MSc/ICY Software Workshop
Graphical User Interfaces

Manfred Kerber www.cs.bham.ac.uk/~mmk

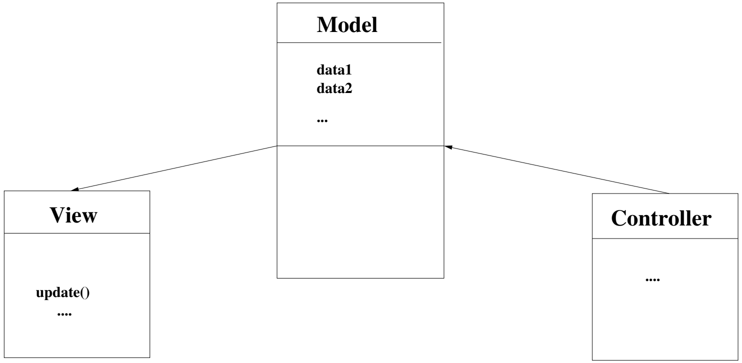
(Slides mainly taken from Jon Rowe's corresponding lecture in Software Workshop 1, 2015, with some modifications)

30 November 2016

Model - View – Control

- When creating a graphical user interface (GUI) for a program, we try to keep the program functionality (model) separate from the display elements (views) and interaction elements (controls).
- We use the Observer – Observable pattern to do this – it enables a clean separation that is easy to maintain.

Model - View – Controller (Cont'd)



Taken from Walter Savitch, Absolute Java, p.1006

Observer – Observable

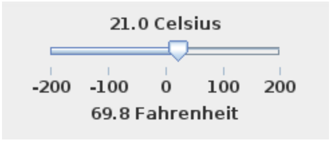
- We “wrap” the underlying program objects into a “model” object.
- The model object is from a class that extends `Observable`.
- This means that it can have observers (using `addObserver`).
- When a change occurs, the observers are notified:
  ```
  setChanged();
  notifyObservers();
  ```

Views are observers

- We make any view class extend the type of view it is (e.g. `JLabel`), and implement the `Observer` interface.
- This means it must provide a method:
  `public void update(Observable obs, Object obj)`
- This is the method that is run when `notifyObservers()` is called.

Controls and Listeners

- Elements that the user can interact with are called controls.
- Often you can use the built-in control class directly (e.g. `JSlider`), but you can create your own class for them if you want to add extra functionality.



- Each control needs a listener to handle the interactions.

Listeners and events

- When the user interacts with a control, it generates an "`event`" object.
- For example, moving the slider creates a `ChangeEvent` object.
- The compiler looks to see if there is a listener attached to the control that can handle the event.
- If so, the appropriate method is run.

Implementing Listeners

- Listeners are interfaces.
- There are different listener interfaces for different kinds of control and event.
- For example, to handle a `ChangeEvent`, you need a `ChangeListener`.
- This requires you to provide the appropriate method: `public void stateChanged(ChangeEvent e)`

The sequence of events

- The user interacts with a control.
- This generates an event.
- The listener handles the event and runs its method.
- This creates changes in the model.
- The model's set methods call `notifyObservers`.
- All the views that are observing the model run their update method.
- This causes them to re-draw themselves appropriately.

Recipe for writing a GUI

- Create model class (wraps underlying program objects, abstracts the program functionality).
- Create view classes, one for each view
- Create control classes, if needed.
- Create listener classes, one for each event.
- Create component class to house everything.
- Create GUI class with main method.

Create model class

- Your model class should have as data fields all the program objects you need.
- It should `extend Observable`.
- Provide wrapper methods for the get methods you need for your GUI.
- Provide wrapper set methods.
- The set methods should call

  ```
  setChanged();
  notifyObservers();
  ```

Create view classes

- For each disjoint view, you need a separate class.
- The class should extend the type of component you want it to be (e.g. `JLabel`).
- It should implement `Observer`.
- You have to provide the update method, which says how to redraw itself if it is told the model has changed.

Create control classes

- For each distinct control, you need to decide whether it needs its own class, or if you can use the built in class directly (e.g. `JSlider`).
- You only need a new class (which extends the component class) if you are adding functionality.
- For example, this might happen if the control is also a view.

Create listener classes

- For each different control and different event type, you need a listener.
- You have to look up what kind of listener goes with what kind of control and event.
- The class should implement that listener.
- It is helpful for it to have the model and the control as data fields.
- You need to implement the appropriate method to handle the event. Typically it changes the model.

Create component class

- Typically we use `JPanel`, rather than `JComponent`.
- This allows several components to be displayed at once.
- You need the constructor, to create all the parts and plug them together.
- If you have any extra graphics, you might need to override `paintComponent`.

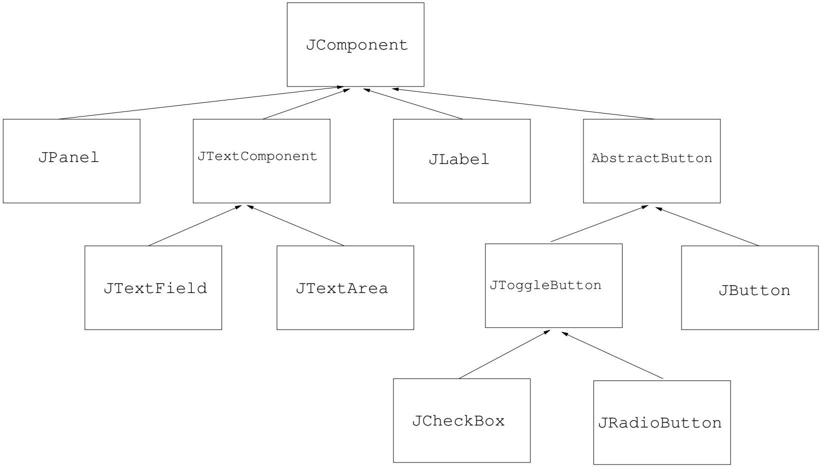
Component constructor recipe

- Should take program objects as arguments.
- Call `super();`
- Create model objects.
- Create view objects.
- Make views observe the model.
- Create control objects.
- Create listener objects.
- Make listeners listen to controls.
- Put views and controls on to panel.

Create GUI

- In this class you will have your main method.
- This should create program objects.
- Then create the component (JPanel).
- Then create the frame.
- Put the component on the frame.
- Make everything visible.

Create GUI (Cont'd)



A Part of the Hierarch of Swing User-Interface Components (taken from Horstmann "Big Java" 4th Edition, p. 721).

Getting organized

- The key to writing a good GUI is to get organized.
- Sketch how you want it to look on paper.
- Then make a list of all the views, controls and listeners you will need.
- Write them one by one, working your way through the list.
- Do not forget to plug everything together!

Note on Observer pattern

- The Observer pattern allows a very clean separation of Model – View – Control.
- However, it is not necessarily very efficient (why?).
- Developers often do not use it – instead they make their control listeners directly manipulate views as well as the model.
- This is more efficient – but potentially messier to write and maintain.