Software Architecture
软件体系结构

Lecture 1. MVC Design Pattern
MVC Architecture

Professor:

Yushan (Michael) Sun
Fall 2020

Contents of this lecture

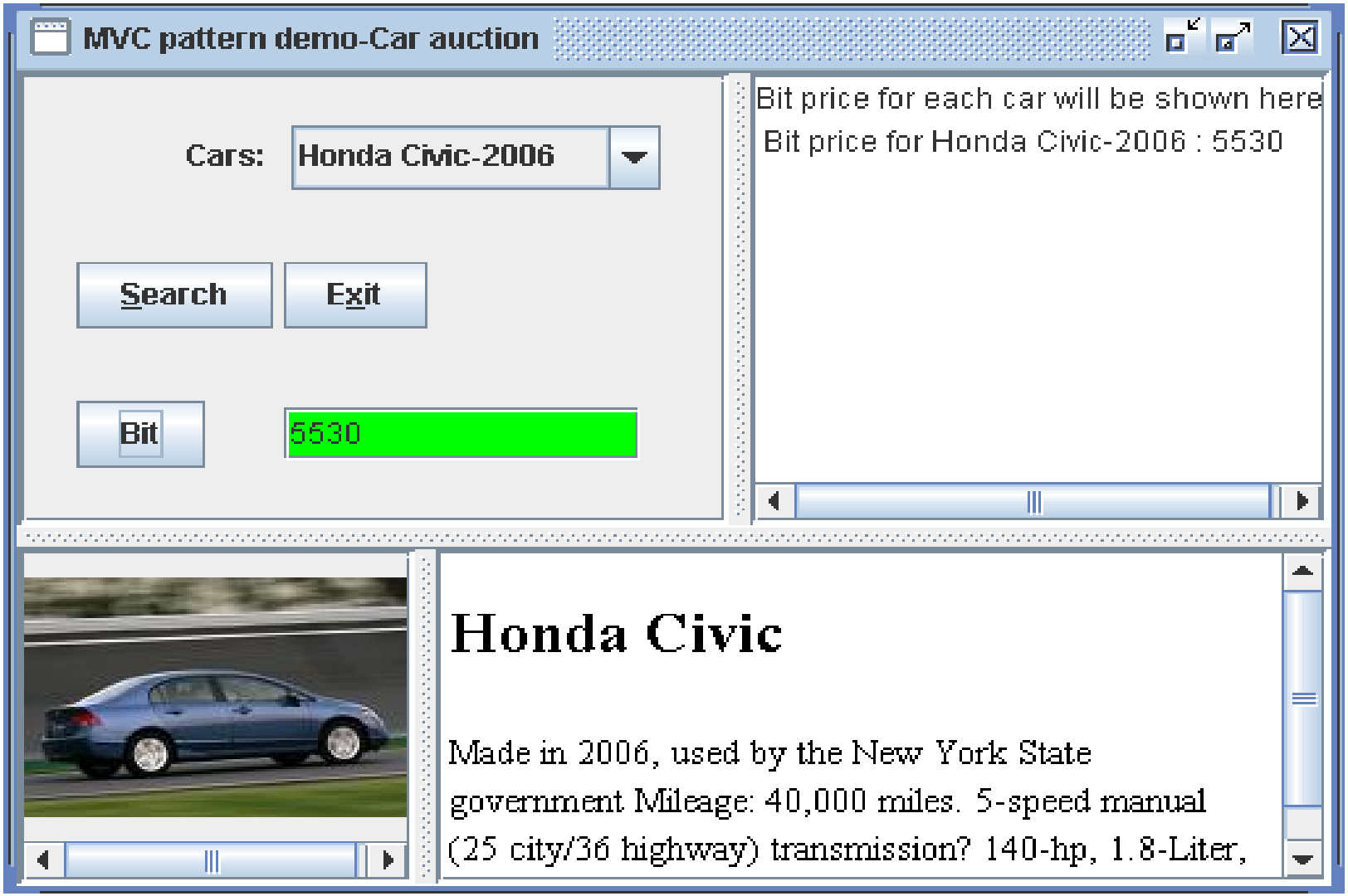
1. Introductory Example to the MVC Design Pattern or Architecture
2. Theory of the MVC design pattern
3. Design examples using the MVC design pattern

Introductory Example to the MVC Architecture

网上二手车拍卖系统的设计

- Example. software requirement: need a used car auction software that runs on the Internet。
- This software has a graphical user interface, which can display the car pictures, the descriptions of cars, and the current bit price
  a) 用户首先在车的列表中选择一款车。User firstly chooses a car from a list, and then clicks on search button
  b) 该车的图片，描述，与当前拍价将在屏幕上显示. Then the picture, descriptions and the current bit price of the chosen car will be shown on screen
  c) 用户输入拍价 User inputs his/her own bit price
  d) 新的拍价在另外一个视窗中显示。The new bit price will be displayed on another window

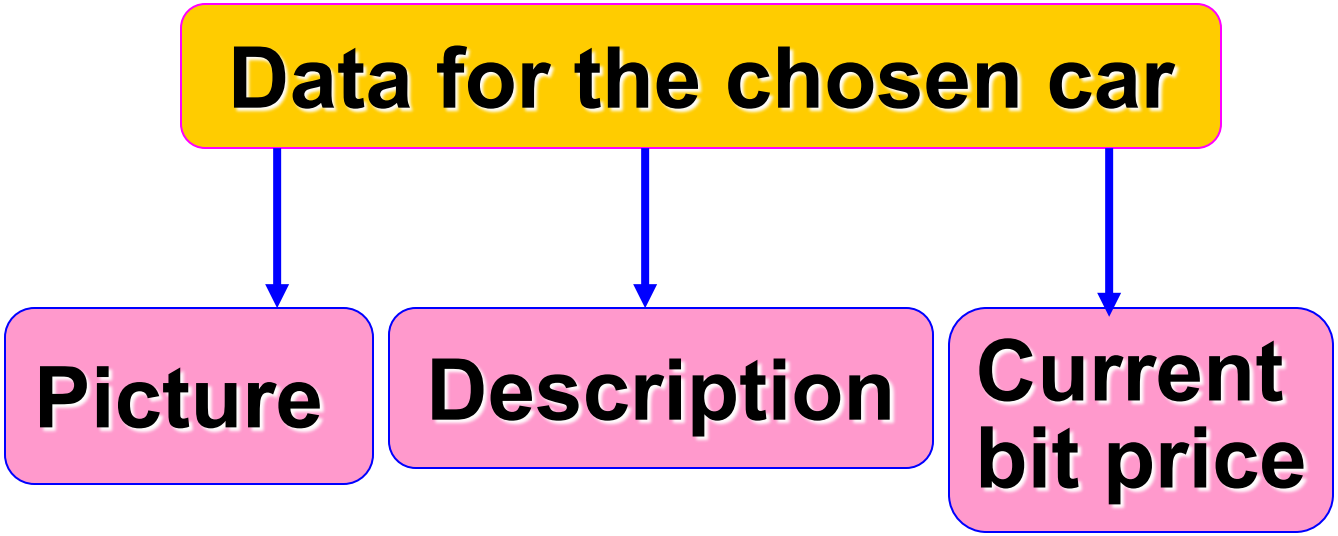
网上二手车拍卖系统的设计



The prototype of the graphical user interface

网上二手车拍卖系统的设计

关于数据的显示：有三种显示方法



Different data displays

网上二手车拍卖系统的设计：方案1

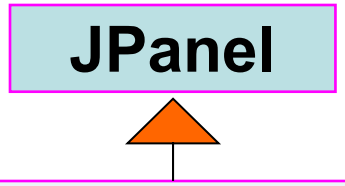
- **Design 1: 仅仅使用一个Java类实现全部功能**
  use only one java class to implement all the functions

  1) 产生用户图形界面. Produce user interface

  2) 包含所有的业务逻辑. All the business logics are in this java class

网上二手车拍卖系统的设计：方案1

设计方案1：只用一个Java类

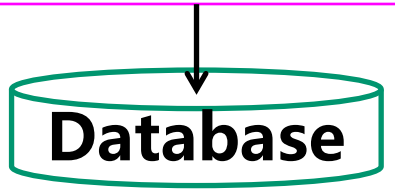


包含GUI,业务逻辑数据库访问

CarAuctionGUI

+getSelectedCar(): String //get user input
+getBitPrice(): String //get user input bit price
+extractCarList(): String[] //display car list
+setUpCarList(JComboBox cmbCarList,String[] carList) //add car list onto cmbCarList
+constructCarFileUrl(String carChosen): URL //produce an UML based on car name
+updateCarDescription(JEditorPane editorPane, URL url) //update car description
+showBitPrice(JTextArea bitShownText, String price) display bit price
+produceCarImaIcon(String carChosen): ImageIcon //produce image icon for car
+updateCarPicture(JLabel imgLabel,ImageIcon imgIcon) //update car picture
#createImageIcon(String path): ImageIcon

