Mac OS X Programmierung

Eine Einführung in Cocoa, Objective-C und Xcode

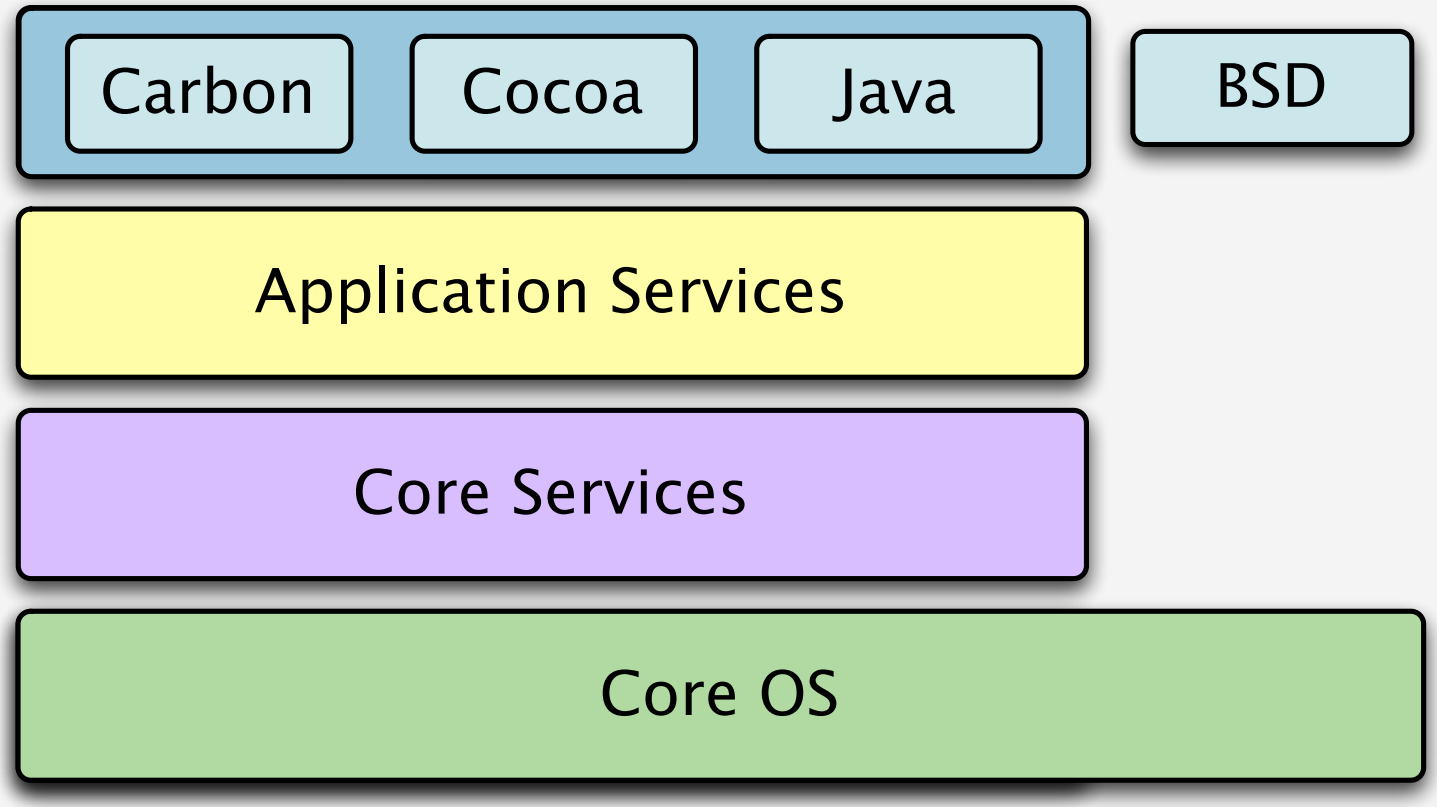
Christina Zeeh
Mac User Group Stuttgart

17.08.2004

Überblick

- Erster Teil
  - Softwareentwicklung für Mac OS X – ein Überblick
  - Objektorientierung in 10 Minuten
  - Cocoa – noch ein Überblick
  - Objective-C für Cocoa
- Pause
- Zweiter Teil
  - Xcode
  - Interface Builder
  - Cocoa Konzepte
- Dritter Teil: Pizza :-)

