CompSci 230
Software Construction

Swing and MVC S1 2015

Learning Goals

- You will learn an architectural design pattern: the MVC
  - You will start to understand why high-level design is important
- You will learn how Swing implements MVC
  - You will start to understand why architecture is an art, not a science.

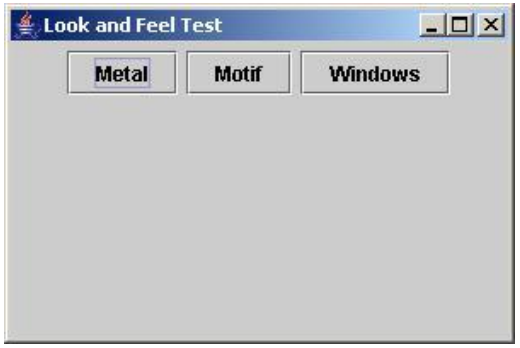
Swing

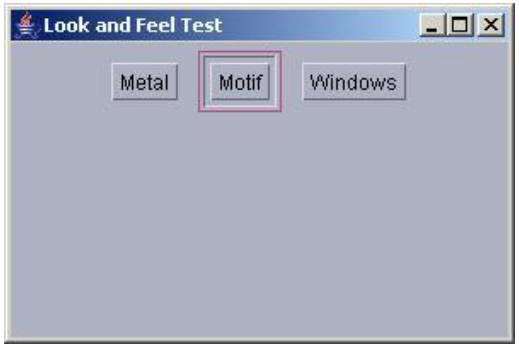
- "The overall goal for the Swing project was:
  - *To build a set of extensible GUI components to enable developers to more rapidly develop powerful Java front ends for commercial applications.*
- "To this end, the Swing team established a set of design goals early in the project that drove the resulting architecture. These guidelines mandated that Swing would:

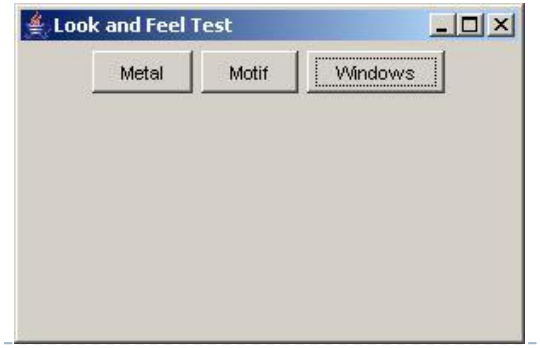
1. ***Be implemented entirely in Java*** to promote cross-platform consistency and easier maintenance.
2. ***Provide a single API capable of supporting multiple look-and-feels*** so that developers and end-users would not be locked into a single look-and-feel.
3. ***Enable the power of model-driven programming*** without requiring it in the highest-level API.
4. ***Adhere to JavaBeans™ design principles*** to ensure that components behave well in IDEs and builder tools.
5. ***Provide compatibility with AWT APIs*** where there is overlapping, to leverage the AWT knowledge base and ease porting."

[Amy Fowler, "A Swing Architecture Overview", Sun Microsystems, 2002. A corrupted version is available at http://www.oracle.com/technetwork/java/architecture-142923.html, April 2015. Archival version: http://web.archive.org/web/20020809043740/http://java.sun.com/products/jfc/tsc/articles/architecture/index.html.]

Pluggable look and feel

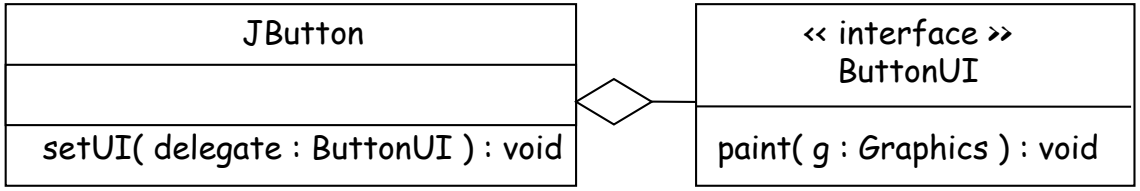






With a Swing application, it is possible to change the look-and-feel at run-time.

What technique do you think has been used in implementing this feature?

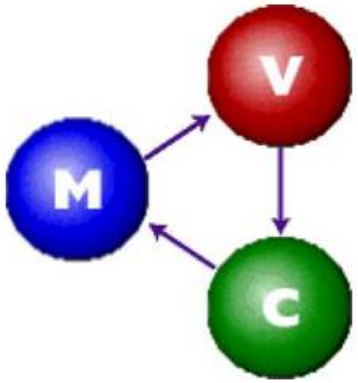


Implemented by one class for each different look-and-feel, e.g. Windows, Metal, Motif.

Swing's Model-based Architecture

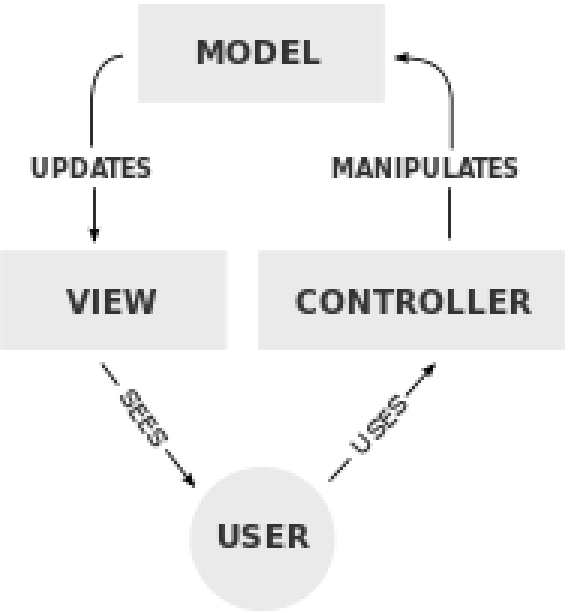
- "Swing architecture is rooted in the *model-view-controller* (*MVC*) design that dates back to SmallTalk.
- "MVC architecture calls for a visual application to be broken up into three separate parts:
  - A *model* that represents the data for the application
  - The *view* that is the visual representation of that data
  - A *controller* that takes user input on the view and translates that to changes in the model."

