MVC Design pattern

陳偉凱
台北科技大學資工系

How do you cook meatball spaghetti? (two choices)

2

Two Choices

(A) Cook each ingredient and then mix

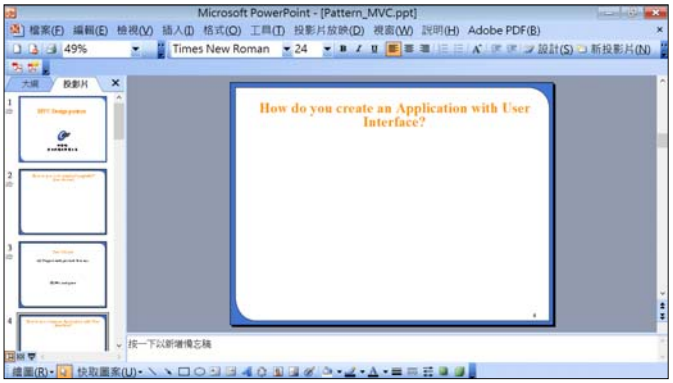
(B) Mix all ingredients and then cook

Advantages of (A) over (B)?

- Each ingredient needs a different cooking method/time
- Each ingredient can be stored (maintained) differently
- Changing spaghetti, meat sauce, or meat balls is easy
- Each ingredient can be used in other dishes

3

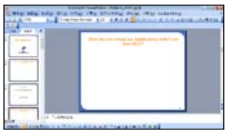
How do you develop an application with UI? Two Choices



4

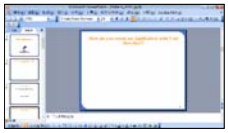
Two Choices

(A) Develop modules (classes) and then integrate

Module 1 (spaghetti) + Module 2 (meat sauce) + Module 3 (meat ball) → 

(meatball spaghetti)

(B) Develop a BIG app that includes everything

App (spaghetti + meat sauce + meat ball) → 

Advantages of (A) over (B)?

5

How do you develop an application with UI?

What is a high-level module for PowerPoint?

(A) Menu

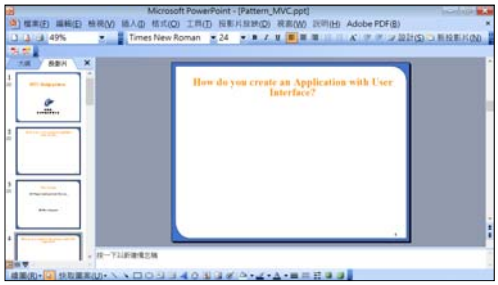
(B) Toolbar

(C) Statusbar

(D) Thumbnails pane

(E) Slide pane

(F) All of the above



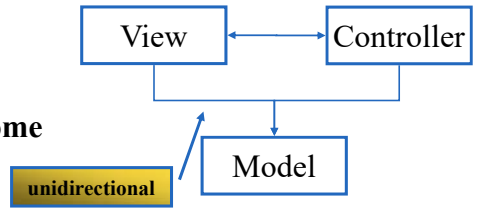
(G) None of the above (explained in the next slide)

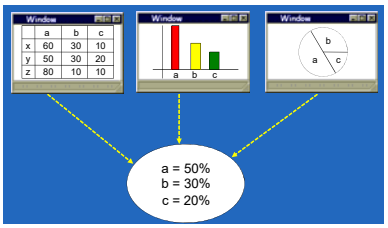
6

MVC: Model View Controller

Splits user interface interaction into three distinct roles.

- **The *model* represents some information about the domain.**
- **The *view* represents the display of the model in the UI.**
- **The *controller* takes user input, manipulates the model, and causes the view to update appropriately.**



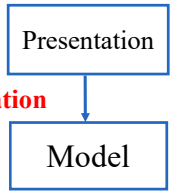


7

MVC: How it Work

Separating the presentation from the model

- **Fundamentally presentation and model are about different concerns.**
- **Depending on context, users want to see the same basic model information in different ways.**
- **Non-visual objects are usually easier to test than visual ones.**
- **The presentation depends on the model**
  - **Model does not know the existence of presentation**
- **Observer pattern [GOF]**

