CS193E
Lecture 8

Undo & Redo
NSError
More Key-Value Coding

Today's Topics

- Questions?
- FavoriteThings Review
- Personal Timeline Overview
- Undo & Redo
- NSError
- Miscellaneous

FavoriteThings Review

How real-world was it?

Overall a well-behaved application

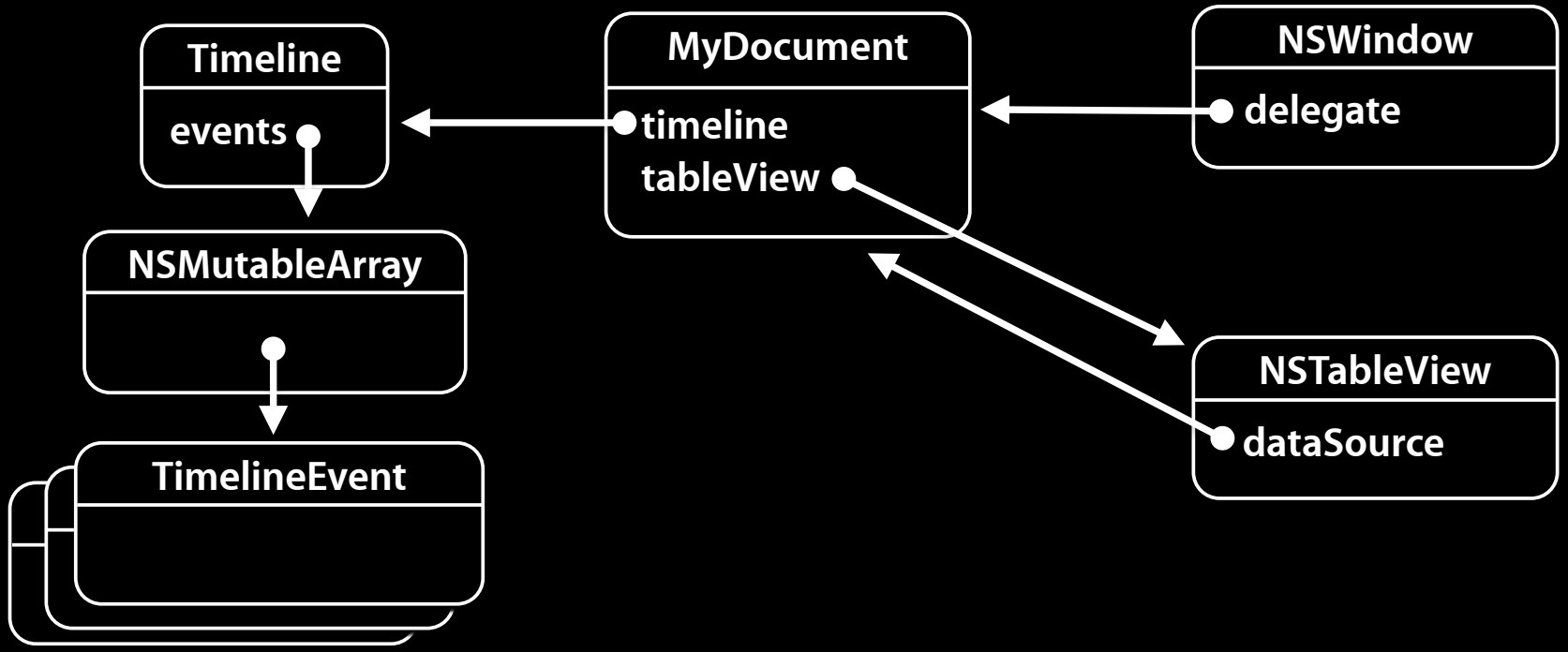
- Standard interface elements
- Uses correct place in file system for application data
- Reasonable interface for unsaved changes

Some things left out of Favorite Things

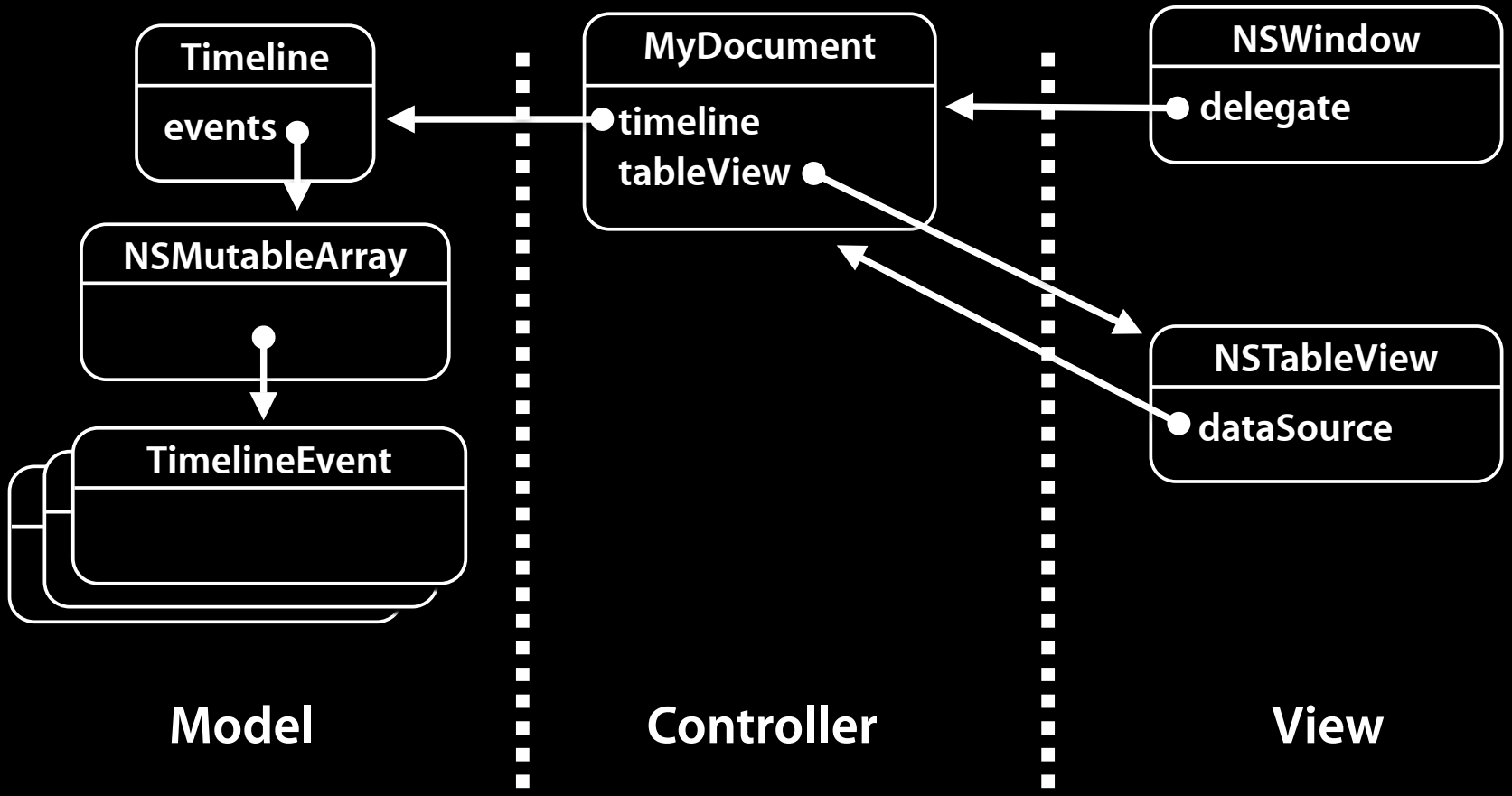
- A single-window app like that usually autosaves changes
  - iTunes
  - iPhoto
- Error checking
- Image would usually be stored separately, not in archive
- Various ‘fit and finish’ issues
  - Better handling of editing (text did change notification)
  - Closing window causes unsaved changes check, quitting app doesn’t (app should terminate delegate method).
  - Might not let user choose last-window quit behavior
  - No way to get window back once you’ve closed it