Design 1: use only one java GUI file to implement all the functions



网上二手车拍卖系统的设计：方案1

缺点 Drawbacks:

- 可扩展性不好 When new function is added, need to recompile the whole class
- 可维护性不好 When you Modify something, need recompile whole class

➢解决方案(Solution)：redesign

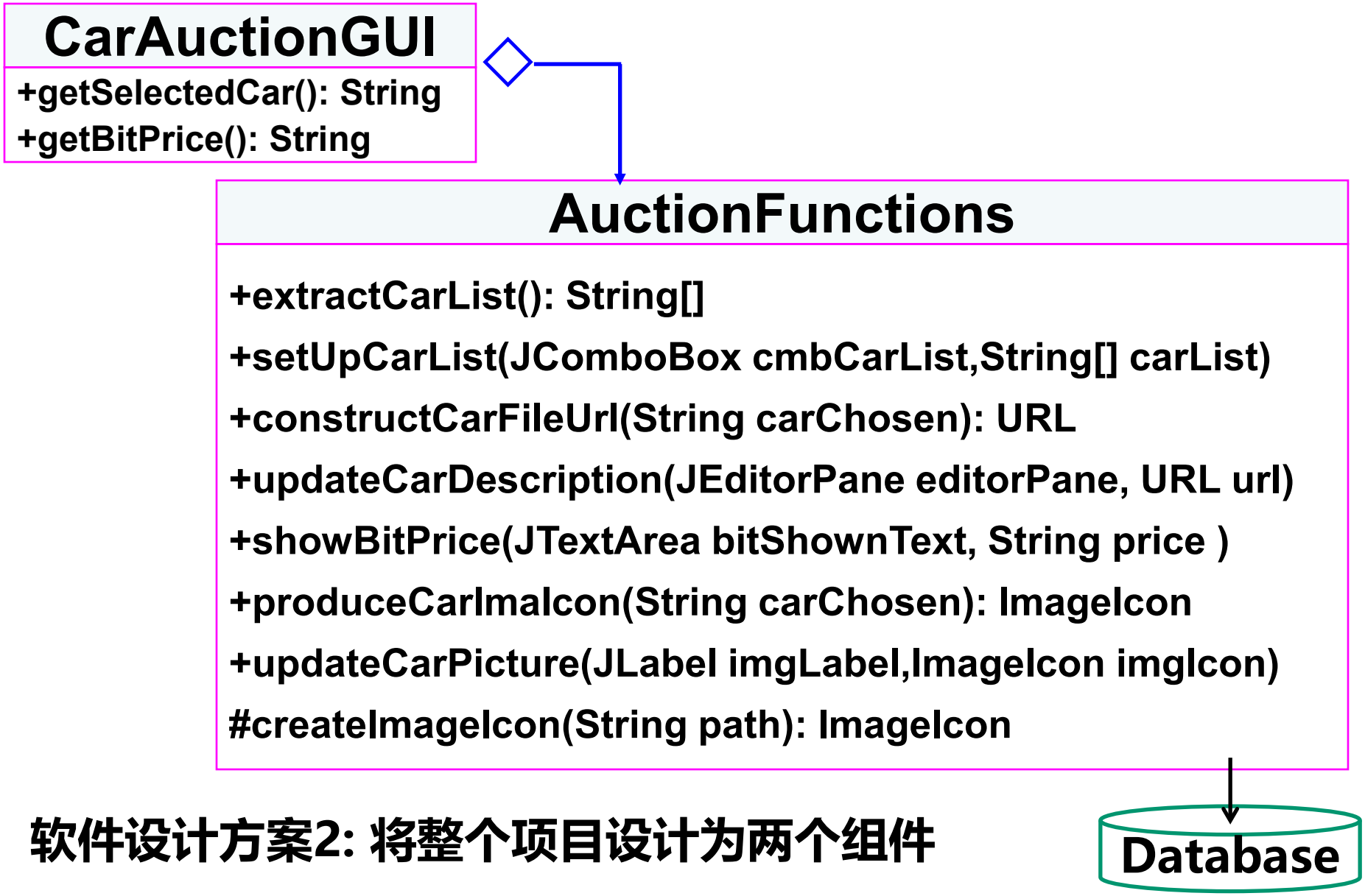
网上二手车拍卖系统的设计：方案2

软件设计方案2: 将整个项目设计为两个组件

- **用户图形界面CarAuctionGUI**:
  a) 用户输入功能
  b) 查询结果

- **类AuctionFunctions：**
  1. 从一个文件夹中提取车的信息描述文件
  2. 从一个文件夹中提取车的图片文件
  3. 相关的图片显示功能
  4. 处理竞拍价格 与 竞拍价格显示功能
  5. 所有的业务逻辑方法
  6. 可能的数据库访问方法

网上二手车拍卖系统的设计：方案2



软件设计方案2: 将整个项目设计为两个组件

网上二手车拍卖系统的设计：方案2

设计方案2的优点: 实现了责任分离

1. 在用户界面程序中，除了搭建所有的图形组件外， 仅仅提供了用户输入(和简单的输出功能)的获取功能：
   - getSelectedCar(): String
   - getBitPrice(): String
2. 将所有其它的功能都放到单独的一个类AuctionFunctions中

网上二手车拍卖系统的设计：方案2

设计方案2的缺点

在一个类AuctionFunctions中，集中了

- 业务逻辑
- 部分辅助显示功能
- 其它的所有辅助功能

造成了接口污染

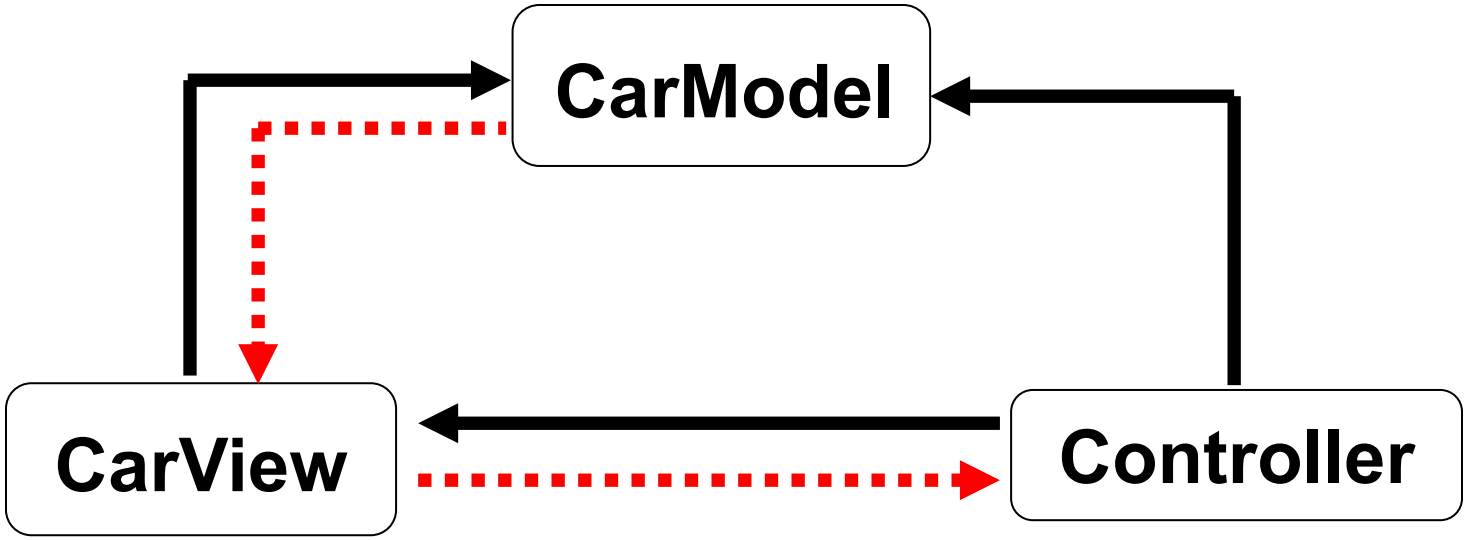
MVC Design Patterns

新设计方案考虑将来(追加)可能扩展的功能：

1. **要求在输入方面有可扩展性，例如**
   - **键盘输入，**
   - **图形界面上的按钮输入**
   - **手机、平板的触摸屏输入**
2. **对于汽车信息，允许有不同的显示**
   - **图片显示**
   - **文字显示**
   - **性能比较、分析图形**
   - **销售情况（柱形图，饼形图）**
3. **希望将所有的业务逻辑都封装到一个类中**