Mac OS X Architektur



Die Qual der Wahl

- Cocoa
  - Anwendungsentwicklung für Mac OS X
  - Optimale Einbindung in die Mac OS X Umgebung
- Carbon
  - Portierung von Mac OS 9 Anwendungen
  - Portierung von Anwendungen in prozeduralen Programmiersprachen
- Java
  - Plattformunabhängige Anwendungsentwicklung
- BSD, X11, QT, ...

Xcode Tools

- Entwicklertools für Mac OS X
- Neueste Version (1.5) auf http://connect.apple.com (kostenlose ADC Online-Mitgliedschaft)
- Wichtige Komponenten:
  - Xcode (IDE)
  - Interface Builder (GUI Entwicklung)
  - Compiler (gcc)
  - CHUD Tools (Performance, Debugging)
  - weitere Tools (Hilfe-Index, Icons, Packages, ...)

Objektorientierung

Objekte

- Objekte sind Einheiten aus
  - Daten (Instanzvariablen) und
  - Funktionalität
- Daten und Funktionalität eines Objektes können nur über eine definierte Schnittstelle (Methoden) genutzt werden
- Der Nutzer eines Objekts muss nur die Schnittstelle, nicht die Implementierung kennen
- Instanzvariablen können Zeiger auf andere Objekte sein
- Objekte kommunizieren untereinander über Botschaften (Methodenaufrufe)

Klassen

- Eine Klasse ist ein Bauplan für Objekte gleicher Art
  - Liste der Instanzvariablen
  - Methodendefintion (Schnittstelle)
  - Methodenimplementierung (Funktionalität)
- Objekte sind konkrete Instanzen einer Klasse, sie benötigen für die Werte der Instanzvariablen Speicherplatz
- In Objective-C sind Klassen wiederum Objekte, ihre Funktionalität besteht vor allem darin, Instanzen entsprechend ihrem Bauplan erzeugen zu können

