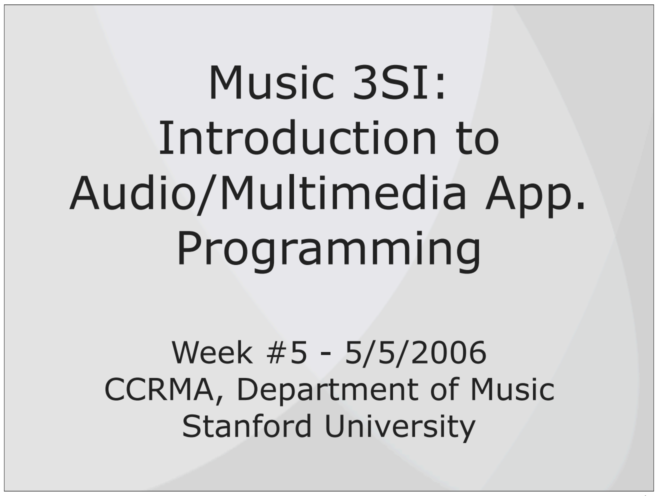
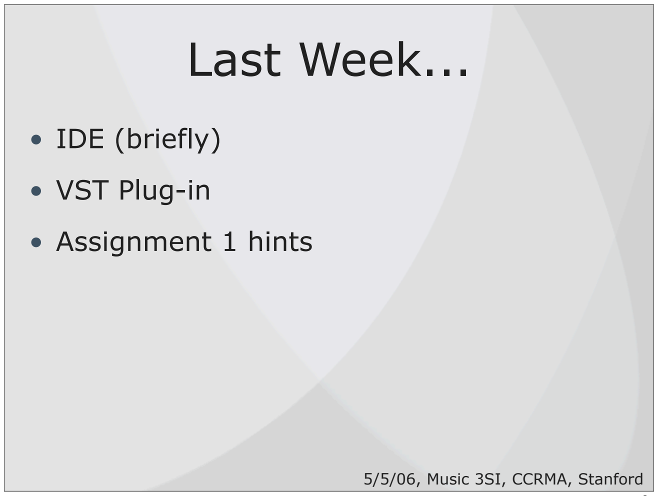


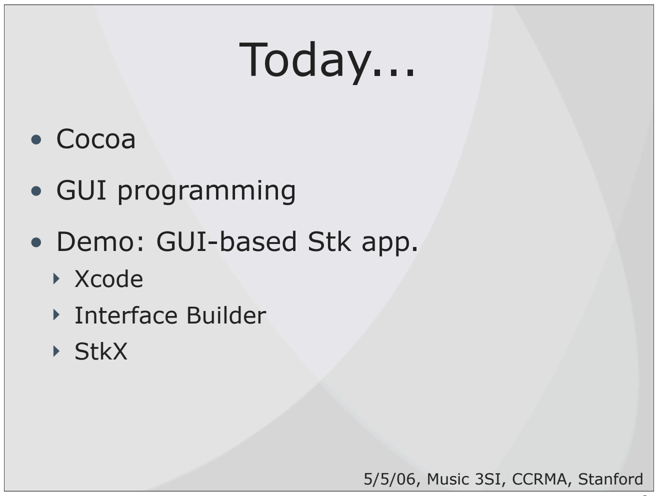
Music 3SI: Introduction to Audio/Multimedia App. Programming

Week #5 - 5/5/2006
CCRMA, Department of Music
Stanford University



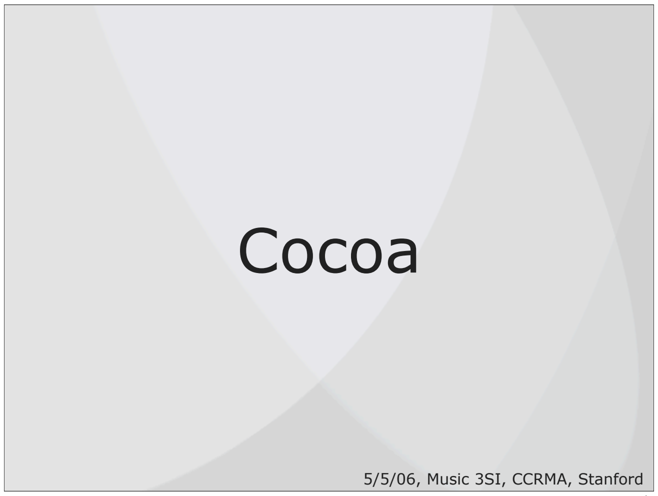
Last Week...