PersonalTimeline Architecture

Personal Timeline



Personal Timeline



Undo

and its much overlooked counterpart, redo

NSUndoManager

- Records operations on objects
- As changes are made and operations are recorded, they're pushed onto an undo stack
- When an undo is requested, the top item on the stack is taken off and its operations are performed
- Undo stack is unlimited by default

Undo and NSDocument

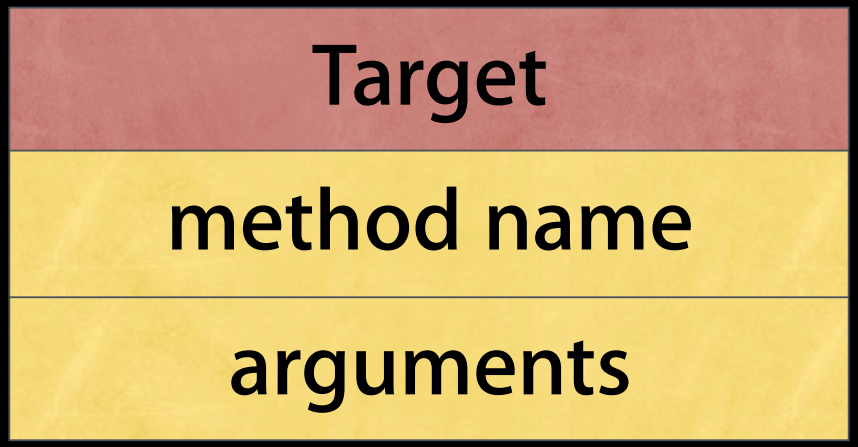
- Default undo manager provided by document

  - (NSUndoManger *)undoManager;

- NSDocument monitors undo manager to track dirty state automatically
- If nothing is on the undo stack, the document is clean
- If there are undo operations on the stack, it's dirty

Undo Operations

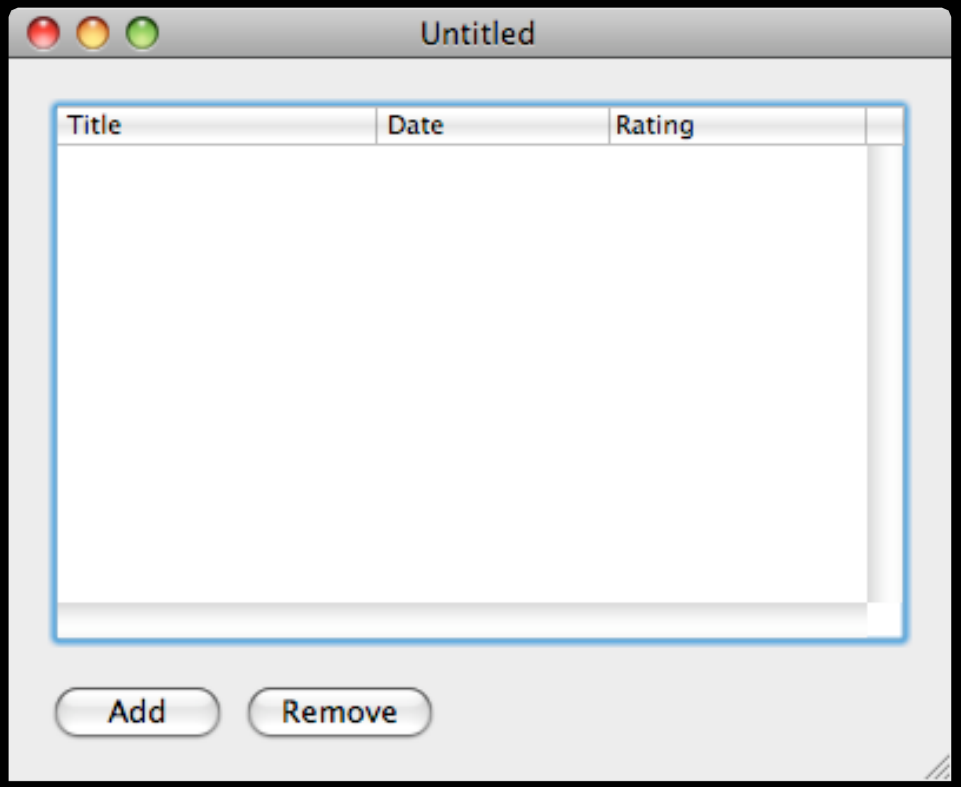
- An undo operation is a collection of everything needed to revert a change
- Not a recording of what happened — it's how to undo what happened
- Undo operations are a composed of



Undo Example

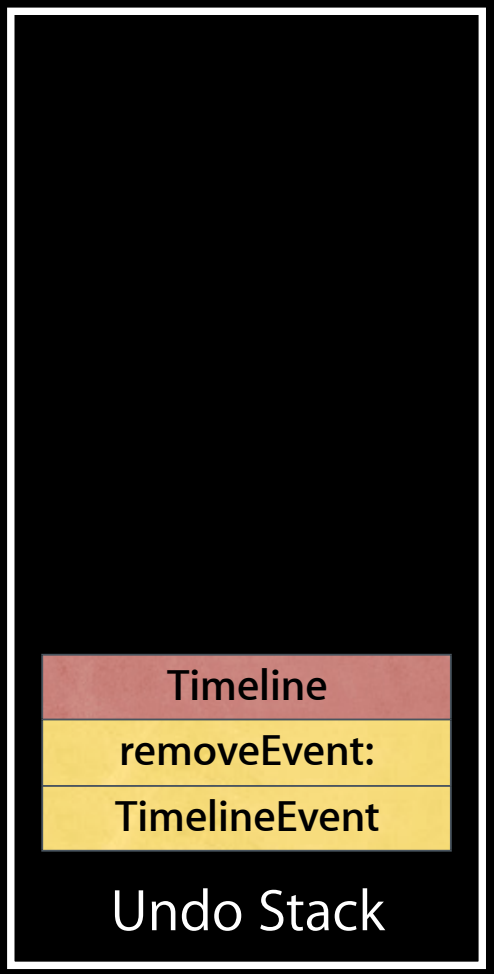
Psuedocode - don't do exactly this in your code