Vererbung

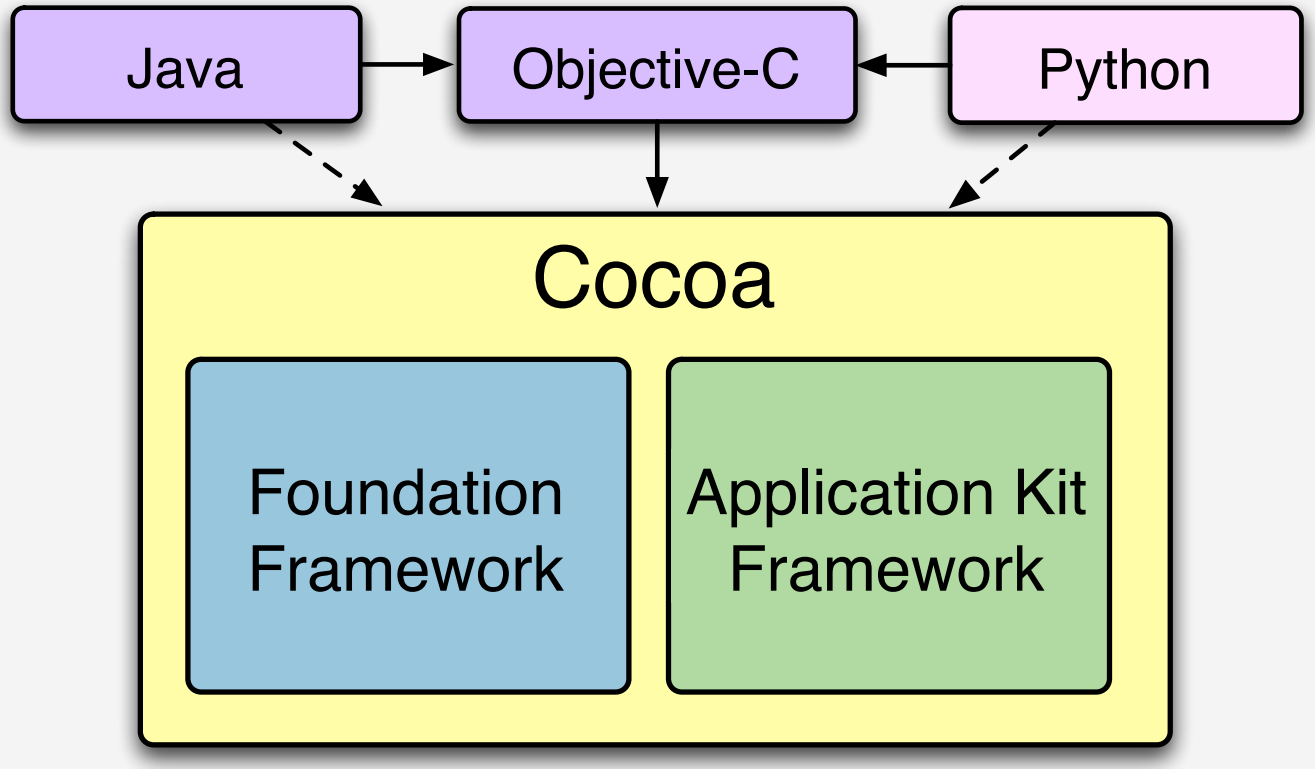
- Erstellung einer neuen Klasse basierend auf einer bestehenden Klasse
  - Instanzvariablen hinzufügen
  - Methoden hinzufügen
  - Methoden verändern (selber Name, andere Implementierung)
- Alle Instanzvariablen und nicht veränderten Methoden werden aus der Oberklasse übernommen (geerbt)
- Polymorphismus erlaubt den Aufruf von Methoden ohne die konkrete Klasse des Empfängers zu kennen

Vorteile

- Natürlichere Abbildung der Realität
- Einfachere Wartung
- Vermeiden von Fehlern
- Einfachere Wiederverwendung von Programmteilen

Cocoa

Cocoa


Java
Objective-C
Python
Cocoa
Foundation Framework
Application Kit Framework

Objective-C
für Cocoa

Objective-C

- Hintergrund: NeXTstep, OpenStep, Mac OS X
- C erweitert um Objektorientierung
  - Definition von Klassen
  - Instanziierung von Objekten dieser Klassen
  - Senden von Botschaften an Objekte
- Syntax der objektorientierten Erweiterungen ähnelt Smalltalk
- Objective-C Schlüsselwörter beginnen mit @

Objective-C Klassen

- Programmcode für eine Klasse besteht aus
  - Schnittstellendefinition: `@interface`
  - Implementierung: `@implementation`
- Üblicherweise auf zwei Dateien aufgeteilt
  - Schnittstellendefinition: `Klassenname.h`
  - Implementierung: `Klassenname.m`

Schnittstellendefinition (.h)

```
#import “Oberklasse.h”

@interface Klasse : Oberklasse
{
      Instanzvariablen
}
+ Klassenmethoden
- Methoden
@end
```

```
#import <Cocoa/Cocoa.h>

@interface Point : NSObject
{
      int x;
      int y;
}
- init;
- initWithX:(int)xValue
          Y:(int)yValue;
- (void)setX:(int)to;
- (void)setY:(int)to;
- (int)x;
- (int)y;
- (int)distanceTo:(Point *)p;
@end
```

Implementierung (.m)

```
#import “Klasse.h”

@implementation Klasse

+ Klassenmethoden {

}

- Methoden {

}

@end
```

```
#import “Point.h”

@implementation Point

(...)

- (void)setX:(int)to {
    x = to;
}

(...)

- (int)x {
    return x;
}

(...)

@end
```