- IDE (briefly)
- VST Plug-in
- Assignment 1 hints

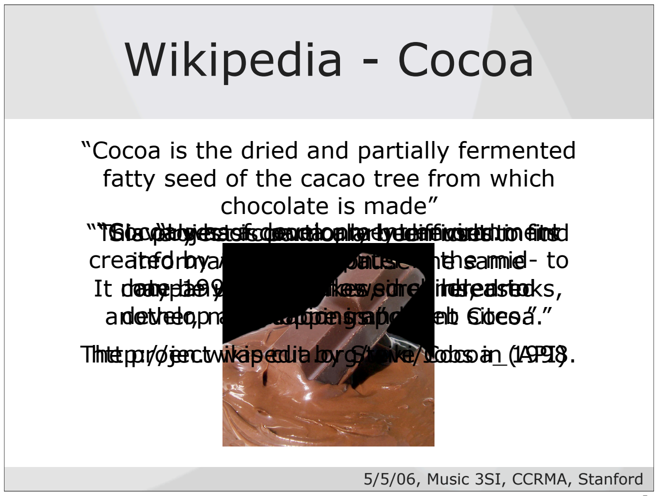


Today...

- Cocoa
- GUI programming
- Demo: GUI-based Stk app.
  - Xcode
  - Interface Builder
  - StkX



Cocoa

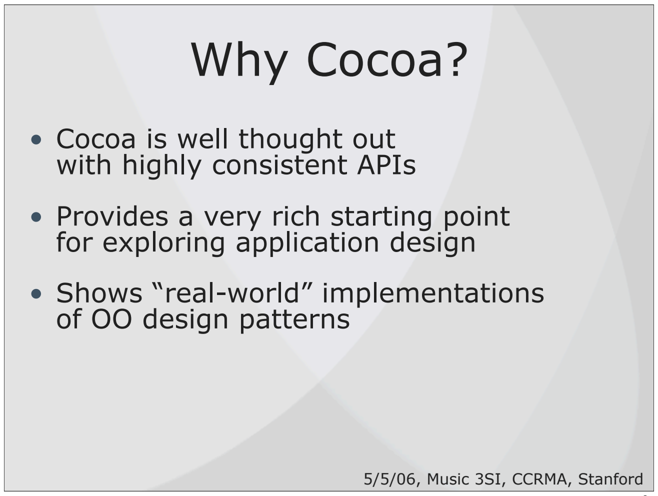


Wikipedia - Cocoa

"Cocoa is the dried and partially fermented fatty seed of the cacao tree from which chocolate is made"

[illegible]

[illegible]



Why Cocoa?

- Cocoa is well thought out with highly consistent APIs
- Provides a very rich starting point for exploring application design
- Shows "real-world" implementations of OO design patterns

Cocoa Applications

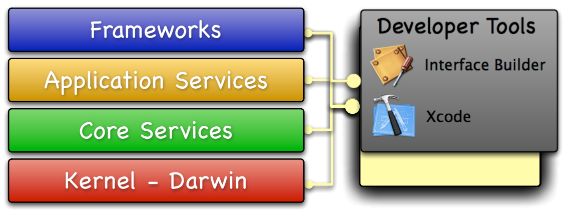
Cocoa Is Many Things

- It's a runtime environment
  - Dynamic dispatch is fundamental
- It's a user interface framework
  - Events, views, buttons, sliders and so on
- It's a development framework
  - A collection of reusable and extendable objects

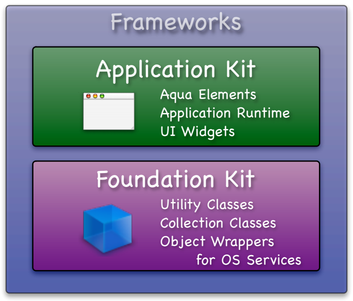
Using Cocoa

- GUI (Graphical User Interface) applications
- Command-line tools
- Plug-ins
- Even device drivers!

Mac OS X Architecture



Cocoa Architecture



Event-Driven Applications

- AppKit manages the flow of events
- Your code is invoked automatically as the user interacts with the application
- You write small chunks of code that handle specific events
- Simple, easy-to-use model

Basic Tools

- Xcode
  - coding
  - app-level specifications
  - building
  - debugging
- Interface Builder
  - user-interface design
  - basic connections between objects

Xcode

- “Wizard” helps you create new projects
  - no *Harry Potter* this
- Best to stick with Xcode-defaults in new projects for now
- Don’t let the complexity overwhelm you

Xcode - A Development

- Edit your code
- Specify how your code is compiled and linked
- Build and run your code
- Debug your code

Interface Builder

- Lays out and connects user-interface elements
  - Target/action
  - Outlets
  - Bindings
- Edits “nib” files
  - A nib file a collection of archived objects (your user interface) stored on disk

OOP

Objects: Evolution of C

- In C, the building blocks are structures and functions
- OOP provides an abstraction over these
- A way of organizing structures and functions into self-contained units
- Lets you group functions with the data they operate on

OOP Vocabulary

- Class:
  - defines the grouping of data and code (“type”)
- Instance:
  - a specific allocation of a class
- Method:
  - a “function” that an object knows how to perform
- Instance Variable:
  - a specific piece of data belonging to an object

Encapsulation

- Keeps implementation details private
- Forces a clearly defined interface to access data or functionality
- Interface is the public “contract” or API
- Implementation can be changed without affecting callers