MVC Design Patterns

- **解决方法**：使用MVC (Model-View-Controller)设计模式进行设计；将二手车拍卖系统设计为3个包。



Back

Theory of the MVC design pattern

MVC Design Patterns

- **MVC历史**
- **MVC was first described in 1979 by Trygve Reenskaug, then working on Smalltalk at Xerox PARC (施乐帕克研究中心)**

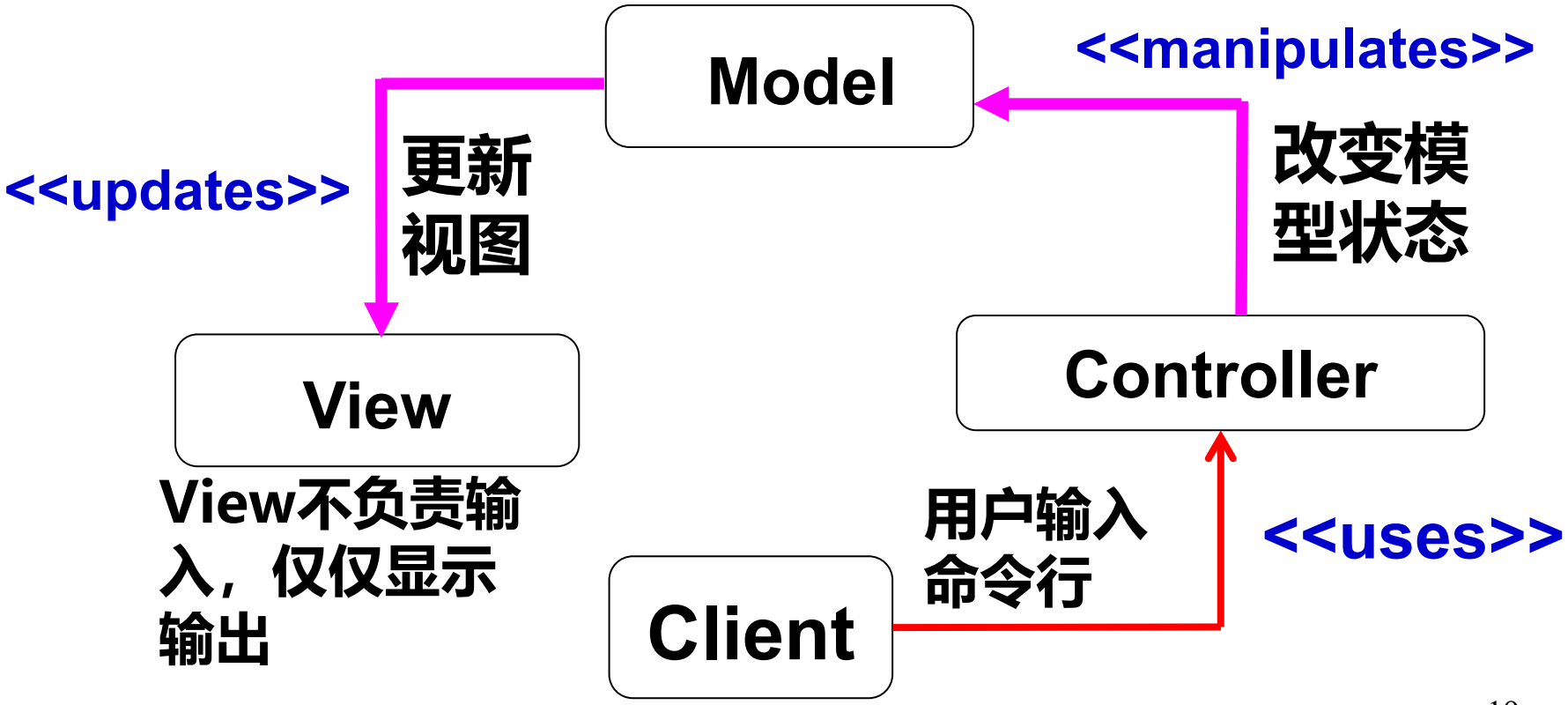
- **从二十世纪90年代开始，MVC被认为是架构形模式**
- **Since the late 1990s, MVC is commonly classified as an architectural pattern**

MVC Design Patterns

- **定义：In software engineering, Model–View–Controller (MVC) is an architectural pattern that splits interactions between users and applications into three roles:**
  - **the Model (business logic),**
  - **the View (user interface), and**
  - **the Controller (user input).**

MVC Design Patterns

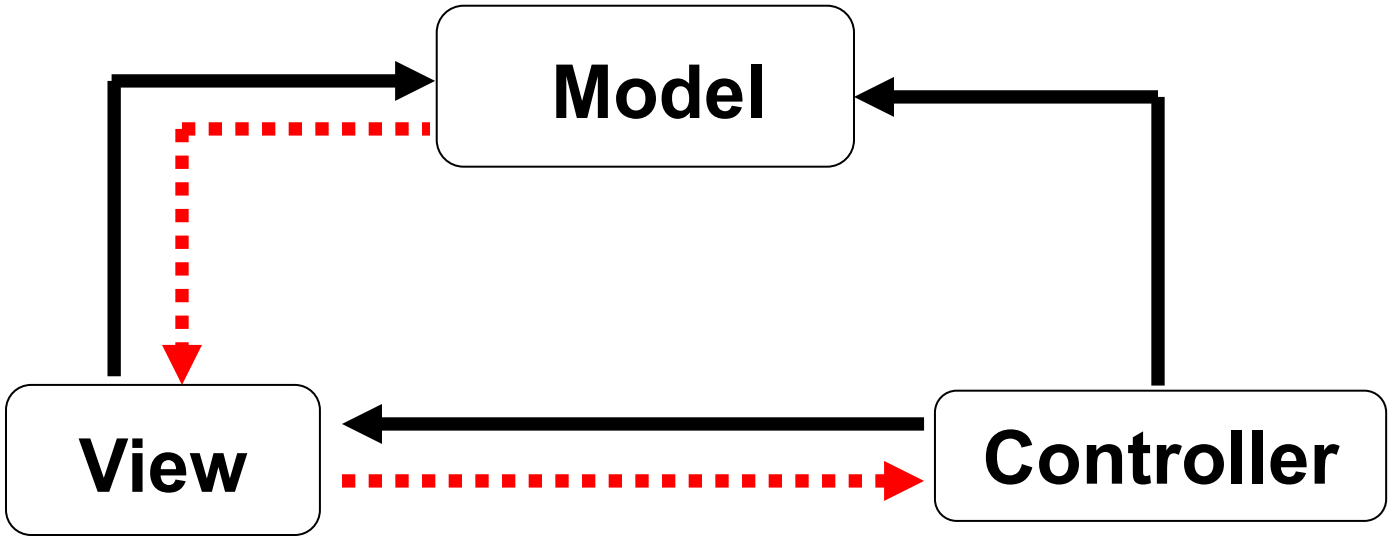
- MVC设计模式将系统分成3个组件： Model，View， Controller



MVC设计模式的较早的应用的交互情况

MVC Design Patterns

- **Components:** Model, View , Controller
- **Connectors**: Explicit invocation， implicit invocation or other mechanisms (e.g. HTTP protocol)



MVC模式架构图

MVC Design Patterns

- **Model的责任**
- Model is responsible for
  - Providing the data from the database and saving the data into the data store **(负责从数据库中提取数据；将数据存储到data store).**
  - All the business logics are implemented in the Model **(负责业务逻辑实现).**
  - Data entered by the user through View are checked in the model before saving into the database **(负责数据验证，然后将数据存入数据库).**