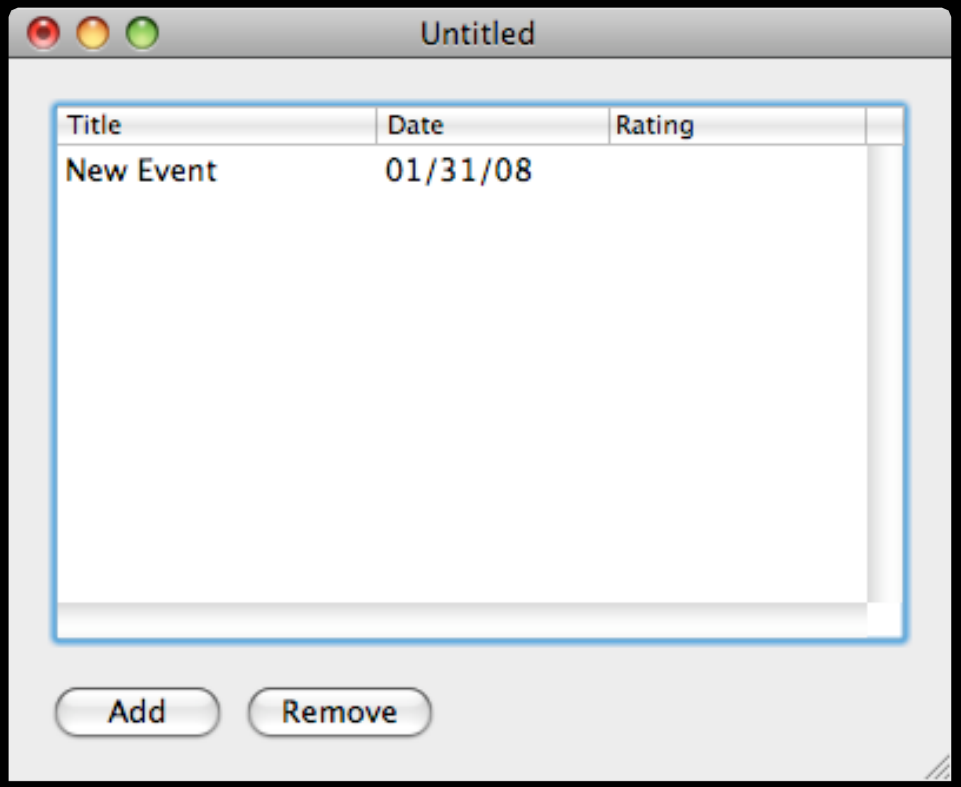
Undo Stack



Undo Example

Psuedocode - don't do exactly this in your code





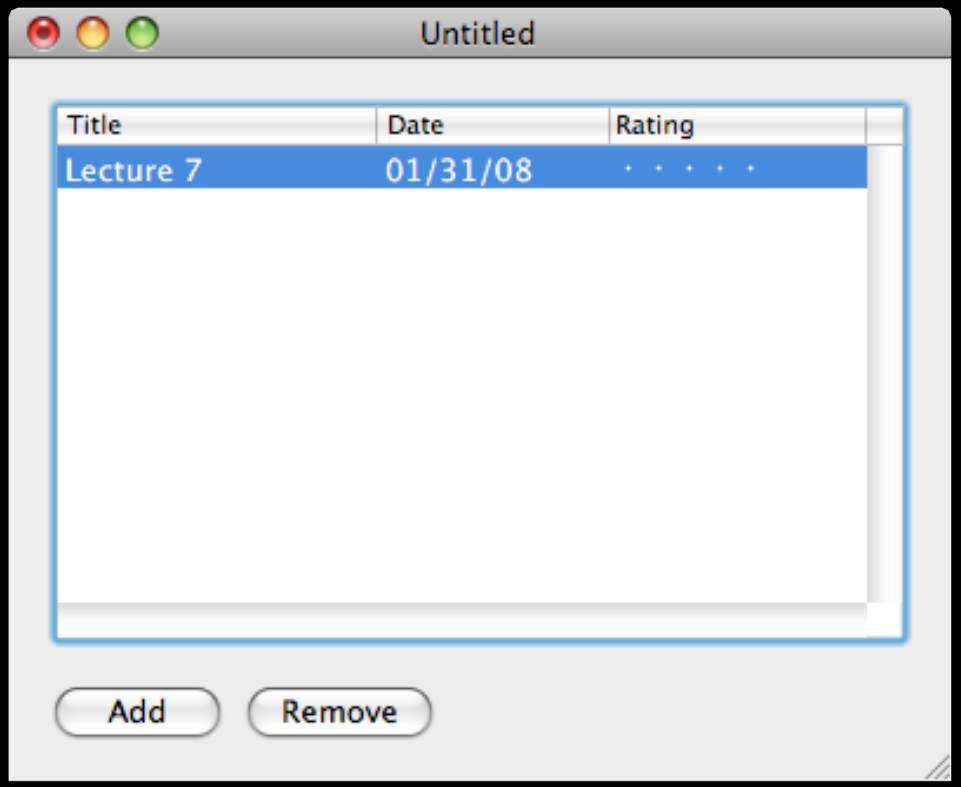
Undo Example

Psuedocode - don't do exactly this in your code

TimelineEvent
setTitle:
@"New Event"

Timeline
removeEvent:
TimelineEvent

Undo Stack



Undo Example

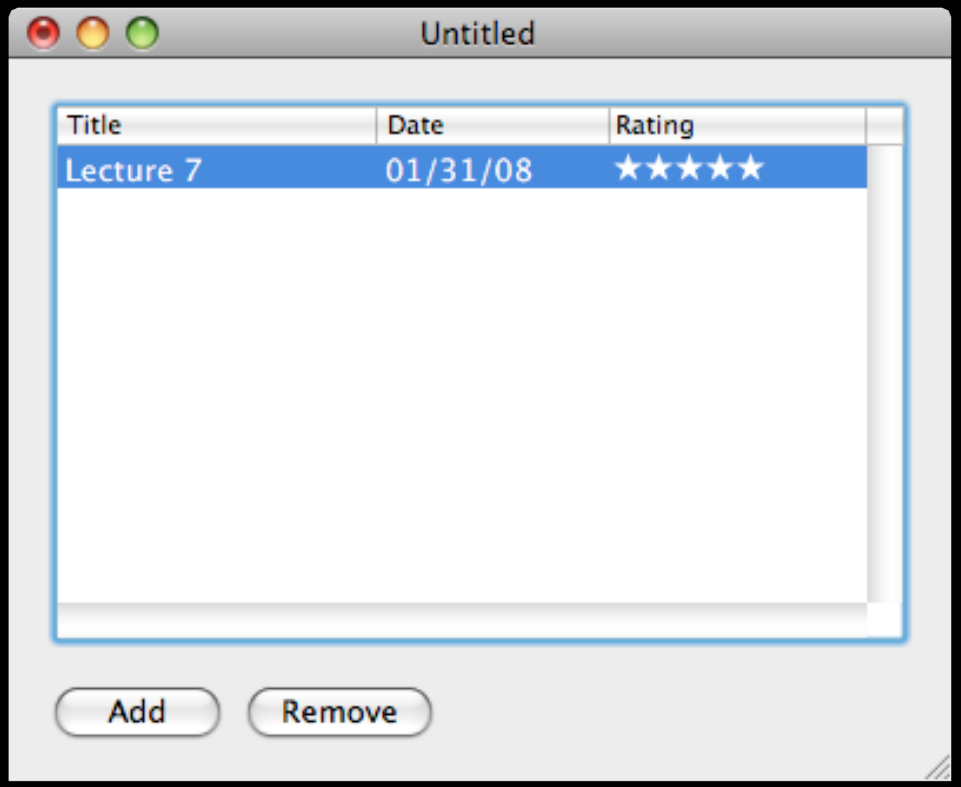
Psuedocode - don't do exactly this in your code