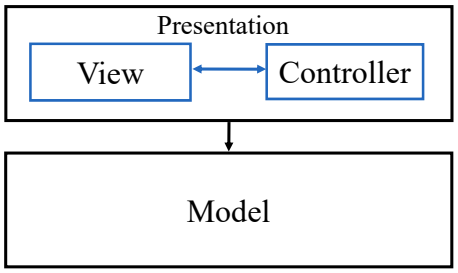


8

MVC: How it Work

Separating the controller from the view

- **View does not catch user input**
- **Controller = Strategies pattern [GOF]**



9

MVC: When to Use It

The value of MVC lies in its two separations

- **Separating the presentation from the model (must)**
- **Separating the controller from the view (less important; next page)**

The separations can be accomplished by separating into

- **Method, Class, Module, Subsystem**

V/C → M

M←V←→C

10

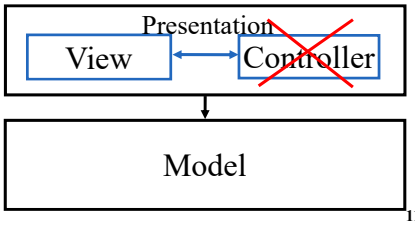
MVC: When to Use It

Modern rich GUI

- **Microsoft Windows and other graphical operating systems provides widgets from which user interfaces can be constructed.**

A widget (control) can both display information and take user inputs: a widget already includes both V and C.

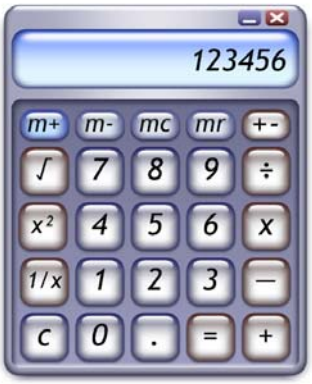
Modern GUI: Only MV, no C



11

MVC Exercise

What is the View and Model of the following calculator?



Answer:

View

A window with 25 buttons and a TextBox

Model

1.Stores data (e.g., the current result and memory)
2.Support operations on the data (e.g., +, -, *, /)

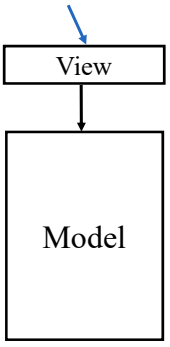
12

MVC: Thin View

Thin View

NOT in original MVC

Also called presentation

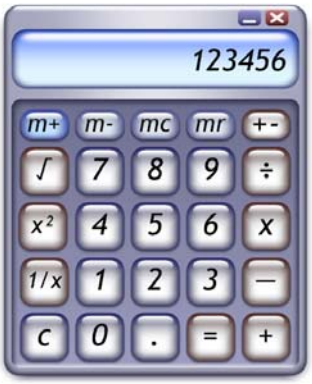


- Make your UI as thin as possible
  - Avoid "Fat View Smell"
- Allows better testing and debugging of the model
- The Model class should have public interfaces supporting (handling) every event raised in the View
- Most logics should reside in the model
  - Except for logics related only to the view
    - For different presentations of the same model.
  - Put logics related only to the view to Presentation Model

13

MVC Exercise

What is a typical Model class declaration for the following Calculator that supports thin View?




```
class CalculatorModel
{
    ???
    ???
    ???
};
```

14

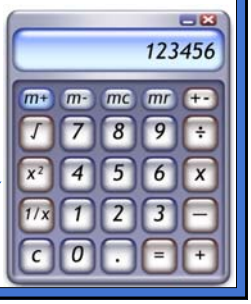
MVC Exercise

```
class CalculatorModel
{
public:
        CalculatorModel();
        // digits
        void ProcessDigit(int digit);
        void ProcessDot();
        // memory
        void ProcessMP();
        void ProcessMM();
        void ProcessMC();
        void ProcessMR();
        // Others
        void ProcessC();
        void ProcessPlus();
        void ProcessMinus();
        void ProcessMultiply();
        void ProcessDivide();
        void ProcessPlusMinus();
        void ProcessEqual();
        void ProcessSqrt();
        void ProcessSqr();
        void ProcessReciprocal();
        // Output
        string GetDisplay();
private:
        double memory;
        double lhs;
        double display;
};
```

void

string

All events



MVC: Unidirectional dependency

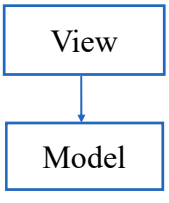
View uses (calls) Model

- But, Model does not use (call) View

```
class View
{
    Model model; // association
    void Foo()
    {
        …
        model.Foo();
        …
        xxx = model.GetDisplay();
    }
}
```