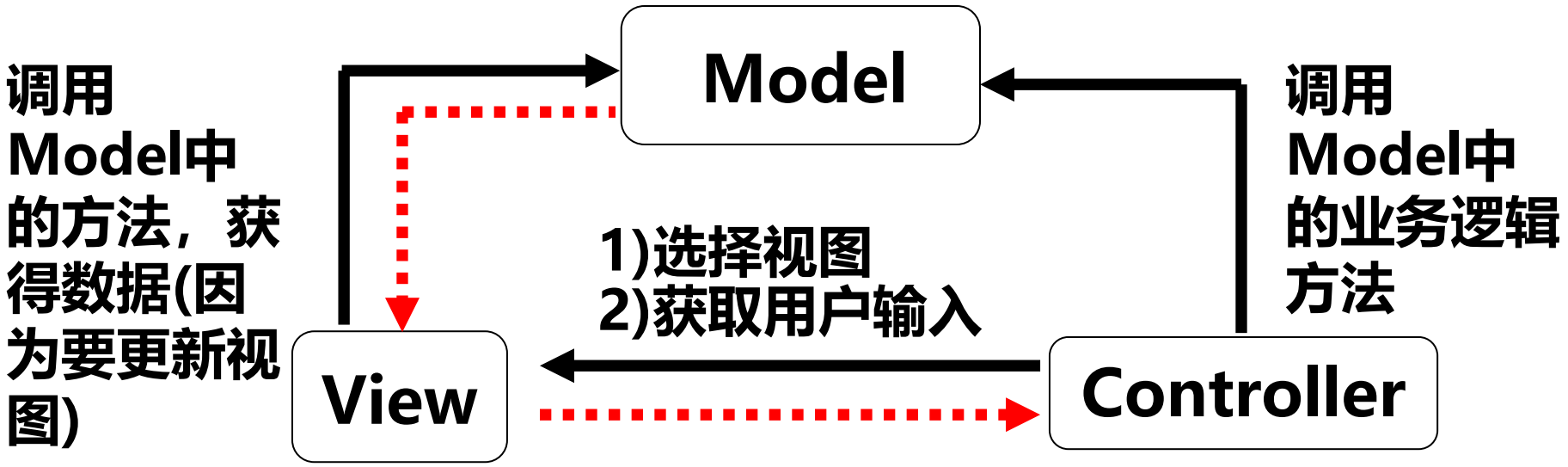
MVC Design Patterns

- **视图的责任**
- View is responsible for:
  - **捕捉用户输入**. Taking the input from the user
  - Dispatching the request to the controller, and then **(向controller发送处理请求)**
  - **显示输出(遵照控制器指示)**. Receiving response from the controller and displaying the result to the user
- **一个model可能有多个View.** Multiple views can exist for a single model for different purposes.

MVC Design Patterns

- **控制器的责任**
- Controller is responsible for:
  - Receiving the request from client **(接收来自客户的请求)**
  - Executing the appropriate business logic from the Model **(调用model业务逻辑方法)**
  - Producing the output to the user using the View component **(调用View显示执行结果)**

MVC Design Patterns



a) View接收用户请求；通知Controller
b) Controller 调用View；选择视图，获取用户输入
c) 根据用户输入，Controller调用Model的业务逻辑方法
d) Model执行业务逻辑后通知View: 我的数据有所变化
e) View调用Model中的方法，获得数据(用于更新视图)

MVC模式架构图-交互情况

MVC Design Patterns

- MVC架构的两个主要的分离
- Two principal separations:
  - 将表示从模型中分离出来 Separating the presentation from the model
  - 将控制器从View中分离出来Separating the controller from the view

将整个程序设计为一个单独的类的情况。

用户图形界面和控制器在一起的情况。

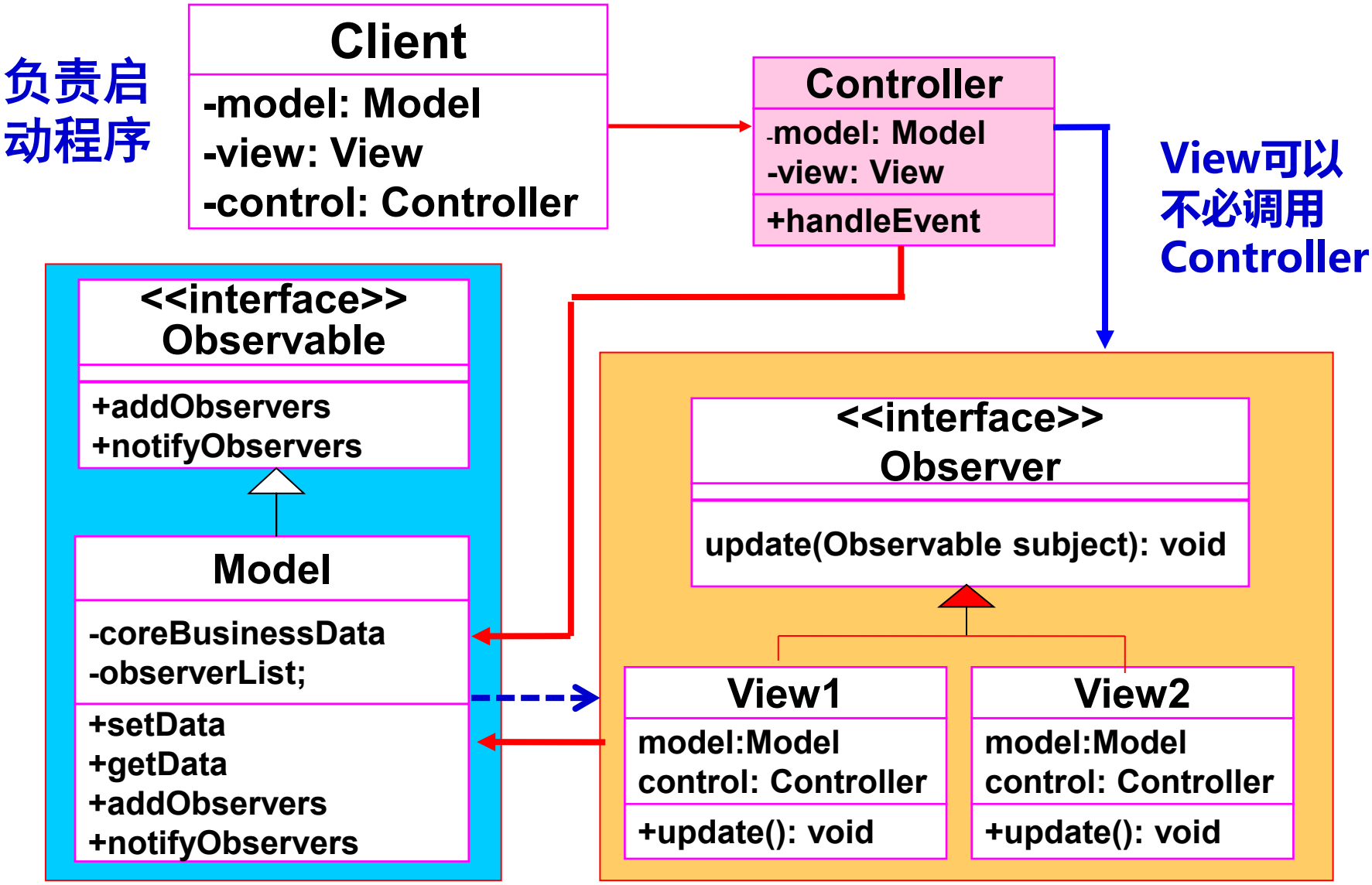
MVC Design Patterns

数据更新：改变-传播机制(change-propagation)

- 如果用户通过一个view的controller改变了model，所有的view必须反映出该改变。
- 当数据发生变化的时候，model通知所有的view，告诉他们数据已经改变了；
- Views可以**遍历(访问)**model中的数据，以便发现到底是什么改变了。然后更新显示数据。