TimelineEvent
setRating:
0

TimelineEvent
setTitle:
@"New Event"

Timeline
removeEvent:
TimelineEvent

Undo Stack



Redo Example

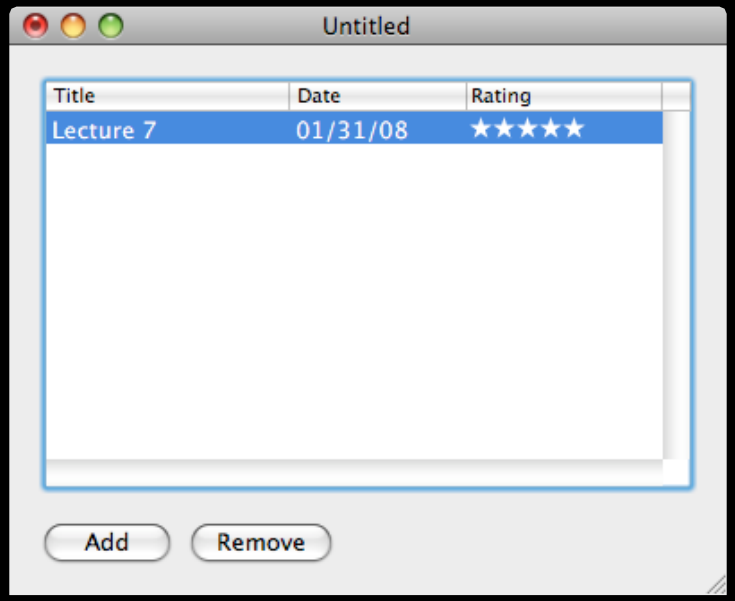
Psuedocode - don't do exactly this in your code

TimelineEvent
setRating:
0

TimelineEvent
setTitle:
@"New Event"

Timeline
removeEvent:
TimelineEvent

Undo Stack



Redo Stack

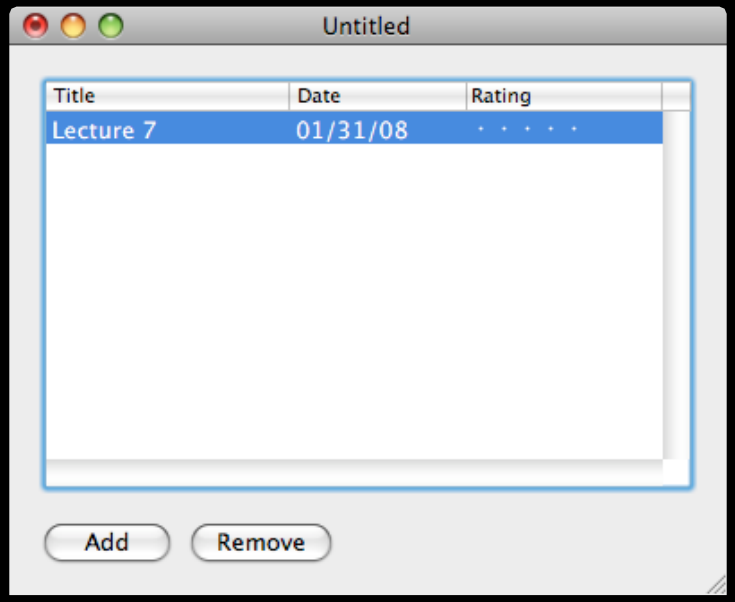
Redo Example

Psuedocode - don't do exactly this in your code

TimelineEvent
setTitle:
@"New Event"

Timeline
removeEvent:
TimelineEvent

Undo Stack



TimelineEvent
setRating:
5

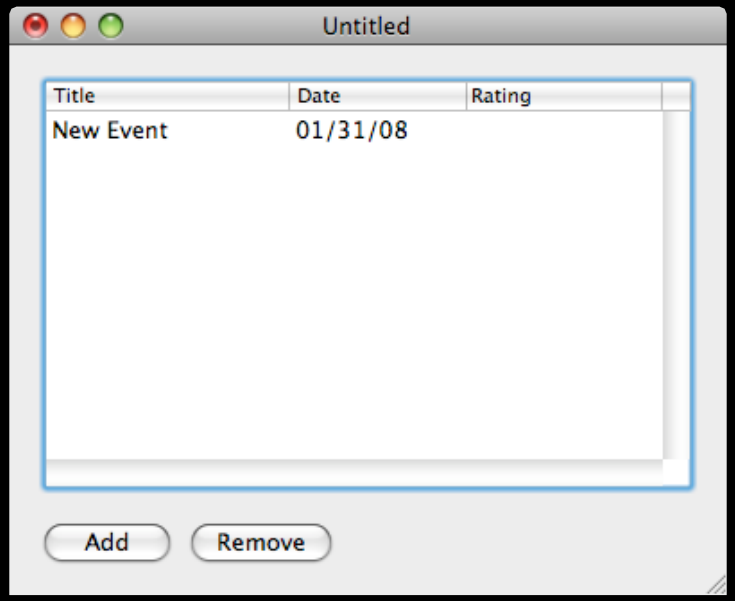
Redo Stack

Redo Example

Psuedocode - don't do exactly this in your code

Timeline
removeEvent:
TimelineEvent

Undo Stack



TimelineEvent
setTitle:
@"Lecture 7"

