Quick Start
Patterns
Observer -- Problem

The observer pattern (aka publish/subscribe) involves an object, called the subject, which maintains a list of its dependents, called observers, and notifies them automatically of any state changes.

Do you know the problem?
Do you know a solution?
Observer -- Solution


(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)
Proxy -- Problem

- Additional access behavior for objects
- Access management for remote objects
- Access management for expensive objects
Proxy -- Solution


(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)
Copyright notice

These slide decks are derived from text parts, figures and source fragments from the following online article: © Robert Eckstein: Java SE Application Design With MVC, Sun Developer Network (SDN), http://java.sun.com/developer/technicalArticles/javase/mvc/, March 2007. If you derive any work from the present slide deck, you must keep the copyright notice in tact. All remaining copyright, if any, is with Ralf Lämmel.

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)
Elevator speech

If you do business programming with graphical user interfaces (GUIs), you need the model-view-controller (MVC) design; it helps you to decouple data access and business logic from the manner in which it is displayed to the user.
Motivation: an editor

- Data is stored in objects.
- One view may change data so that other view needs update.

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)
Motivation: an editor

- Data is stored in objects.
- One view may change data so that other view needs update.

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)
Straightforward implementation

- Design
  - Use **GUI listeners** to access data.
  - Use **Observer pattern** to notify editor of changes.
Straightforward implementation

- Design
  - Use *GUI listeners* to access data.
  - Use *Observer pattern* to notify editor of changes.

- Consequences
  - GUI is locked into the data access layer.
  - Restrictions of user actions hard-coded into GUI.
  - Multiple views are hard to sync efficiently.
Model View Controller

MVC is an important architectural or design pattern for decoupling data model (data access), business logic in an application and presentation in the user interface.

- The view submits gestures to controller.
- The view registers with model for data changes.
The elements of MVC

• **Model** - The model represents data and the rules that govern access to and updates of this data. In enterprise software, a model often serves as a software approximation of a real-world process.

• **View** - The view renders the contents of a model. It specifies exactly how the model data should be presented. If the model data changes, the view must update its presentation as needed. This can be achieved by using a *push model*, in which the view registers itself with the model for change notifications, or a *pull model*, in which the view is responsible for calling the model when it needs to retrieve the most current data.

• **Controller** - The controller translates the user's interactions with the view into actions that the model will perform. In a stand-alone GUI client, user interactions could be button clicks or menu selections, whereas in an enterprise web application, they appear as GET and POST HTTP requests. Depending on the context, a controller may also select a new view -- for example, a web page of results -- to present back to the user.

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)
The elements of MVC

- **Model** - The model represents data and the rules that govern access to and updates of this data. In enterprise software, a model often serves as a software approximation of a real-world process.

- **View** - The view renders the contents of a model. It specifies exactly how the model data should be presented. If the model data changes, the view must update its presentation as needed. This can be achieved by using a *push model*, in which the view registers itself with the model for change notifications, or a *pull model*, in which the view is responsible for calling the model when it needs to retrieve the most current data.

- **Controller** - The controller translates the user's interactions with the view into actions that the model will perform. In a stand-alone GUI client, user interactions could be button clicks or menu selections, whereas in an enterprise web application, they appear as GET and POST HTTP requests. Depending on the context, a controller may also select a new view -- for example, a web page of results -- to present back to the user.
Although different architectures allow the three components to interact in different ways, the following figure shows a common implementation of the MVC design pattern, as shown in the Sun BluePrints Catalog.
Registration and dependencies

1. The view registers as a listener on the model. Any changes to the underlying data of the model immediately result in a broadcast change notification, which the view receives. Note that the model is not aware of the view or the controller -- it simply broadcasts change notifications to all interested listeners.

2. The controller is bound to the view. This typically means that any user actions that are performed on the view will invoke a registered listener method in the controller class.

3. The controller is given a reference to the underlying model so that it can issue model changes, for example. The view will not change the model directly, but always let the controller do it. The view may read data from the model, though.
Actions and events

1. The view recognizes that a GUI action -- for example, pushing a button or dragging a scroll bar -- has occurred, using a listener method that is registered to be called when such an action occurs.

2. The view calls the appropriate method on the controller.

3. The controller accesses the model, possibly updating it in a way appropriate to the user's action.

4. If the model has been altered, it notifies interested listeners, such as the view, of the change. In some architectures, the controller may also be responsible for updating the view. This is common in Java technology-based enterprise applications.
Two views that use the same model

```java
int xPosition = 50;
int yPosition = 50;
double opacity = 0.35;
Font font = new Font("Arial", Font.BOLD, 24);
String text = "Some Transparent Text";
```