【注】change-propagation机制保证了用户界面和模型的一致性。这个改变-传播机制可以由**观察者模式**实现。

MVC Design Patterns

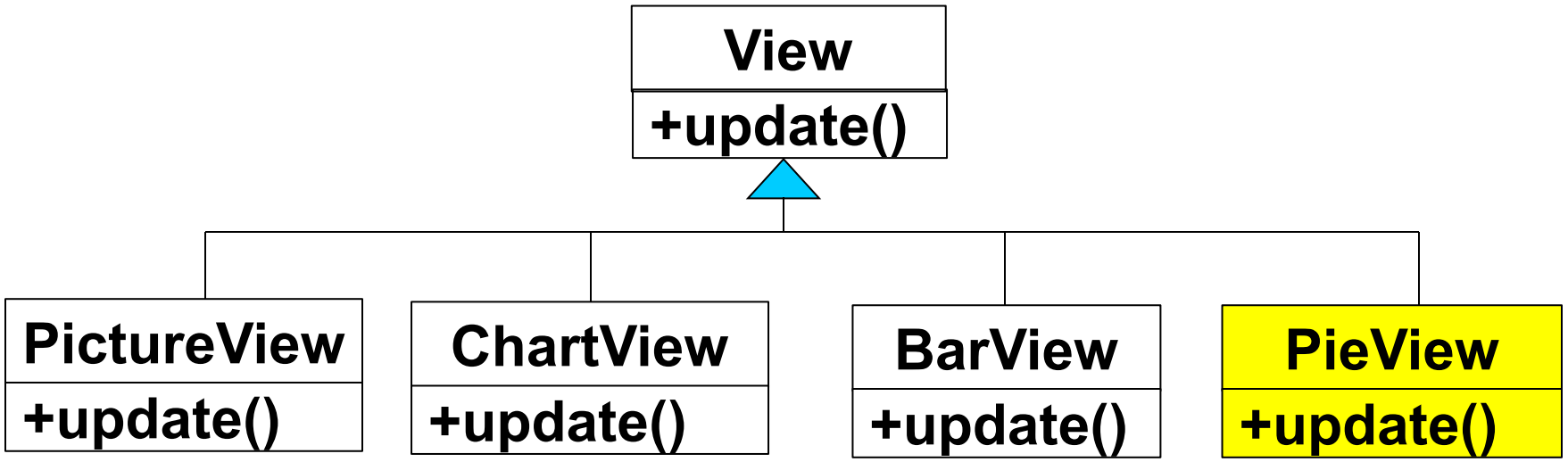


当代的一种MVC模式架构类图-使用了观察者模式

MVC的优点

1）容易增加或者改变视图

业务逻辑被封装在Model中，所以在新增加一个视图的时候，不必要改动模型，而是因为业务逻辑都是一样的，所以只需要新增加一个视图类即可。



MVC的优点

2) **容易独立地更新每个独立的软件模块**
由于一个应用被分离为三个软件模块，因此，我们可以独立地改变其中的一个模块，而不影响其它两个模块。例如，一个应用的业务流程或者业务规则的改变只需改动MVC的Model模块。

3) **代码易开发易维护** Make source code more flexible and maintainable

4) **业务逻辑更易测试** Make it very easy to test all the domain logic. Because Non-visual objects are usually easier to test than visual ones

MVC Frameworks

- **MVC架构在以下的架构中均有所体现**
- **J2EE:**
  - **Struts**
  - **Spring MVC**
- **PHP**
  - **CakePHP**
  - **Struts4php**
- **C#.NET**
  - **Girders**

Back

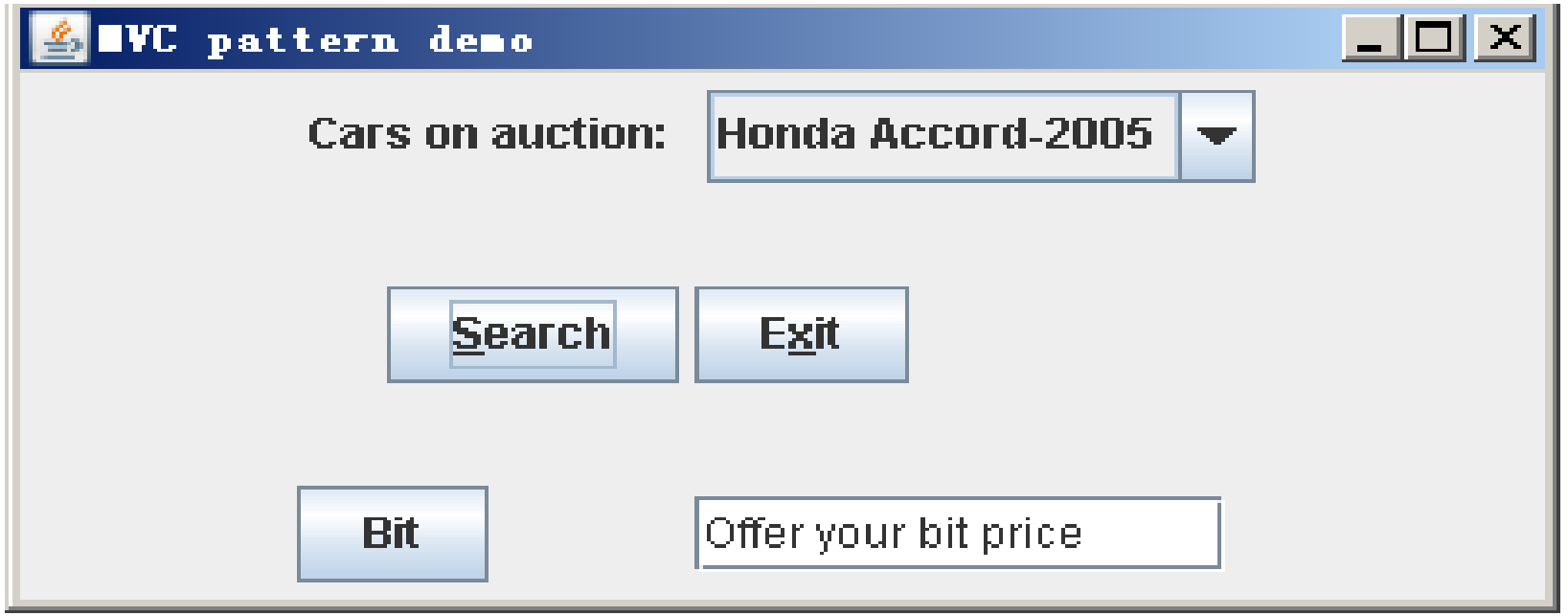
Design Examples Using the MVC Design Pattern
MVC Architecture

网上二手车拍卖系统的设计：方案3

方案3：采用MVC模式-使用观察者机制的情况，用户输入界面和两个显示视图分别独立显示

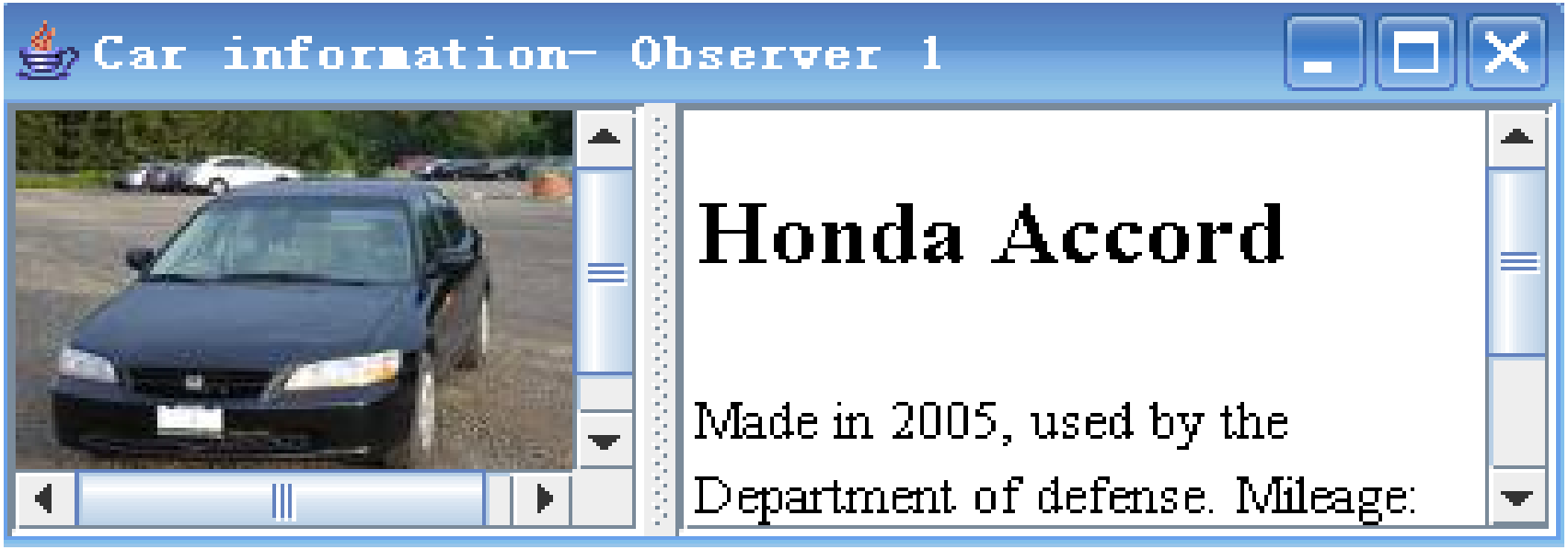
- 将事务逻辑部分和用户界面部分分开，采用MVC模式。设计如下：
  - CarAuctionGUI，提供用户输入界面，代表用户；
  - CarModel，Model部分，封装事务逻辑功能
  - CarBitView，view部分，显示用户给出的拍卖价格
  - CarGUIView，View部分，显示汽车图片
  - Controller，controller部分，处理用户输入，调用model和view