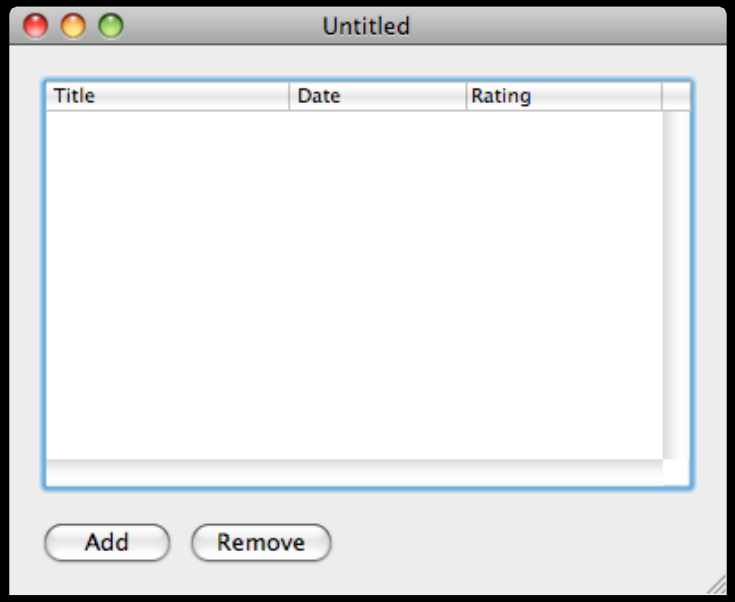
TimelineEvent
setRating:
5

Redo Stack

Redo Example

Psuedocode - don't do exactly this in your code

Undo Stack



Timeline
addEvent:
TimelineEvent

TimelineEvent
setTitle:
@"Lecture 7"

TimelineEvent
setRating:
5

Redo Stack

Registering operations

- Two methods for registering undo operations
- The first method is very direct

```
-(void)registerUndoWithTarget:(id)target
          selector:(SEL)selector object:(id)object
```

- For example

```
[undoManager registerUndoWithTarget:person
        selector:@selector(setName:) object:@"Bob"];
```

Registering operations

What about methods that take multiple arguments?

- Second method is easy to use, and conceptually intriguing

```
- (id)prepareWithInvocationTarget:(id)target
```

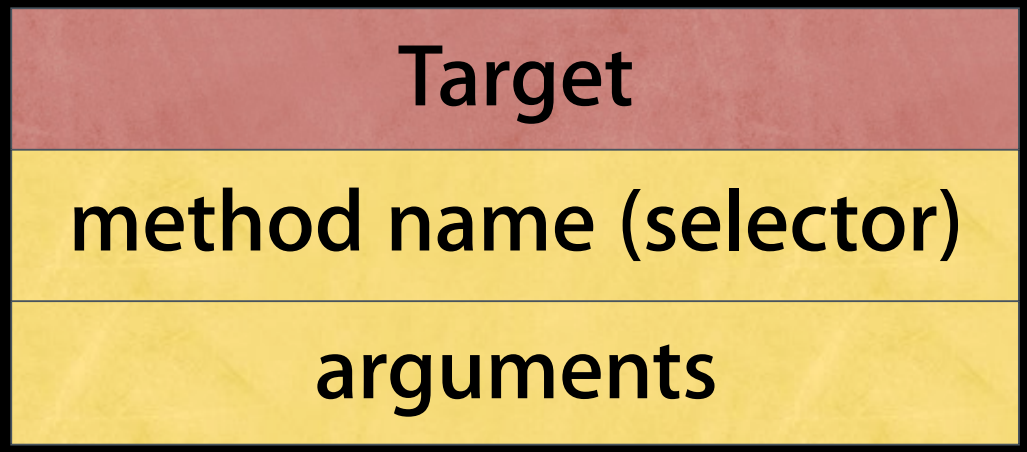
- For example:

```
[[undoManager prepareWithInvocationTarget:person]
                setFirstName:@"Bob" lastName:@"Jones"];
```

- What's going on here?
  - First method provides target, returns a ‘prepped’ undo manager
  - Selector and args of next message not sent, but packaged up

NSInvocation

- An encapsulation of a method call and the arguments passed to the method
- Contains the following:



Hmmmm, where have we seen that before?

Undo uses NSInvocation

- Undo manager builds an NSInvocation
- The invocation records the method and arguments
- Undo manager pushes the invocation on the undo stack
- To undo something, an invocation is taken off the stack and "invoked", which performs the original method call

Descriptive Names

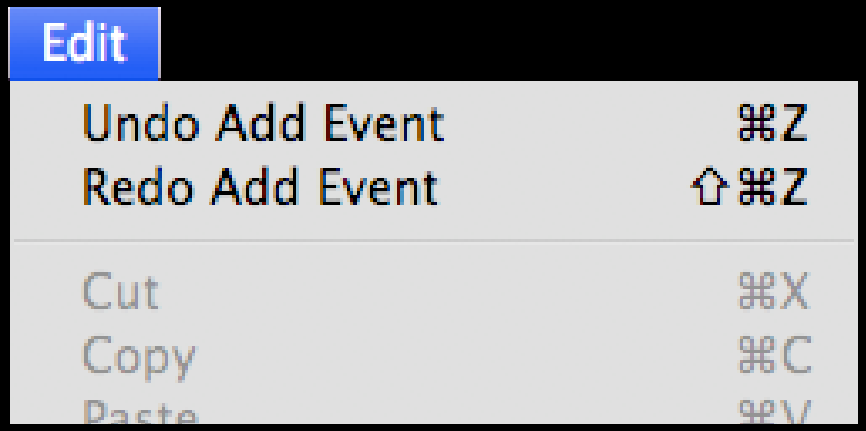
- Used for menu items to provide a human-readable name

```
- (void)setActionName:(NSString *)name;
```

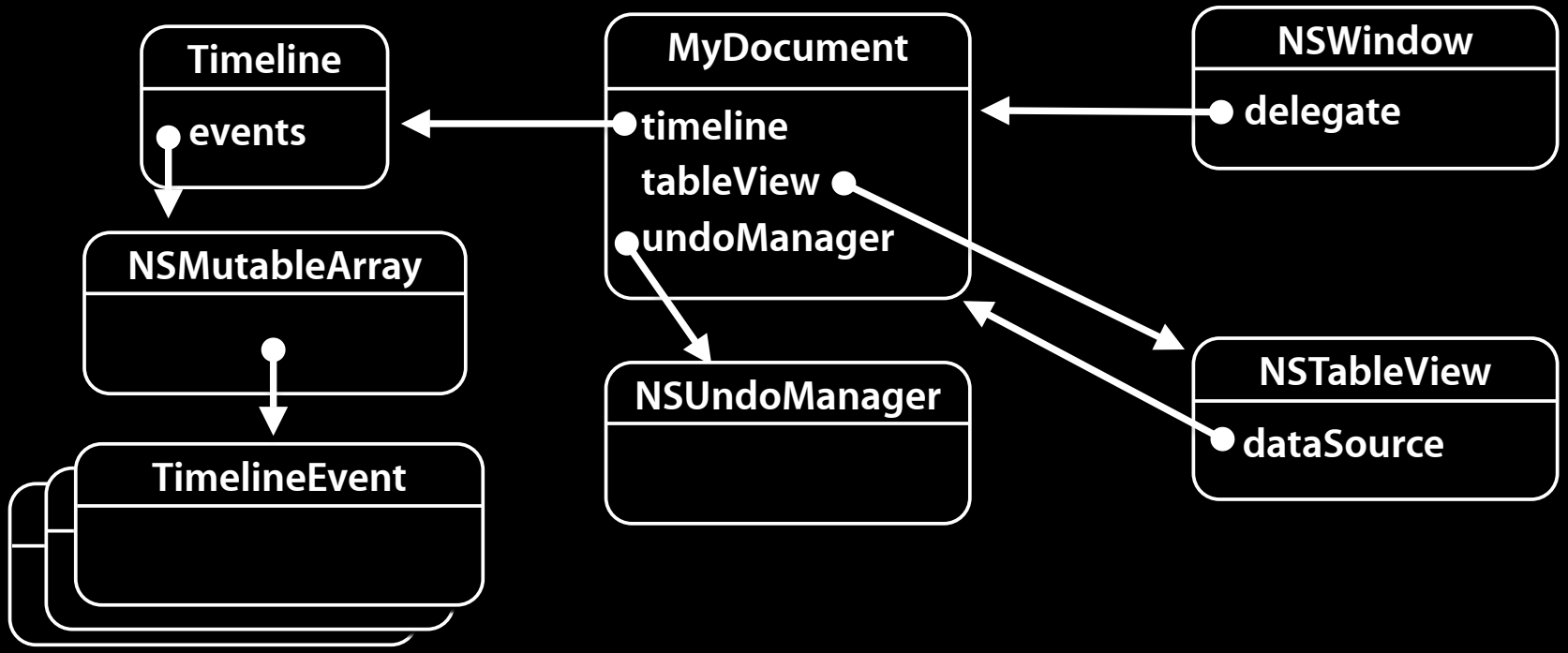
- For example,

```
[undoManager setActionName:@"Add Event"];
```