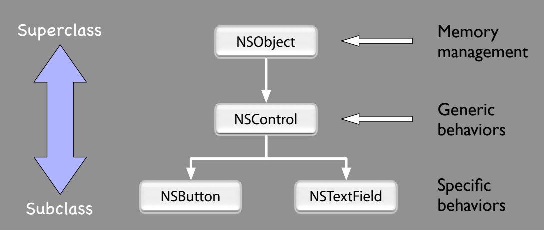
Polymorphism

- Different objects can respond to the same methods in specific ways
- Because data is bound to functionality, methods know what to operate on
- Simplifies interfaces by using consistent terminology

Inheritance

- A class is always derived from a “base” class
- Subclasses can:
  - Add new variables or methods
  - Replace method implementations
  - Refine or extend inherited methods
- Code that is common among objects can be factored to a superclass for reuse

Inheritance



More OOP Info?

- Tons of books and articles on OOP
- Most Java or C++ book have OOP introductions
- ADC document
  - http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC

Objective-C

Objective-C

- A very simple language, but some new syntax
- Strict superset of C
- Single inheritance
  - classes inherit from one and only one superclass
- Dynamic runtime

Why ObjC?

- Exposure to other languages is always good
- A language focused on simplicity and the elegance of OO design
- A data point to compare with designs of C, C++ and Java

Class Interfaces

Instance Variables

```
typedef struct
@interface Person:NSObject
{
    char *name;
    int age;
    float weight;
} Person;

void printName(Person *person);
- (void)printName;
@end
```

Methods

Class Implementations

```
@implementation Person
void printName(Person *person)
- (void)printName
{
    printf (“Name: %s\n”, person->name);
    printf (“Name: %s\n”, name);
}

@end
```

ObjC Files

Person.h

```
@interface Person:NSObject
{
    char *name;
    int age;
    float weight;
}
- (void)printName;
@end
```

Person.m

```
@implementation Person
- (void)printName
{
    printf (stderr,
      "Name: %s\n", name);
}
@end
```

Messaging Syntax

- Calling a method called "doSomething"

C Function: `doSomething(anObject);`

C++ or Java: `anObject.doSomething();`

ObjC: `[anObject doSomething];`

Messaging Syntax

- Calling a method "divide" with arguments

C Function: `divide(arg1, arg2);`

C++ or Java: `obj.divide(arg1, arg2);`

ObjC: `[obj divide:arg1 by:arg2];`

```
- (float)divide:(float)arg1 by:(float)arg2;
```

Selector: `divide:by:`

Types of Methods

- Instance methods operate on a specific object
- Class methods are global and have no specific data associated with them
- "-" denotes instance method
  - `- (void)printName;`
- "+" denotes class method
  - `+ (void)alloc;`

Using Classes

```
#include "Person.h"

main () {
    Person *person;

    person = [[Person alloc] init];
    [person init];
    [person printName];
}
```

"self" and "super"

- Methods have an implicit local variable named "self" (like "this" in C++)

  -
    ```
    - (void)doSomething {
        [self doSomethingElseFirst];
            ...
        }
    ```
- Also have access to "super" methods

  -
    ```
    - (void)doSomething {
        [super doSomething];
            ...
        }
    ```

String Constants

- In C constant strings are
  - `"simple"`
- In ObjC, constant strings are
  - `@"just as simple"`
- Constant strings are NSString instances

More ObjC Info?

- Cocoa Programming for Mac OS X (Ch. 3)
  - by Aaron Hillegass
- ADC document
  - http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC
- Concepts in Objective C are applicable to any other OOP language

Cocoa Application Design

Basic App Functionality

- Save / Load documents
- Open multiple files simultaneously
  - stagger windows nicely to keep things tidy
  - offer good default document names
- Keep track of changes user has made
  - let them undo and redo changes
  - prompt to save or discard when closing
- Double click on documents in Finder

What Cocoa Gives Us

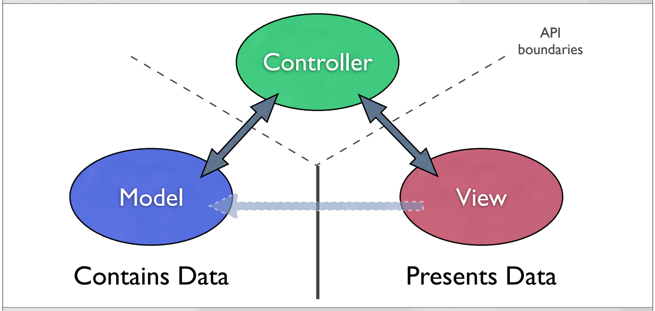
- Look and feel similar to other applications
- Object oriented access to system services
- Lots of building blocks to tinker with
- Strong design paradigms to follow

Model, View, & Controller

- Breaks an application into 3 main categories
  - model:
    manages the app data and state, not concerned with UI or presentation
  - view:
    displays the model objects to the user
  - controller:
    coordinates the model and the view, keeps the view updated when model changes, etc. Typically where app "logic" is.

Model, View, & Controller

- Coordinates between Model & View


Controller
API boundaries
Model
View
Contains Data
Presents Data