网上二手车拍卖系统的设计：方案3

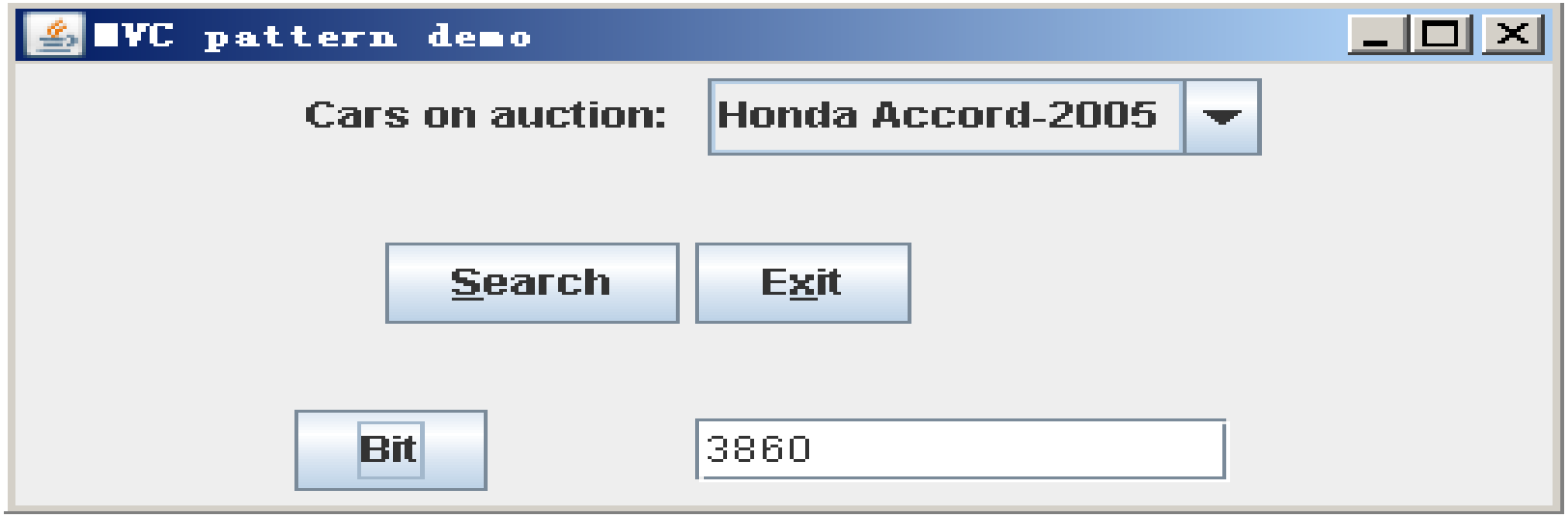


用户输入界面原型（Prototype）-点击search按钮的时候

网上二手车拍卖系统的设计：方案3

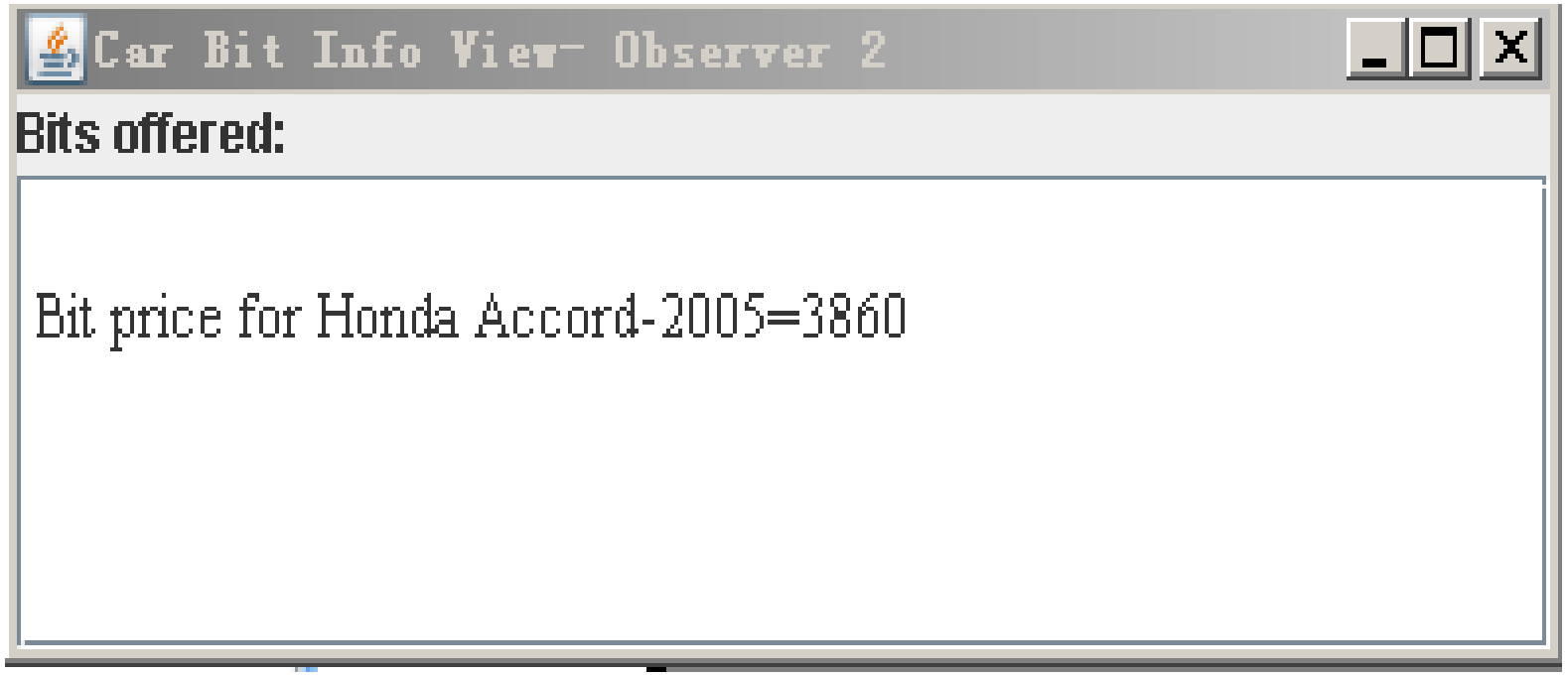


二手车信息显示界面原型（Prototype）

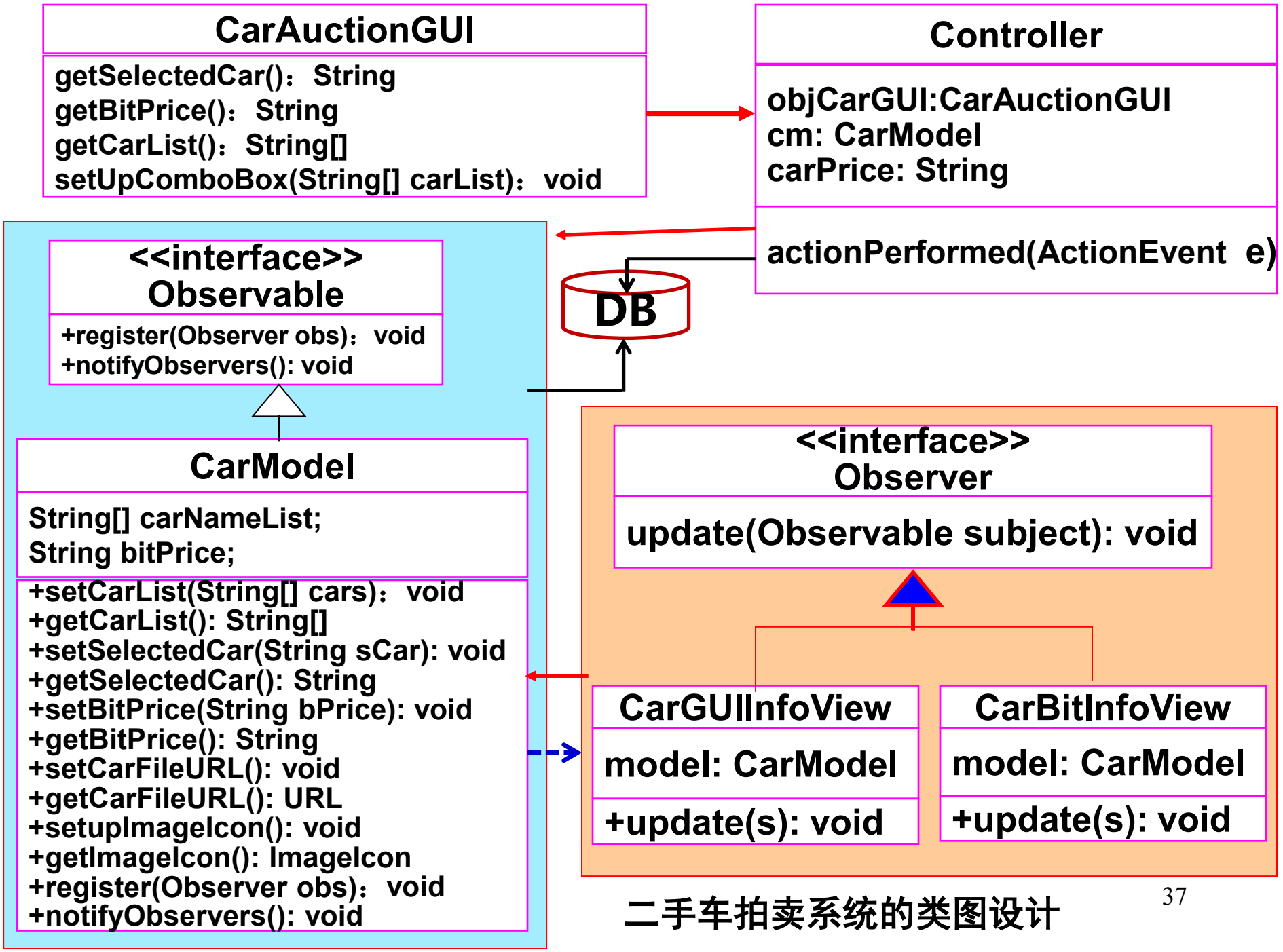


用户输入界面原型（Prototype）-点击bit按钮的时候

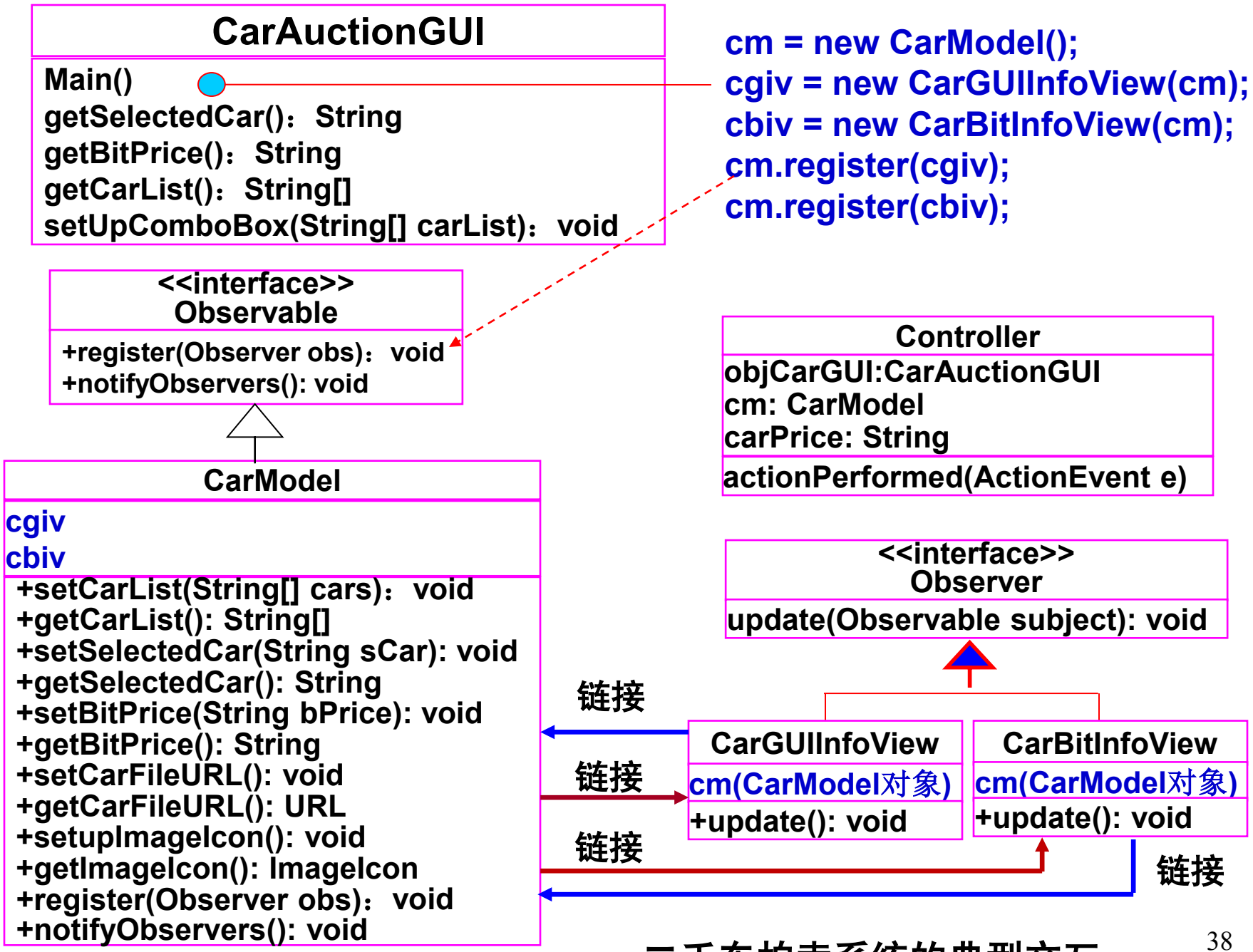
网上二手车拍卖系统的设计：方案3



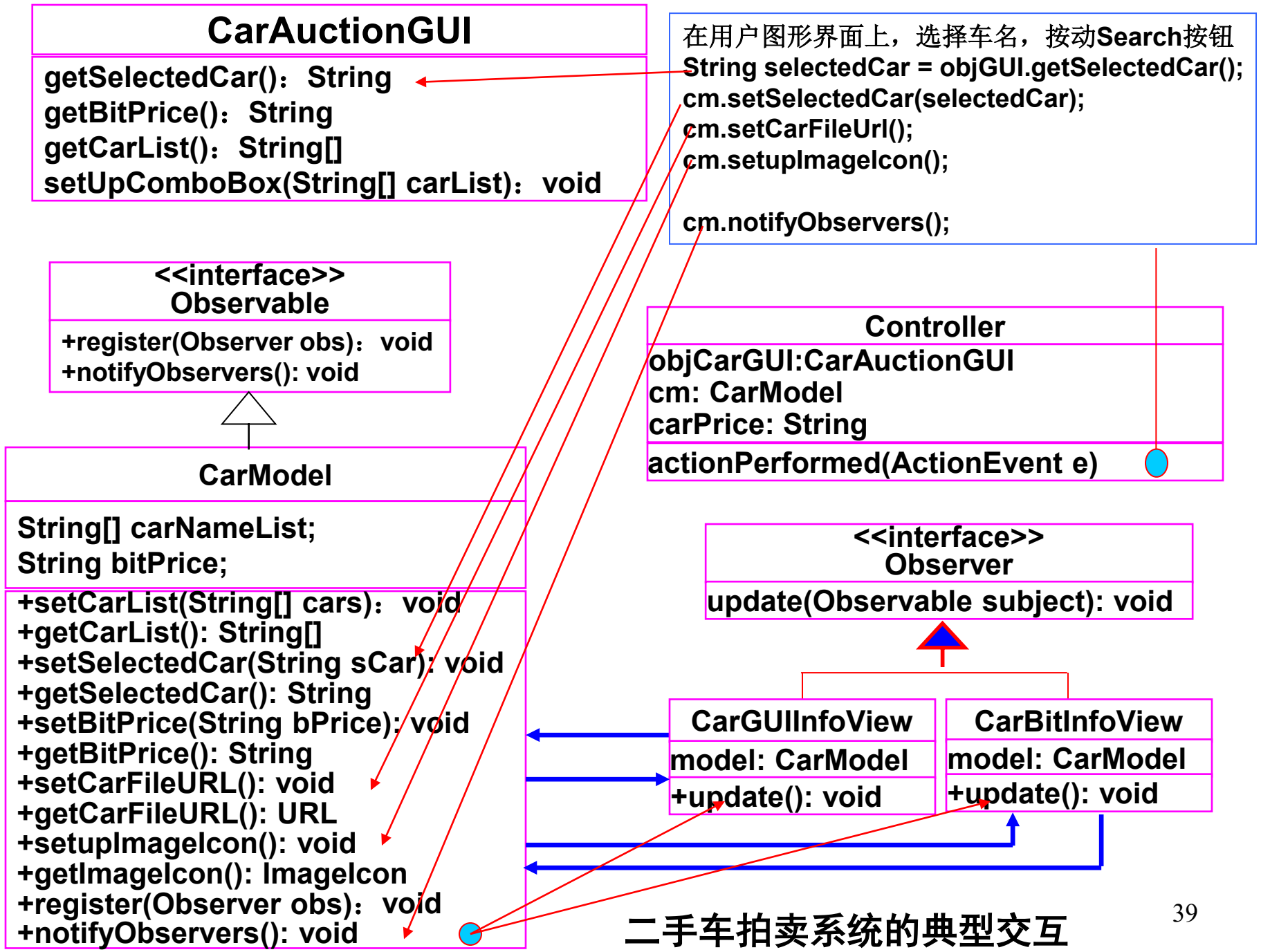
二手车当前拍卖价格显示界面原型（Prototype）



二手车拍卖系统的类图设计



二手车拍卖系统的典型交互



二手车拍卖系统的典型交互

网上二手车拍卖系统的设计：方案3

设计方案3工作原理

a) **用户输入.** 用户通过使用CarAuctionGUI中的Cars列表选择待卖的车，点击Search，以便获取车的图片；然后输入拍价，点击Bit，给出价格。

b) **捕捉用户输入.** 每次点击按钮产生的事件，都被Controller对象捕捉，然后调用actionPerformed()方法。

c) **更新模型数据.** 在actionPerformed()方法中，将根据事件的类型，分别使用setSelectedCar 或者setBitPrice 方法更新CarModel的数据

d) **Model通知View.** 由于CarGUIView、CarBitView与CarModel之间存在观察者/被观察者的关系，所以当CarModel的状态改变了的时候，CarGUIView和CarBitView的update()方法将被自动调用。

e) **更新View.** 在update()方法中，调用CarModel的getSelectedCar 或者getBitPrice 方法得到最新的数据，然后根据数据，更新视图。

网上二手车拍卖系统的设计：方案3

- **设计要点**
- CarModel实现了interface Observable，实现了方法notifyObservers()和register(Observer obs)。
- CarGUIInfoView和CarBitInfoView实现了interface Observer, 实现了方法update()。
- 每个CarGUIInfoView和CarBitInfoView都将自己注册为CarModel的观察者。每当CarModel的状态改变的时候，其方法notifyObservers将改变告诉给它所有的观察者，观察者的update()可以自动查询CarModel，了解到底是什么状态改变了，从而作出响应的反应。
- **事实上，notifyObservers()的代码中，直接调用了观察者中的update()方法。以下是代码。**

```
public class CarAuctionGUI extends JFrame {
static public void main(String argv[]) {
    javax.swing.SwingUtilities.invokeLater(new
   Runnable() {
        public void run() {
          cm = new CarModel();
          cgiv = new CarGUIInfoView(cm);
          cbiv = new CarBitInfoView(cm);
          cm.register(cgiv); // 将视图对象注册到模型
          cm.register(cbiv); // 将视图对象注册到模型

          CarAuctionGUI frame = new CarAuctionGUI();
           }
      });
}
```

```
class Controller implements ActionListener {
   private CarAuctionGUI objCarGUI;
   private CarModel cm;
   private String carPrice;
   public void actionPerformed(ActionEvent e){
      String searchResult = null;
      if (e.getActionCommand().equals(SEARCH)){
         String selectedCar = objCarGUI.getSelectedCar();
         cm.setSelectedCar(selectedCar); //更新模型数据