Variablen

- Variablendeklaration wie in C

```
int x;
int someNumbers[5];
char name[255];
char *description;
```

- Variablen können Zeiger auf Objekte sein

```
NSString *name;
NSArray *members;
Point *aPoint;
```

Methoden und Botschaften

- Methodendefinition

```
- (Rückgabetyp)Methode:(Parametertyp)Parameter;
- (void)setX:(int)to;
- initWithX:(int)xValue Y:(int)yValue;
```

  - Definition von Klassenmethoden analog

- Botschaften (Methodenaufrufe)

```
[Empfängerobjekt Methode:Parameter];
[aPoint setX:10];
```

  - Für Klassenmethoden

```
[Klassenobjekt Methode:Parameter];
[SomeClass aClassMethod:someObject];
```

Initialisierung von Objekten

- Ein Objekt benötigt Speicherplatz
- Klassenmethode `alloc` reserviert den Speicher
- `init...` initialisiert die Instanzvariablen
- Erzeugen einer Instanz `aPoint` der Klasse `Point`:

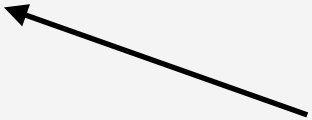
```
aPoint = [[Point alloc] init]; oder
aPoint = [[Point alloc] initWithX:10 Y:15];
```

- Mehrere Initialsierungsmethoden sind möglich ...

  ... **eine** davon ist der **designated initializer**, sie wird von den anderen Initialisierungsmethoden verwendet (i.d.R. diejenige mit den meisten Parametern).

Designated Initializer

```
- initWithX:(int)xValue Y:(int)yValue {
    self = [super init];
    x = xValue;
    y = yValue;
    return self;
}
```



Nur der designated initializer ruft die Initialisierungsmethode von `super` auf!

```
- init {
    return [self initWithX:0 Y:0];
}
```

Abgeleitete Klassen müssen den designated initializer aufrufen, hier also `initWithX:Y:`

```
- initWithX:(int)xValue Y:(int)yValue Z:(int)zValue {
    self = [super initWithX:xValue Y:yValue];
    z = zValue;
    return self;
}
```

Speicherverwaltung

- Der reservierte Speicherplatz muss freigegeben werden, wenn das Objekt nicht mehr benötigt wird:

```
[myObject dealloc];
```

- Aber: In der OO Programmierung wird ein Objekt oft von mehreren anderen Objekten verwendet
- Problem: Woher weiß man, dass das Objekt von niemandem mehr benötigt wird?
- Lösung: Referenzzähler

Referenzzähler

- Mit jedem Objekt ist ein Referenzzähler (**retain count**) verbunden
- Ein mit `[[Class alloc] init]` instanziiertes Objekt hat einen retain count von 1
- `dealloc` wird nicht mehr direkt aufgerufen (!!!)
- Stattdessen: `release` (und das Gegenstück `retain`)
  - `release` vermindert den retain count um 1
  - `retain` erhöht den retain count um 1
  - Ist der retain count 0, wird `dealloc` automatisch aufgerufen

Regeln

1. Wenn man ein Objekt mittels einer Methode mit `alloc` oder `copy` im Namen erhalten hat, muss es mit `release` freigegeben werden, wenn es nicht mehr benötigt wird.
2. Gleiches gilt, wenn man einem Objekt eine `retain`-Botschaft geschickt hat.
3. Hat man ein Objekt auf andere Weise erhalten und will man es nach Verlassen der aktuellen Methode weiterverwenden, muss man eine `retain`-Botschaft schicken.

