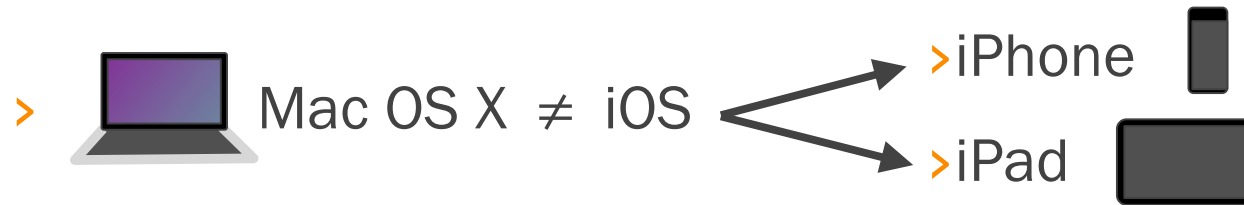




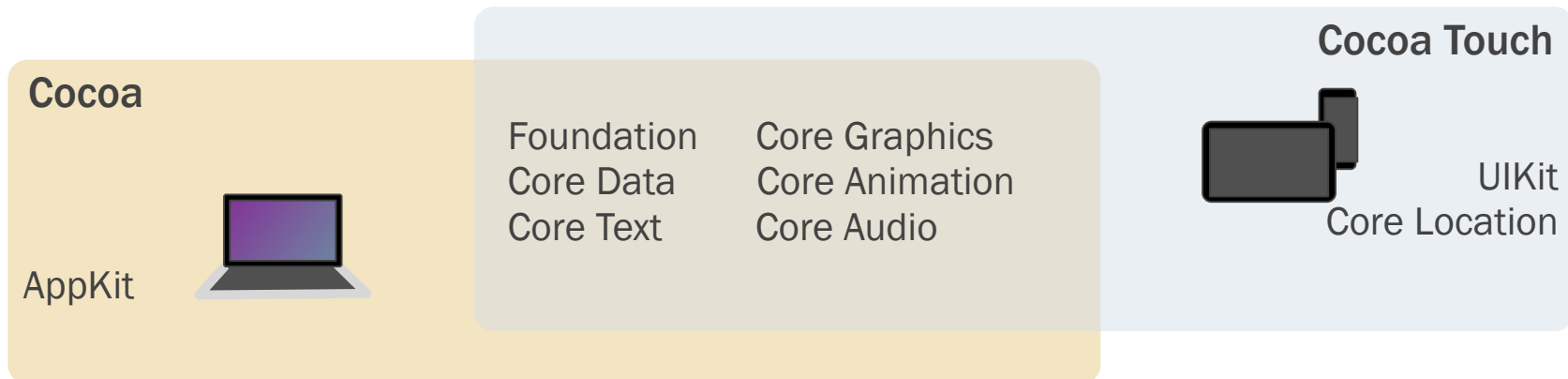
Objective-C für Eclipse-Entwickler

Gegenüberstellungen von Syntax, Konzepten, Patterns.

iOS: Überblick



> Cocoa \neq Cocoa Touch



Objective-C – der »Angstgegner«

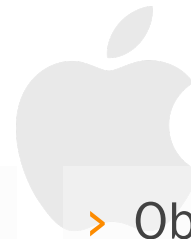
- Objective-C = C + OO
 - Pointer, Headerdateien
 - Klassen, Vererbung, Polymorphie
- Echte **Obermenge** von C
 - > Jeder gültige C-Code ist auch in Objective-C gültig.
 - > Ergänzt C um **Objektorientierung** (+ dynamic typing + reflection).
- Herkunft/Geschichte/Einflüsse
 - > Ein Kind der 80er Jahre (wie ich).
 - > OO-Umsetzung an **Smalltalk** angelehnt.

Entwicklungs-Stacks

The Eclipse logo, a purple sphere with a white ring, is positioned behind the text 'eclipse'.

eclipse

- > Java
- > Equinox, RCP, Riena, RAP, ...
- > Eclipse IDE, WindowBuilder, ...
- > OS X, Windows, Linux ...



- > Objective-C
- > Cocoa / Cocoa Touch
- > Xcode
- > OS X

Methodenaufrufe

eclipse

Java

- > `person.greet();`
- > `Factory.createCar();`
- > `obj.sendMessage ("Hello!",
"Yang", true);`
- > `person.getName();`



Objective-C: wie Smalltalk.

- > `[person greet];`
- > `[Factory newCar];`
- > `[obj sendMessage:@"Hello!"
to:@"Yang"
saveCopy:YES];`
- > `[person name];`
-oder-
`person.name; // dot syntax`

Interface & Impl

eclipse

Eine .java-Datei mit **privaten**
und von außen **sichtbaren** ...