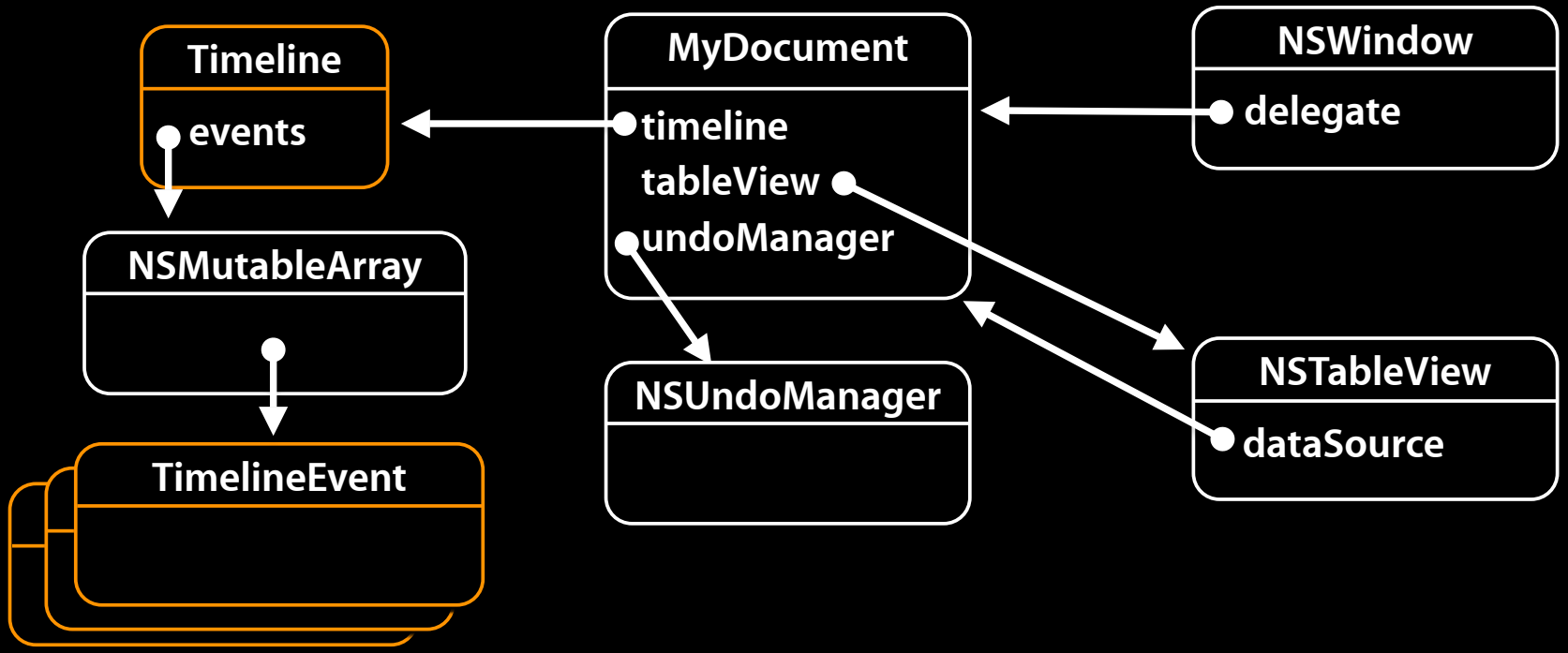
- For opposite actions like add/remove, you probably need to use either the `-isUndoing` or `-isRedoing` method to decide which string is appropriate



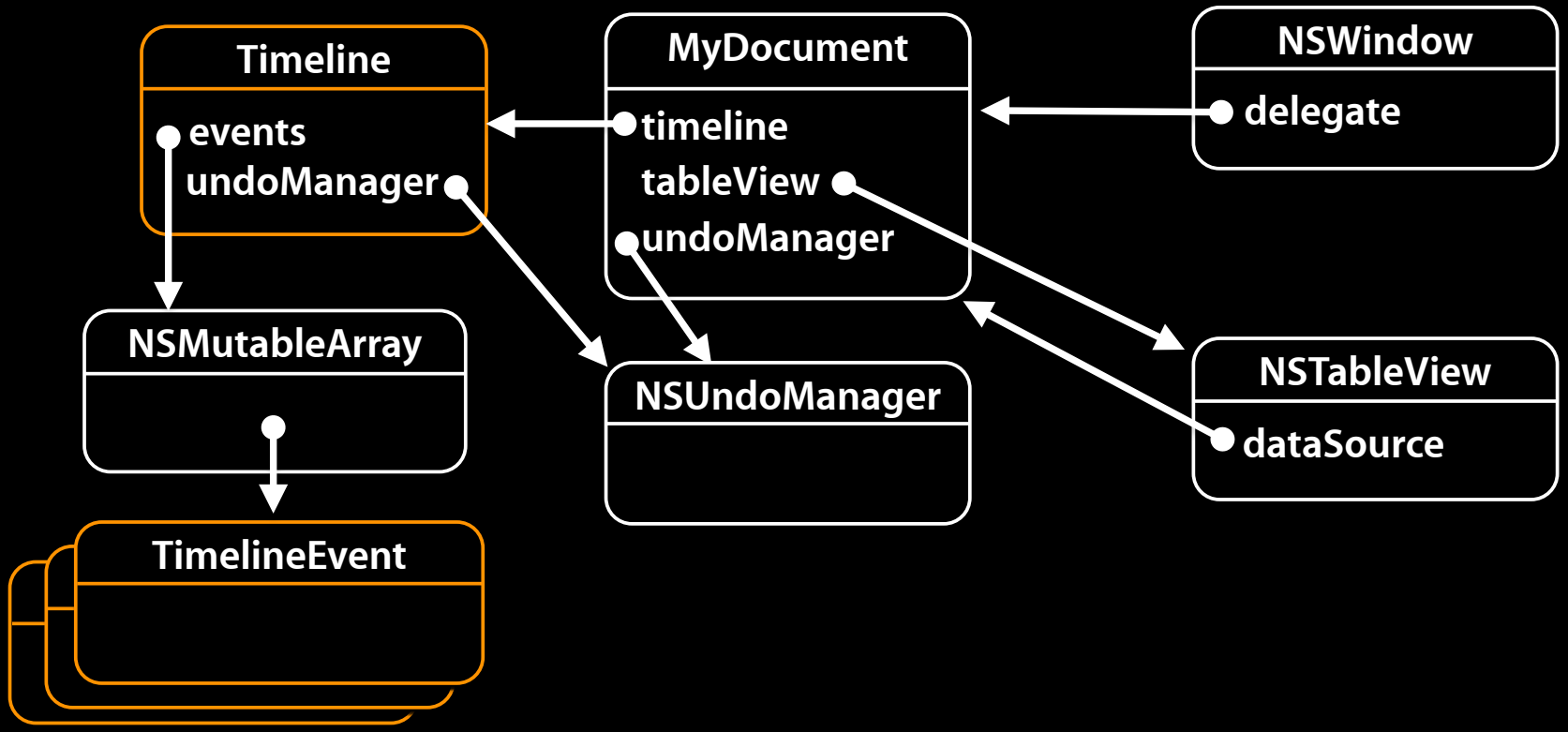
Where to put the undo code?