Autorelease

- Problem: Erzeugung eines temporären Objektes, um es an ein anderes Objekt weiterzureichen:

```
- (NSString *)xAsString {
   NSString *s = [[NSString alloc] initWithFormat:@"%i",x];
   return s;
}
```

- Widerspricht Regel Nr. 1 – unser Objekt ist für `release` verantwortlich!
- Aber: Kein `release` möglich, da `s` deallokiert werden würde, bevor das Empfängerobjekt die Chance zum `retain` hat
- Lösung: `autorelease` statt `release`

Regeln v2.0

1. Wenn man ein Objekt mittels einer Methode mit `alloc` oder `copy` im Namen erhalten hat, muss es mit `release` oder `autorelease` freigegeben werden, wenn es nicht mehr benötigt wird.
2. Gleiches gilt, wenn man einem Objekt eine `retain`-Botschaft geschickt hat.
3. Hat man ein Objekt auf andere Weise erhalten und will man es nach Verlassen der aktuellen Methode weiterverwenden, muss man eine `retain`-Botschaft schicken.

Pause :-)

Xcode

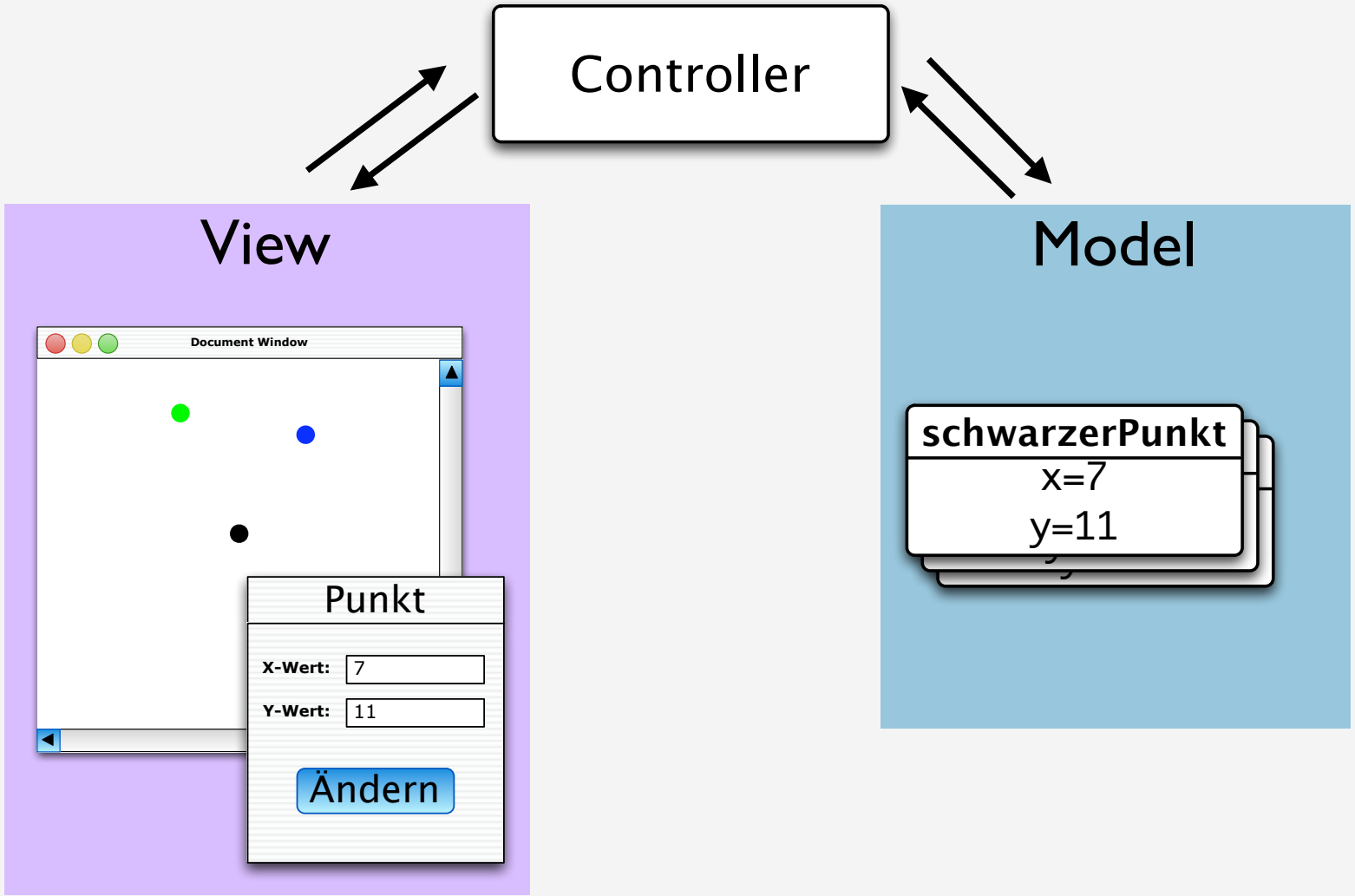
Xcode

- Groups & Files
  - Build Products
  - Targets
- Editor
- Entwicklerdokumentation
- Compiler
  - Run Log
- Debugger

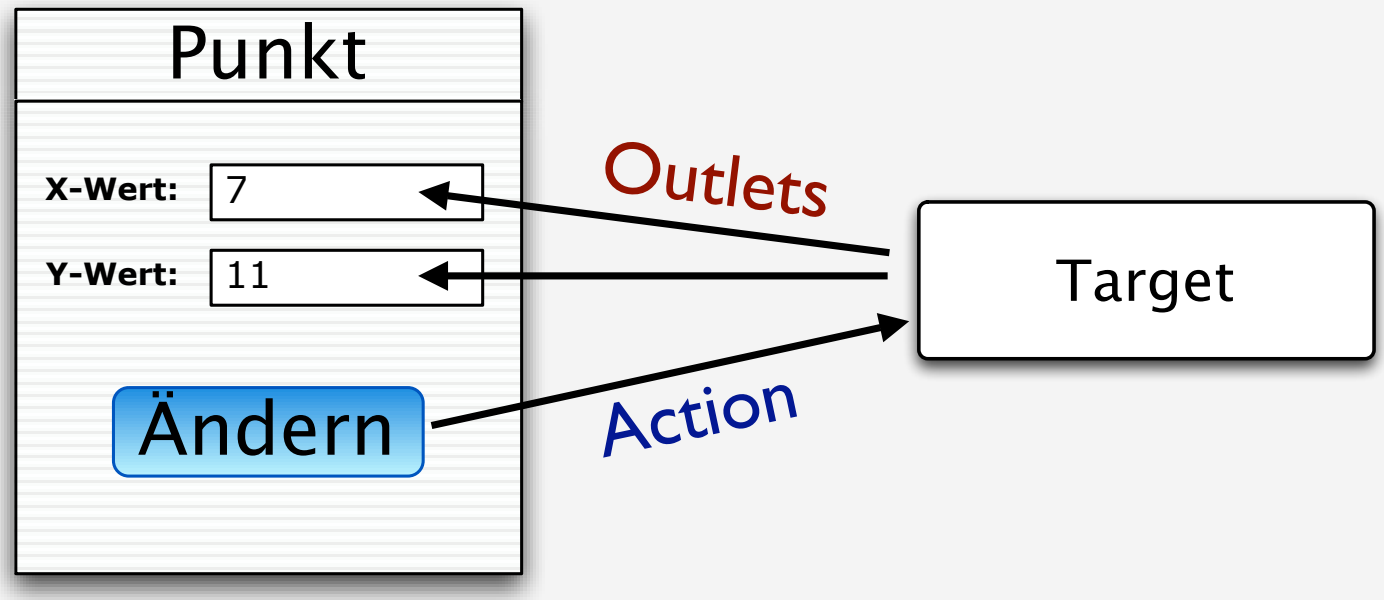
Interface Builder

Cocoa Konzepte

Model-View-Controller



Target-Action und Outlets



Hello, World!