- > Methoden
- > Instanzvariablen
- > statischen Variablen

Header (.h-Datei)

- > Instanzvariablen
- > Sichtbare Methoden und Properties

Implementierung (.m-Datei)

- > Private Methoden und Properties

Interface & Impl

eclipse

Eine Klasse mit **privaten** *und* von außen **sichtbaren** ...

- > Methoden
- > Instanzvariablen
- > Statische Variablen

Instanzvariablen

Sichtbare
Methoden + Properties

Class extension:
Private Methoden

Header (MyClass.h)

Typdeklaration

```
@interface MyClass : NSObject {
    NSArray* protectedIvar;

    @private
    NSUInteger privateIvar;
}

@property int publicProperty;
- (void) publicMethod;
@end
```

Implementierung (MyClass.m)

```
@interface MyClass ()
- (void) privateMethod;
@end

@implementation MyClass
:
@end
```

Property-Veränderungen beobachten

eclipse

JavaBeans + PropertyChangeSupport

```
// MyBean.java
public class MyBean
    implements Serializable {

    private final PropertyChangeSupport pcs
        = new PropertyChangeSupport(this);

    private String title;

    public String getTitle() {
        return this.title;
    }

    public void setTitle(String title) {
        String old = this.title;
        this.title = title;
        this.pcs.firePropertyChange
            ("title", old, title);
    }
    :
}
```

Declared properties + Key-value observing

```
// MyClass.h
@interface MyClass : NSObject
@property (retain) NSString* title;
@end

// MyClass.m
@implementation MyClass
@synthesize title; // getter+setter
@end
```

Sonst nichts.



Property-Veränderungen beobachten



eclipse

JavaBeans + PropertyChangeSupport

```
// MyBean.java
:
```

```

public void addPropertyChangeListener
    (PropertyChangeListener pcl) {
    this.pcs.addPropertyChangeListener(pcl);
}

public void removePropertyChangeListener
    (PropertyChangeListener pcl) {
    this.pcs.removePropertyChangeListener(pcl);
}

```

```
// MyController.java
```

```

...
myBean.addPropertyChangeListener(new
    PropertyChangeListener() {
    public void propertyChanged
        (PropertyChangeEvent e) {
        // do something
    }
});

```

Declared properties + Key-value observing

Kein Glue-Code nötig.

```

// MyController.h
@interface MyController <NSKeyValueObserving>
@end

```

```
// MyController.m
```

```

...
[myClass addObserver:self
    forKeyPath:@"title"
    options:(NSKeyValueObservingOptionNew
        | NSKeyValueObservingOptionOld)
    context:NULL];

```

```

...
- (void) observeValueForKeyPath:(NSString*)path
    ofObject:(id)object
    change:(NSDictionary*)change
    context:(void*)context {
    // do something
}

```

<http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/KeyValueObserving/KeyValueObserving.html>

Getypte Collections

eclipse

Generics

Core



(gibts nicht)

Pattern

```
List<String> myList =
    new ArrayList<String>();

String element = myList.get(0);

System.out.println(element);
```

```
NSArray* myList = [NSArray array];

NSString* element =
    (id)[myList objectAtIndex:0];

NSLog(element);
```

» Wanting this is often a **sign of a weak design**. NSArray is immutable, so it will not "take a pointer to any object" ... you generally expose an NSArray and a couple of methods for modifying that array. «



Memory Management

eclipse

Garbage Collection

... it just works.TM *)

*) ... meistens.

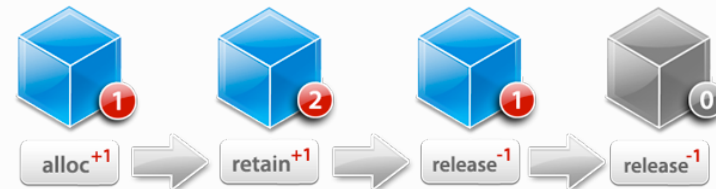
z.B. Lapsed-Listener Problem: <http://www.ibm.com/developerworks/java/library/j-jtp07265/index.html>



Reference counting

(oder Garbage Collection auf OS X)

- Ziele: **Leaks** und **Crashes** vermeiden.
- Wir zählen die Referenzen auf ein Objekt.
- Keine Referenz mehr?
→ Runtime gibt den Speicher frei.