[Amy Fowler, ibid.]



MVC: According to Wikipedia

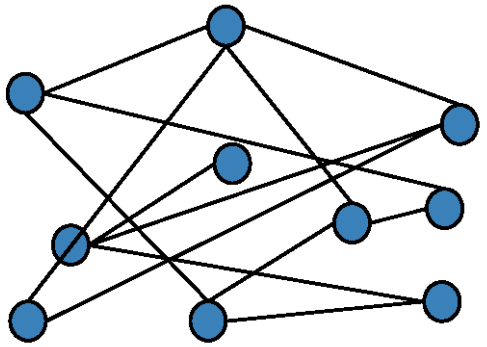
- A **controller** can send commands to the model to update the model's state (e.g., editing a document).
  - It can also send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document).
- A **model** notifies its associated views and controllers when there has been a change in its state.
  - This notification allows the views to produce updated output, and the controllers to change the available set of commands.
  - In some cases an MVC implementation may instead be 'passive' and other components must poll the model for updates rather than being notified.
- A **view** requests information from the model that it uses to generate an output representation to the user.



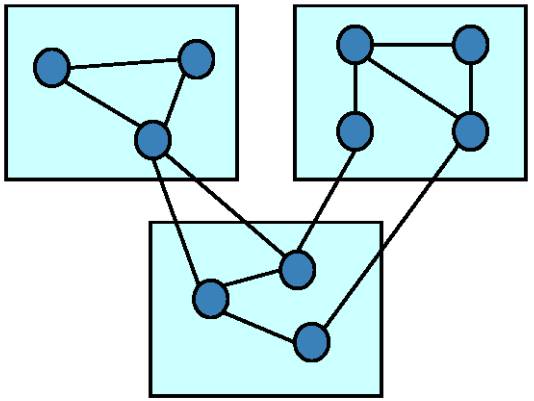
[http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller]

Spaghetti Code vs Modular Design

- Spaghetti Code
  - Haphazard connections, probably grown over time
  - No visible cohesive groups
  - High coupling: high interaction between random parts
  - Understand it: all or nothing
- Modular System
  - High cohesion within modules
  - Low coupling between modules
  - Modules can be understood separately
  - Interaction between modules is easily-understood and thoroughly specified

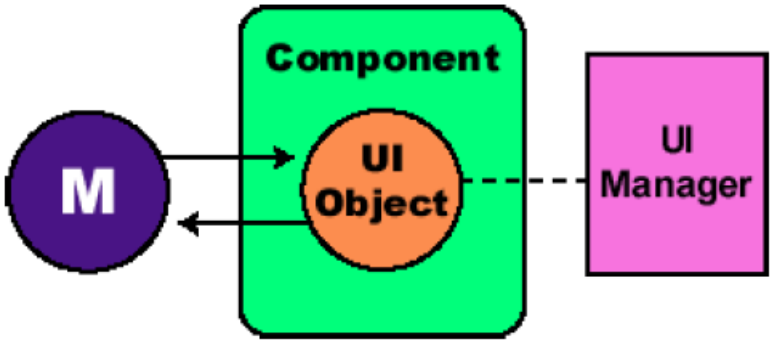


Both examples have 10 modules and 13 connections!



Architectural Design History of Swing [Fowler, *ibid.*]

- "The first Swing prototype followed a traditional MVC separation in which each component
  - had a separate model object and
  - delegated its look-and-feel implementation to separate view and controller objects.
- "I quickly discovered that this split didn't work well in practical terms
  - because the view and controller objects required a tight coupling
  - (for example, it was very difficult to write a generic controller that didn't know specifics about the view).
- "So I collapsed these two entities into a single UI (user-interface) object, as shown in this diagram:"



Separable Model Architecture [Fowler, *ibid.*]

- "In the world of Swing, this new quasi-MVC design is sometimes referred to as a *separable model architecture*.
- "Swing's separable model design treats the model part of a component as a separate element, just as the MVC design does.
  - But Swing collapses the view and controller parts of each component into a single UI (user-interface) object.
- "… as an application developer, you should think of a component's view/controller responsibilities as being handled by the generic component class (such as `Jbutton`, `JTree`, and so on).
  - The component class then delegates the look-and-feel specific aspects of those responsibilities to the UI object that is provided by the currently installed look-and-feel." [Amy Fowler, ibid.]

GUI-state models, Application-data models

- "*GUI state models* are interfaces that define the visual status of a GUI control, such as
  - whether a button is pressed or armed, or
  - which items are selected in a list.
- "An *application-data model* is an interface that represents some quantifiable data that has meaning primarily in the context of the application, such as
  - the value of a cell in a table or
  - the items displayed in a list.
- "Of course, with some components, the model categorization falls somewhere between GUI state models and application-data models … This is the case with
  - the `BoundedRangeModel` or
  - `JProgressBar`."

Learning Goals: Review

- You will learn an architectural design pattern: the MVC
  - You will start to understand why high-level design is important
- You will learn how Swing implements MVC
  - You will start to understand why architecture is an art, not a science.