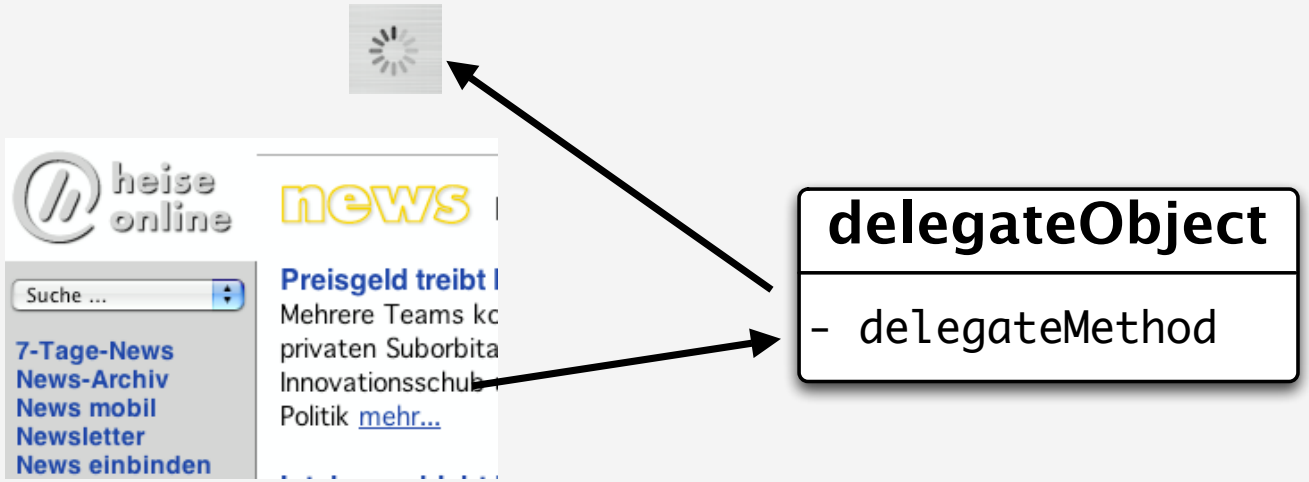
Vorgehensweise

- Neues Projekt “Cocoa Application” anlegen
- MainMenu.nib im Interface Builder öffnen
- Benutzeroberflächen-Objekte einfügen
- MyController Unterklasse von NSObject erstellen
- Outlets und Actions definieren
- Instanz von MyController erzeugen
- Outlets und Actions verbinden
- Quellcode-Dateien für MyController Klasse erstellen
- Code schreiben

Hello, World!
Randomized

Webbrowser

Delegation



- Delegation
  - überlässt die Kontrolle über Teil der Oberfläche
  - informelles Protokoll
- Data Source
  - überlässt Kontrolle über Daten

Noch ein paar Hinweise ...

- Development/Deployment Build Style
- CVS/SVN Integration
- Human Interface Guidelines
- Internationalisierung
- Hilfe
- Preferences
- Document-Based Applications
- Bindings
- Read the source :-)

Weiterführende Informationen

Weiterführende Informationen

- Apple Developer Connection (ADC)
  http://developer.apple.com
- CocoaDev Wiki
  http://cocoadev.com
- StepWise
  http://www.stepwise.com/StartingPoint/Cocoa.html
- OmniGroup (Frameworks, Mailinglisten)
  http://www.omnigroup.com/developer/
- MacDevCenter (o'Reilly)
  http://www.macdevcenter.com
- IRC: #macdev auf irc.freenode.net

Literatur

- **Cocoa Programming for Mac OS X** (2nd Edition)
  Aaron Hillegass
  ISBN 0-321-21314-9
- **Cocoa Programming**
  Scott Anguish, Erik M. Buck, Donald A. Yacktman
  ISBN 0-672-32230-7
- **The C Programming Language**
  Brian W. Kernighan, Dennis M. Ritchie
  ISBN 0-13-110362-8
- **Design Patterns**
  Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
  ISBN 0-201-63361-2

Fragen