- **+1:** alloc / new* / retain / *copy*
- **-1:** release / autorelease

Bildquelle: http://cocoadevcentral.com/d/learn_objectivec/

<http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/MemoryMgmt/MemoryMgmt.html>



Memory Management



eclipse

Garbage Collection

*... it just works.TM *)*



Managed-memory

```
- (void)didReceiveMemoryWarning {  
    // Releases the view if it  
    // doesn't have a superview.  
    [super didReceiveMemoryWarning];  
  
    // Release any cached data,  
    // images that aren't in use.  
    self.lazyLoadedImage = nil;  
}
```



Event-Benachrichtigungen

Listeners und Delegates



eclipse

Listeners

```
// Controller.java

MyEventListener mel =
    new MyEventListener() {
    public eventHappened(Event e) {
        // do something
    }
};

foo.addEventListener(mel);
```



Delegates

```
// Controller.h

@interface Controller <MyEventDelegate>
:
@end

// Controller.m
@implementation Controller
:
foo.delegate = self;
:
- (void) eventWillHappen:(Event*)e {
    // do something
}

- (void) eventDidHappen:(Event*)e {
    // do something
}
@end
```



Halbe Sachen Halb-implementierte Interfaces



eclipse

Pattern

Core

Adapter classes

```
s.addMouseListener(
    new MouseListener() {
        void mouseDown(MouseEvent e) {}

        void mouseUp(MouseEvent e) {}

        void mouseClicked(MouseEvent e) {
            // do something
        }
    });
```

—oder—

```
s.addMouseListener(
    new MouseAdapter() {
        void mouseClicked(MouseEvent e) {
            // do something
        }
    });
```

Informal protocols

```
@protocol MouseListener
@optional
- (void) mouseDown:(MouseEvent*)e;
- (void) mouseUp:(MouseEvent*)e;
- (void) mouseClicked:(MouseEvent*)e;
@end

- (void)mouseDoubleClick:(MouseEvent*)e {
    // do something
}
```

Mehraufwand für Framework:
Vor Aufruf einer optionalen Methode
immer erst prüfen:

```
if ([delegate respondsToSelector:
    @selector(mouseDown:)]) {
    [delegate mouseDown:event];
}
```



Mit dem “Nichts” arbeiten

Null Object

eclipse

Pattern

null-Checks,
Null-object Pattern
– oder NPEs...

```
if (observer != null
    && !observer
        .shouldDoSomething(this)) {
    return;
}

if (observer != null)
    observer.willDoSomething(this);

this.doSomething();
doSomething(); // alternativ

if (observer != null)
    observer.didDoSomething(this);
```



“NOP”

Core

```
if ([delegate shouldDoSomething:self])
    return;

[delegate willDoSomething:self];

[self doSomething];

[delegate didDoSomething:self];
```





Runtime Magic

eclipse

Lib/FW

ReflectionUtils :-)

org.eclipse.riena.core.util.ReflectionUtils ?
org.eclipse.xtext.util.ReflectionUtil ?
org.eclipse.tptp.platform.analysis.util.java.ReflectionUtils ?
org.eclipse.swordfish.internal.core.util.ReflectionUtil ?
org.eclipse.epsilon.eol.util.ReflectionUtil ?
org.apache.hadoop.util.ReflectionUtils ?
...

```
String controllerName =  
    nameFromXML + "Controller";  
  
Object controller =  
    ReflectionUtils  
        .newInstance(controllerName);  
  
ReflectionUtils.invoke(controller,  
    "calculateResultFor",  
    "Yang");
```



Core

Selector-Erstellung zur Laufzeit

```
NSString* controllerName = [nameFromXML  
    stringByAppendingString:@"Controller"];  
  
Class vcClass = NSClassFromString(  
    controllerName);  
id controller = [vcClass new];  
  
[controller  
    performSelector:@selector(  
        calculateResultFor:)  
    withObject:@"Yang"];
```



Klassen erweitern

eclipse

Notlösungen...

- > Helfer-Klassen mit static-Methoden
 - > StringUtils
 - > ArrayUtils
 - > ...
- > Bytecode-Manipulation
 - > cglib
 - > Javaassist
 - > ASM
- > auf **andere JVM-Sprachen** ausweichen
 - > Scala (Traits, Implicits)
 - > JRuby (Mixins: include/extend)
 - > AspectJ (Inter-type declarations)



Core

Categories

```
// NSString+Capitalization.h
@interface NSString (Capitalization)

- (NSString*) camelcasedString;

@end

// NSString+Capitalization.m
@implementation NSString (Capitalization)

- (NSString*) camelcasedString {
    return [[self capitalizedString]
            stringByReplacingOccurrencesOfString:@" "
            withString:@""];
}

@end
```

Aufruf:

```
[@"Ene mene miste" camelcasedString]
```

Klassenmethoden

eclipse

static methods

```
MySingleton.getInstance();

// normalerweise: new Company();
ReflectionUtils.newInstance
    ("Company.class");

// Vehicle.java
static wheelCount() {
    return Vehicle.wheelsPerAxle()
        * Vehicle.axleCount();
}

System.out.println(
    "Has "+Car.wheelCount()+"wheels");
```



Class methods

```
[MySingleton sharedInstance];

[[Company alloc] init];

// Vehicle.m
+ (int) wheelCount {
    return [self wheelsPerAxle]
        * [self axleCount];
}

NSLog(@"Has %d wheels",
    [Car wheelCount]);
```

Funktional programmieren mit Closures (a.k.a. Lambdas, Blocks)

eclipse

Closures

- > ~~Java 7~~
- > Java 8 ?