         cm.setCarFileUrl();
         cm.setupImageIcon();
         cm.setSearchBtnClickInfo(true);
         cm.notifyObservers(); //通知观察者
         cm.setSearchBtnClickInfo(false);
      }
      if (e.getActionCommand().equals(CarAuctionGUI.BIT)){
         carPrice = objCarGUI.getBitPrice();
         cm.setBitPrice(carPrice); //更新模型数据
         cm.setBitBtnClickInfo(true);
         cm.notifyObservers(); //通知观察者
         cm.setBitBtnClickInfo(false);
      } }}
```

```
public class CarModel implements Observable{
    private ArrayList<Observer> observersList;
    private ImageIcon imgIcon;
    private URL url;
    private String[] carNameList;
    private String carSelected;
    private String bitPrice;
    private boolean isBitBtnClicked = false;
    private boolean isSearchBtnClicked = false;
    static final String CARFILES = "CarFiles/";
    static final String CARIMAGES = "CarImages/";

    public CarModel(){
        observersList = new ArrayList<Observer>();
        carNameList=new String[200];
    }
    public void setCarList(String[] cars){
        carNameList = cars;
    }
    public String[] getCarList(){
        return  carNameList;
    }
```

CarModel代码续

```
public void setSelectedCar(String sCar){
   carSelected = sCar;
}
public String getSelectedCar(){
    return carSelected;
}
public void setBitPrice(String bPrice){
      bitPrice = "";
      bitPrice = bitPrice + bPrice;
}
public String getBitPrice(){
      return bitPrice;
}
public void setCarFileUrl(){
      String fileURLStr=CARFILES+carSelected+".html";
      File file=new File(fileURLStr);
      URI uri = file.toURI();
      url = uri.toURL();
}
```

CarModel代码续

```
public URL getCarFileURL(){
                return url;
}
public void setupImageIcon(){
    String iconStr = CARIMAGES + carSelected+".jpg";
    imgIcon = createImageIcon(iconStr);
}
public ImageIcon getImageIcon(){
    return imgIcon;
}
public void setBitBtnClickInfo(boolean opt){
   isBitBtnClicked = opt;
}
public boolean isBitBtnClicked(){
   return isBitBtnClicked;
}
public void setSearchBtnClickInfo(boolean opt){
   isSearchBtnClicked = opt;
}
public boolean isSearchBtnClicked(){