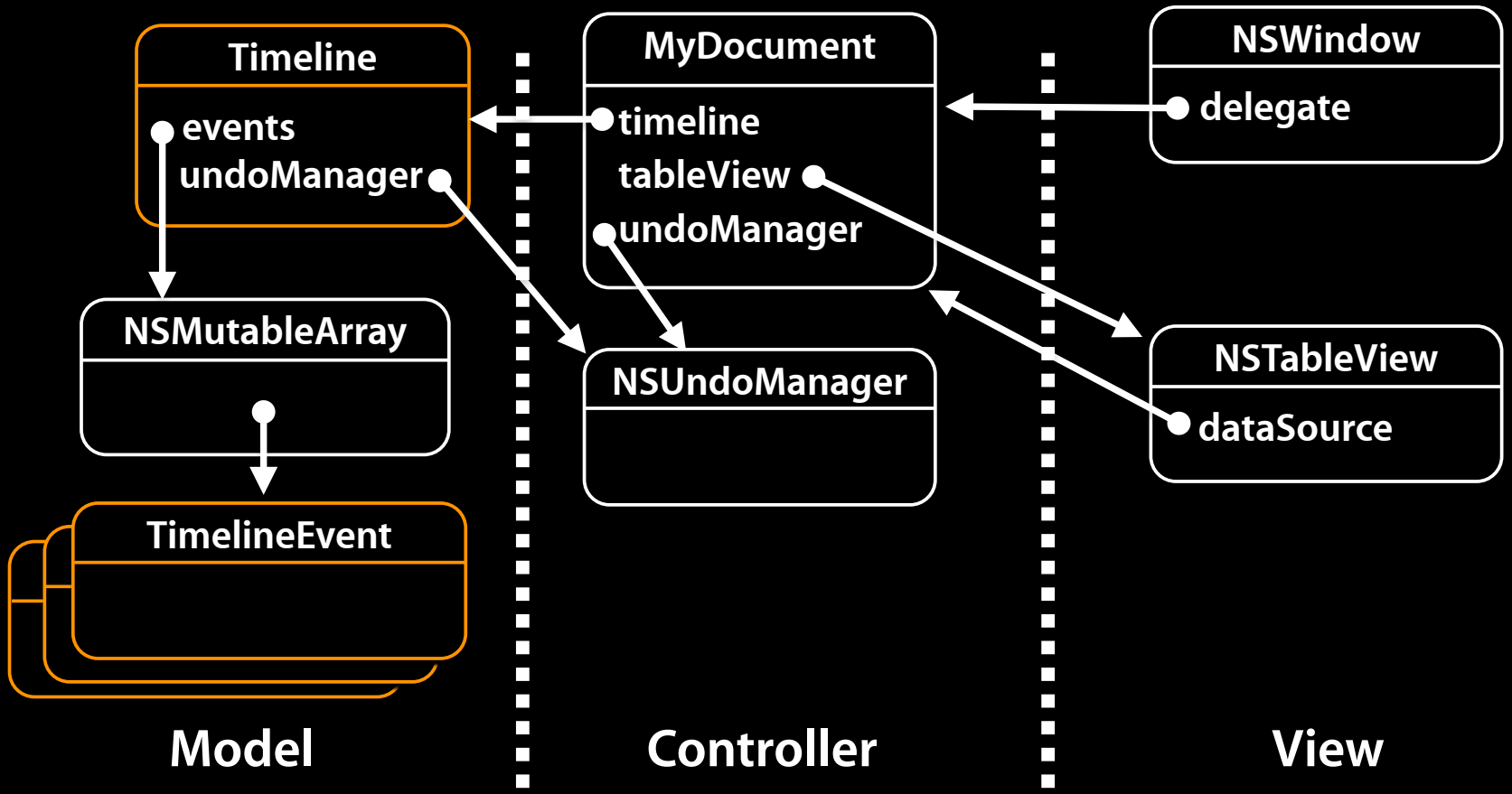
Where to put the undo code?