- > ... oder auf **andere JVM-Sprachen**
ausweichen:
 - > Scala
 - > JRuby
 - > Clojure

Blocks

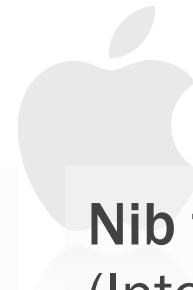
```
// Alle Strings in ALLCAPS verwandeln:  
[myArray map: ^id(NSString* elem) {  
    return [elem uppercaseString];  
}];  
  
// Nur bestimmte Strings auswählen:  
[myArray select: ^BOOL(NSString* elem){  
    return [elem length]>3;  
}];  
  
// Animationen  
[UIView animateWithDuration:0.5  
    animations:^(  
        myView.alpha = 0.0;  
        myView.center=CGPointMake(0.0,0.5);  
    )];
```



GUI Designer

eclipse

WindowBuilder



Nib files (Interface Builder)



The screenshot displays the Eclipse IDE's GUI Designer (WindowBuilder) interface. The main workspace shows a form with fields for "Last Name", "Street", "City", "State", "Zip", "Home", "Mobile", and "Email". The left sidebar contains a "Structure" view showing a project hierarchy, a "Palette" of UI components like "Push Button", "Text", and "Slider", and a "Properties" view for the selected component. The bottom right shows a detailed view of a slider control with its code and a "View" panel for a Label.

```
slider.value = 0
slider.normalizedValue = [ ] ∈ [0.0,1.0]
min=200 max=350
```

```
slider.thumbCenterX
slider.frame.origin.x
+ (slider.normalizedValue *
slider.frame.size.width)
```




Model-View-Controller

Model und View synchronisieren



eclipse

JFace Data Binding

- > `IObservableValue nameWidget = SWTObservables.observeText(name, SWT.FocusOut);`
- > `IObservableValue modelValue = BeansObservables.observeValue(person, "father.firstName");`
- > `bindingContext.bindValue(nameWidget, modelValue, ...);`

Cocoa Bindings (nur OS X)

Deklarativ im Nib (oder programmatisch) den "Key Path" zur Observablen setzen, z.B.

- > `selection.father.firstName`

Model-View-Controller

Verbindung von Controllers und Views

eclipse

FW

Ridgets

am Beispiel von Riena

- > Controllers kennen ihre Views durch Registrierung von **Ridgets**:
 - > `bindToModel(..., <bindingID>)`
... und anschließend ...
 - > `getRidget(<bindingID>)`
- > Events mit Listeners auf den **Ridgets** behandeln:
 - > `myRidget.addListener(... {
 public void callback(...)
});`

Outlets und Actions

Core

- > Controller kennen ihre Views über **Outlets**:
 - > Events als **Actions** verbinden:



Key Takeaways

- Apples Entwicklungswelt ist **anders**.
- Man kann (trotzdem) **gut damit arbeiten**.
- Es gibt **eine IDE, ein App-Framework, eine Sprache, ein OS**.
- Vieles, was in der Java-Welt mit **Patterns** gelöst wird, bietet Objective-C von Haus aus.
- Mit Blocks lässt sich wunderbar **funktional programmieren**.



Danke. Fragen? Gerne!

compeople AG

Untermainanlage 8

60329 Frankfurt/Main

www.compeople.de



Zusammenfassung



	Java/Eclipse	ObjC/Cocoa
Methodensyntax	objekt.methode(arg1, arg2, arg3)	Smalltalk-mäßig
Interface/Impl	Eine Klasse mit public/private-Methoden	Aufteilung in .h- und .m-Dateien
Property-Veränderungen beobachten	JavaBeans + PropertyChangeSupport	Properties + Key-Value-Observing
Getypte Collections	Generics	—
Memory Management	Garbage collection	Reference counting
Event-Benachrichtigungen	Listeners	Delegates
Halbe Sachen (Halb-implementierte Interfaces)	Adapter-Pattern	Informal Protocols
Mit dem “Nichts” arbeiten	Null-checks, Null-object-Pattern – oder NPEs	“NOP”
Runtime Magic	ReflectionUtils :-)	@selector(open:animated:)
Klassen erweitern	—	Categories
Klassenmethoden	static methods	Class methods
Funktional programmieren	Closures??	Blocks
GUI Designer	WindowBuilder	Interface Builder
GUI-Layouting	Layout managers	Pixel-Positionierung + Autoresizing
MVC: Model–View synchronisieren	JFace Data Binding	Cocoa Bindings (nur Mac)
MVC: Verbindung Controller–View	Ridgets + Action Listeners	Outlets + Actions