   return isSearchBtnClicked;
}
```

CarModel代码续

```
public void register(Observer obs){
        observersList.add(obs);
}
public void notifyObservers() {
    for (int i = 0; i < observersList.size(); i++) {
        Observer observer = (Observer);
        observersList.get(i);
        observer.update(this);
    }
}
protected ImageIcon createImageIcon(String path){
   URL imgURL = getClass().getResource(path);
   if (imgURL != null) {
       return new ImageIcon(imgURL);
   } else {
       return null;
   }
}} // End of class
```

```
class CarBitInfoView extends JFrame implements Observer{
    private JLabel bitLabel;
    private JTextArea bitText;
    private CarModel model;
    private JScrollPane textPane;

    public CarBitInfoView(CarModel cmodel) {
      model = cmodel;

      bitLabel = new JLabel("Bits offered:");
      bitText = new JTextArea(4, 20);
      bitText.setFont(new Font("Serif", Font.PLAIN, 14));

      JScrollPane textPane = new JScrollPane(bitText);
      Container contentPane = getContentPane();
      contentPane.add(bitLabel, BorderLayout.NORTH);
      contentPane.add(textPane, BorderLayout.CENTER);
      setVisible(true);
    }
```

```
//CarBitInfoView代码续
   public void update(Observable subject){
     if((subject==model)&&(model.isBitBtnClicked())){
        //从model获得数据
        String sCar= model.getSelectedCar();
        String pr = model.getBitPrice();
        //显示在视图上
        bitText.append("\n Bit price for "+ sCar + "="+ pr);
       }
     }
}// end CarBitInfoView
```

```
public class CarGUIInfoView extends JFrame
   implements Observer{
  private JEditorPane editorPane， imagePane， filePane;
  private JSplitPane splitPane;
  private JLabel imgLabel;
  private CarModel model;
  public CarGUIInfoView(CarModel cmodel){
       model = cmodel;
       buildUpScrollGUI();
  }
  public void update(Observable subj){
    if((subj==model)&&(model.isSearchBtnClicked())){

       URL url = model.getCarFileURL();
       editorPane.setPage(url);
       //从model获得数据
       ImageIcon imIcon = model.getImageIcon();
       imgLabel.setIcon(imIcon); //显示在视图上
       imgLabel.validate();
    }
  }}
```