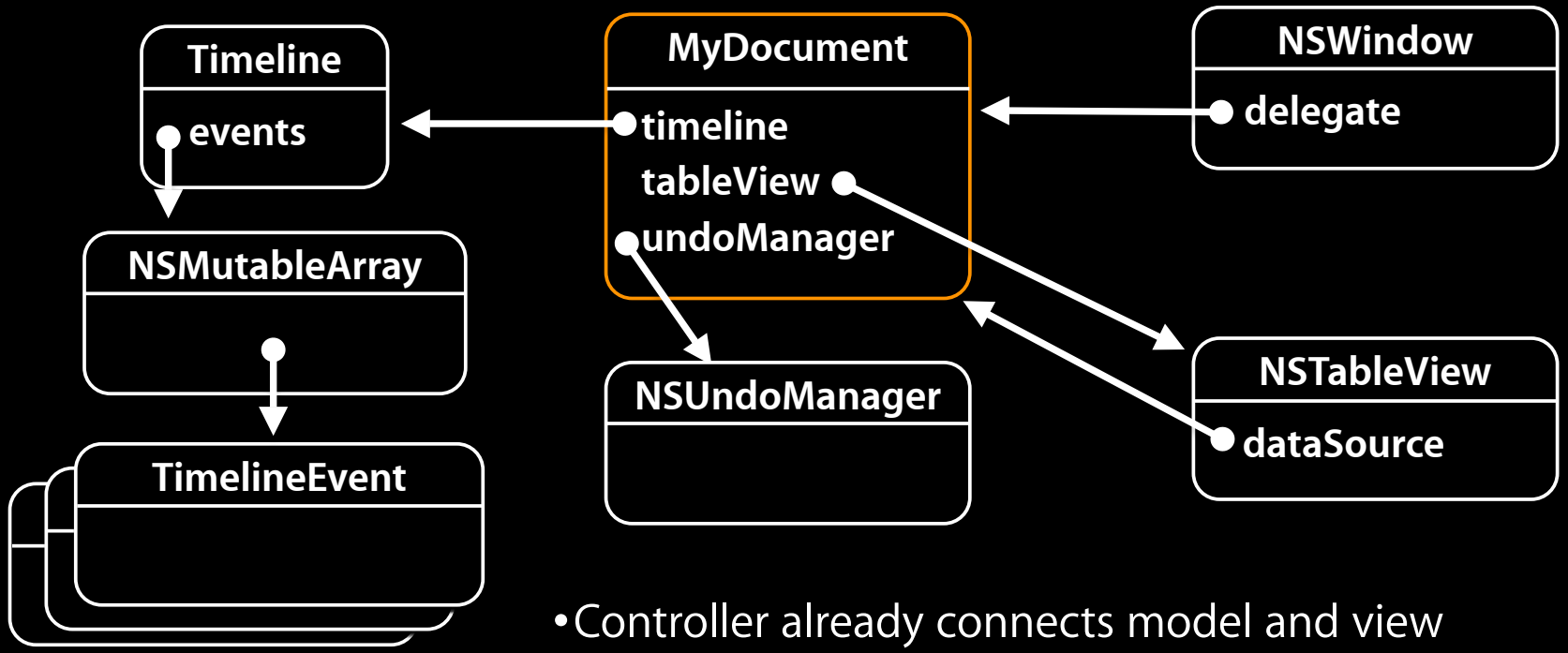
Where to put the undo code?



Where to put the undo code?



Where to put the undo code?



- Controller already connects model and view
- Formalize that connection into funnel points
- All model access goes through the controller
- Funnel point can propagate changes to view

Undo Groups

- One undo operation may be made up of many small undo operations
- Every item on the undo stack is actually a group, containing one or more operations
- Create your own with

```
- (void)beginUndoGrouping;
- (void)endUndoGrouping;
```

Undo and the Event Loop

- By default, all undo operations for one trip through the event loop are put into a single group
- At the beginning of event loop, a group is created
- All undo operations go into that group
- At the end of event loop, group is closed and pushed onto the undo stack

Redo

- During an undo, new operations automatically go onto the redo stack
- Redo stack cleared any time a new undo operation is recorded

Retain Semantics

- NSUndoManager *does not* retain targets
- It *does* retain
  - the "object:" argument
  - any object arguments in the invocation
- If an object is going away, make sure it's not the target of any undo operations

```
[undoManager removeAllActionsWithTarget:object]
```

NSError Basics

NSError

- Object that contains internal and user presentable information about an error.
- Many Cocoa APIs now use errors.
- Integrates with responder chain
- Has a recovery object mechanism to attempt to recover from error
- See Error Handling Guide for all details
- http://developer.apple.com/documentation/Cocoa/Conceptual/ErrorHandlingCocoa

Anatomy of an NSError

- Domain
  - Mac OS X defines a few error domains
  - Provides some information where the error originated
  - Guards against error code collisions from different subsystems
  - Define your own for your project — use reverse DNS style string
- Code
  - Numeric error code, handy for programmatic identification
- User Info Dictionary
  - Key-value pairs using predefined keys to store specific info
  - Localized strings for presentation to the user
  - Information regarding possible recovery options
  - Other customizable information

Calling methods with error argument

```
NSError *error = nil; // declare and initialize
NSString *path; // assume this exists

NSData *data =
      [NSData dataWithContentsOfFile:path
                             options:NULL error:&error];

// Check for nil or NO first
if (data == nil) {
   [self presentError:error];
}
```

Writing methods with error argument

```
- (BOOL)processData:(NSData *) error:(NSError **)error {
    BOOL success = YES;
    // Do your regular stuff, but if error occurs

    // Check that error is not NULL
    if (!success && error) {
      NSDictionary *userInfoDict; // create and populate
      *error = [NSError errorWithDomain:MyAppDomain
                                   code:MyCode
                               userInfo:userInfoDict];
    }
    return success;
}
```

Basic error handling in document

Newer method templates added in Leopard

```
- (NSData *)dataOfType:(NSString *)typeName
                                  error:(NSError **)outError {

    if (outError != NULL) {
        *outError =
          [NSError errorWithDomain:NSOSStatusErrorDomain
                              code:unimpErr
                          userInfo:NULL];
    }

    return nil
}
```

Basic error handling in document

Newer method templates added in Leopard

```
- (BOOL)readFromData:(NSData)data ofType:(NSString *)typeName
                                    error:(NSError **)outError {

    if (outError != NULL) {
        *outError =
          [NSError errorWithDomain:NSOSStatusErrorDomain
                              code:unimpErr
                          userInfo:NULL];
    }


    return YES;
}
```

Errors and exceptions in Cocoa

- Exceptions are for programming errors
  - e.g. asking for an out of bounds index from an array
- Errors are for runtime conditions
  - e.g. attempting to write to a file without permission.
- Errors are designed to be communicated to the user as part of the natural flow of an application.

Miscellaneous

KVC and Collections
Setting formatters and cells

Use immutable types for collection API

- - (NSArray *)employees;
- - (void)setEmployees:(NSArray *)value;

- Even if you have a mutable variable within
- Using mutable types in the API implies that you will somehow react to client changes of the mutable collection.

To-many Relationships

- For immutable to-many relationships, can be accessed the same way as attributes:

```
NSArray *shapes = [canvas valueForKey:@"shapes"];
```

- For mutable to-many relationships you have to request them differently:

```
NSMutableArray *shapes;

shapes = [canvas mutableArrayValueForKey:@"shapes"];
[shapes addObject:newShape];
```

- Returns a "proxy" mutable array for an underlying mutable to-many relationship

Additional KVC array accessor methods

- For key ‘employees’

```
-(unsigned) countOfEmployees;
-(id)objectInEmployeesAtIndex:(unsigned)idx;
-(void)insertObject:(id)obj
                 inEmployeesAtIndex:(unsigned)idx;
-(void)removeObjectFromEmployeesAtIndex:
(unsigned)idx;
```

- Additional ‘replace’ method can be implemented if more speed needed

NSDictionary KVC

- NSDictionary has a custom implementation of KVC that attempts to match keys against keys in the dictionary.
- NSMutableDictionary and KVC allow for getting and setting of arbitrary key value pairs.
- Useful for doing rapid prototyping where you don't have to create custom classes or need extra custom logic
- For example, our timeline could probably just be a dictionary with an "events" property

Demo

Setting formatters and cells

Questions?