View calls Model

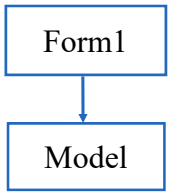
Return value is OK



16

Example code

```
namespace XXX
{
   class Program
   {
      static void Main(string[] args)
      {
          model m = new Model();
          Form1  form = new Form1(m);
          Application.Run(form);
      }
   }
}
```

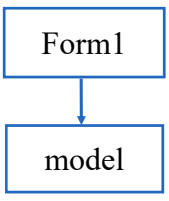


17

Example code

```
namespace XXX
{
   class Form1 : Form
   {
      private model m;
      public Form1(Model m)
      {
         this.m = m;
         …
      }
      private void B1_Click(object …)
      {
         m.press(1);
         xxx.Text = m.GetResult();
      }
   }
}
```

Return value

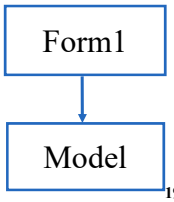


18

Example code

```
namespace XXX
{
   class Model
   {
      …
      public void foo(…)
      {
         …
         // Good: perform some model computations
         str = …
         // Bad: Model should not know View
         form.xxx.Text = str;
      }
   }
}
```

Violates one-way dependency



19

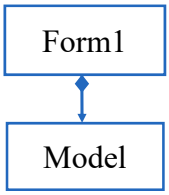
How about creating the Model object from the View object?

```
class Program
{
   static void Main(string[] args)
   {
      Form1  form = new Form1();
      Application.Run(form);
   }
}
```

```
class Form1 : Form
{
   private model m = new Model();
   public Form1()
   {
      …
   }
   private void B1_Click(object …)
   {
      m.press(1);
      xxx.Text = m.GetResult().ToString();
   }
}
```

Is this good?

Not recommended!
Does not work when two different views share the same model, though one-way dependency is not violated.



20

Guideline: Don't pass View objects to Model

```
class View
{
    Model model; // association
    …
    void Foo()
    {
        …
        model.foo(ok_button);
        …
    }
}
```

```
class model
{
    …
    public void foo(Button ok_button)
    {
        …
    }
}
```

A View object should not be passed to model

21

Guideline: Don't pass View objects to Model

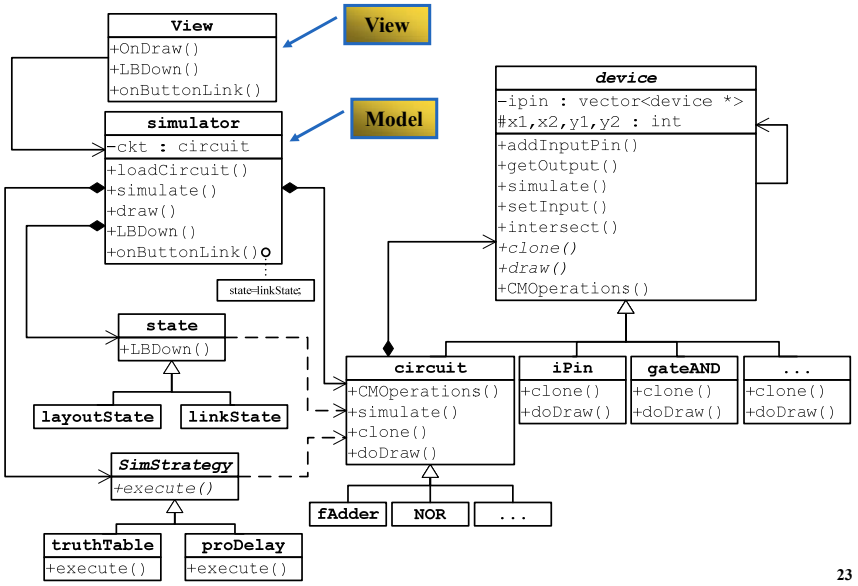
```
class View
{
    Model model; // association
    …
    void Foo()
    {
        …
        model.foo(textBox1.Text);
        …
    }
}
```

```
class model
{
    …
    public void foo(string text)
    {
        …
    }
}
```

Is textBox1.Text a View object?

22

MVC example: Logic circuit simulator



23

MVC

Question: can I implement the code first, and then separate (refactor) the code into MVC according to their roles?

Answer:

24

MVC for other platforms

What about Web applications?

What about Android applications?

What about iOS applications?

25

References

GUI Architectures, http://martinfowler.com/eaaDev/uiArchs.html

The Model-View-Presenter-ViewModel Design Pattern for WPF, http://msdn.microsoft.com/en-us/magazine/hh580734.aspx

26