);
MyDisplayBox *trddb = new MyDisplayBox(…);
trddb->setText("My third output text.");
```

Clasa Model

```
class Model {
 public:
  …
  void setLastInput(std::string inp);
  void setInpView(MyDisplayBox *db);
 private:
  …
  std::string lastInput;
  MyDisplayBox *chView, *inpView;
};
```

Clasa Model

```
void Model::notify() const {
  chView->setText(std::string("Last choice is ") +
                            std::to_string(lastChoice));
  inpView->setText(std::string("Last input is `") +
                             lastInput + std::string("`"));
}
```

Controller

```
class Controller {
 public:
  ...
  void inpControl(std::string  aString);
private:
  …
};
```

aplica actiunea din GUI peste model

Noua colaborarea model-controller-view

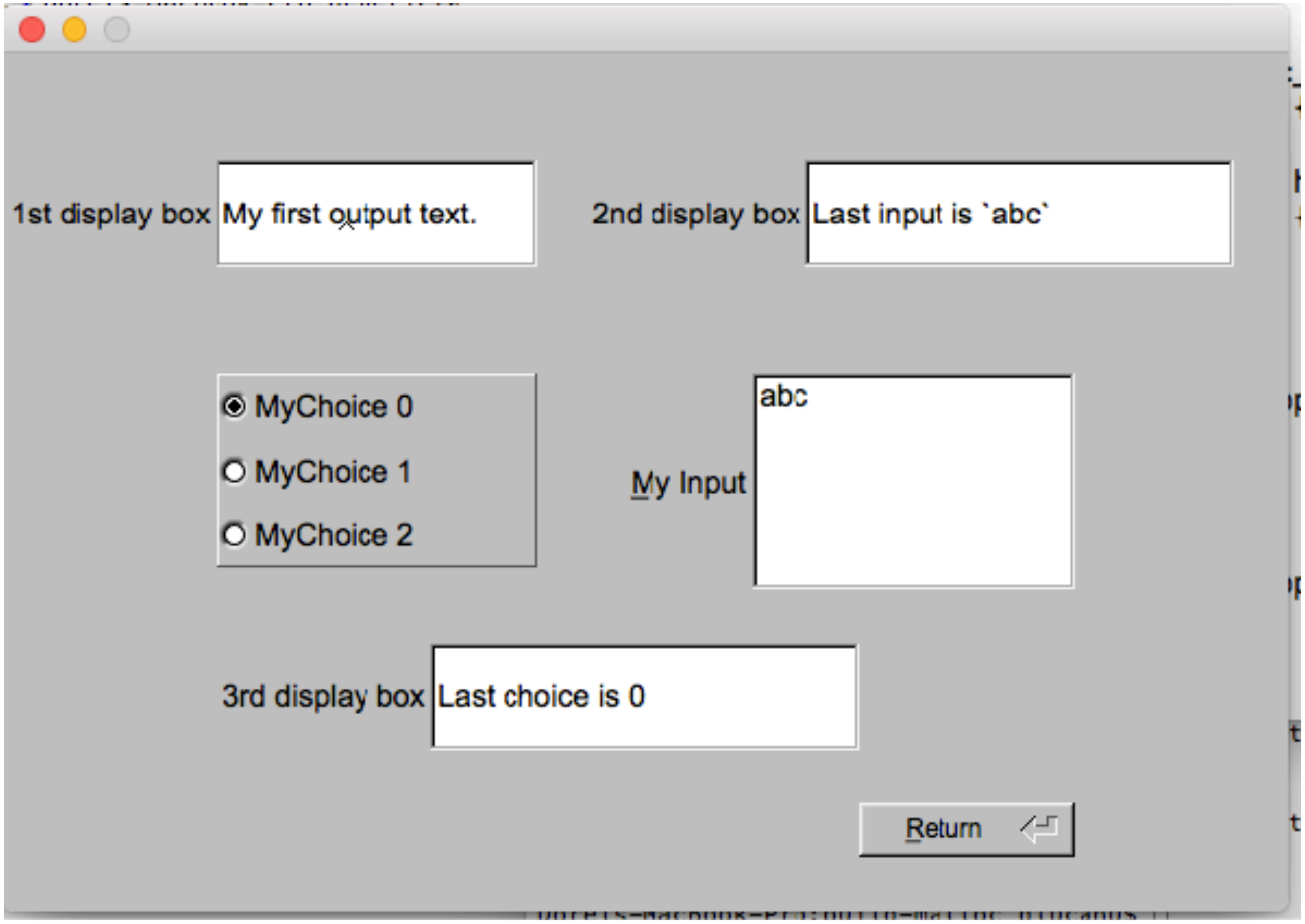
```
eb->setController(&chCntrl);
```

relatia view (GUI) - controller

```
model.setInpView(snddb);
model.setChView(trddb);
```

relatia model - view (GUI)

Demo


1st display box
My first output text.
2nd display box
Last input is `abc`
MyChoice 0
MyChoice 1
MyChoice 2
My Input
abc
3rd display box
Last choice is 0
Return

POO

Sumar al conceptelor si principiilor OO

Intrebari

- Ce este POO?
- Ce este un obiect?
- Ce este o clasa?
- Cum se identifica si se proiecteaza o clasa?
- Ce este incapsularea (sau ascunderea informatiei)?
- Ce inseamna supraincarcarea metodelor?
- Ce este mostenirea?
- Ce relatie exista intre mostenire si derivare in C++?
- Cate tipuri de derivari exista si care sunt diferentele dintre ele?
- Ce inseamna suprascrierea metodelor?
- Ce este polifomofism dinamic si cum se realizeaza in C++?

Intrebari

- Ce este o relatie de asociere?
- Ce este o relatie de agregare?
- Care este diferenta intre relatia de asociere si cea de agregare?
- Ce este abstractizare si generalizare?
- Ce este o clasa abstracta si cum se implementeaza in C++?
- Ce este o interfata si cum se implementeaza in C++?
- Care este diferenta dintre clasa abstracta si interfata?

Intrebari

- Ce este o diagrama “use case”?
- Ce este o diagrama de clasa?
- Ce este o diagrama de secventiere (colaborare)?
- Ce este MVC?
- Ce este Principiul “Inchis-Deschis” (Open-Closed Principle)?
- Ce este Principiul Substituirii al Liskov?
- Ce este principiul Inversarii Dependentelor?

Intrebari

- Ce este un sablon de proiectare?
- Cum se clasifica sabloanele de proiectare in GoF?
- Care este diferenta dintre Abstract Factory si Builder?
- Cand se utilizeaza Abstract Factory?
- Cand se utilizeaza Composite?
- Cand se utilizeaza Adapter?
- Cand se utilizeaza Visitor?
- Cand se utilizeaza Observer?
- Care este diferenta dintre Visitor si Iterator?
- Cand se pot utiliza impreuna Visitor si Composite?
- Cand se pot utiliza impreuna Composite si Abstract Factory?