Xamarin – The First 90 Days

Adventures in iOS App Development

Disclaimer

This presentation is my own opinion and is based on my own experience and may vary substantially from the marketing fluff of the organizations represented.

Who is Ian Johnstone

Working in Software Tech since 1979

Fulfilled all the major roles

- PM, Project Lead, Team Lead, Architect, Analyst, Developer, Tester, Chief Bottle Washer

Prefer technology

Architect, Mentor

Love Object Oriented technologies

- Used Object Pascal (1990) then Delphi
- Made the leap to C# in 2002

Self Employed since 1996

Agenda

Why I wrote the App

Xamarin

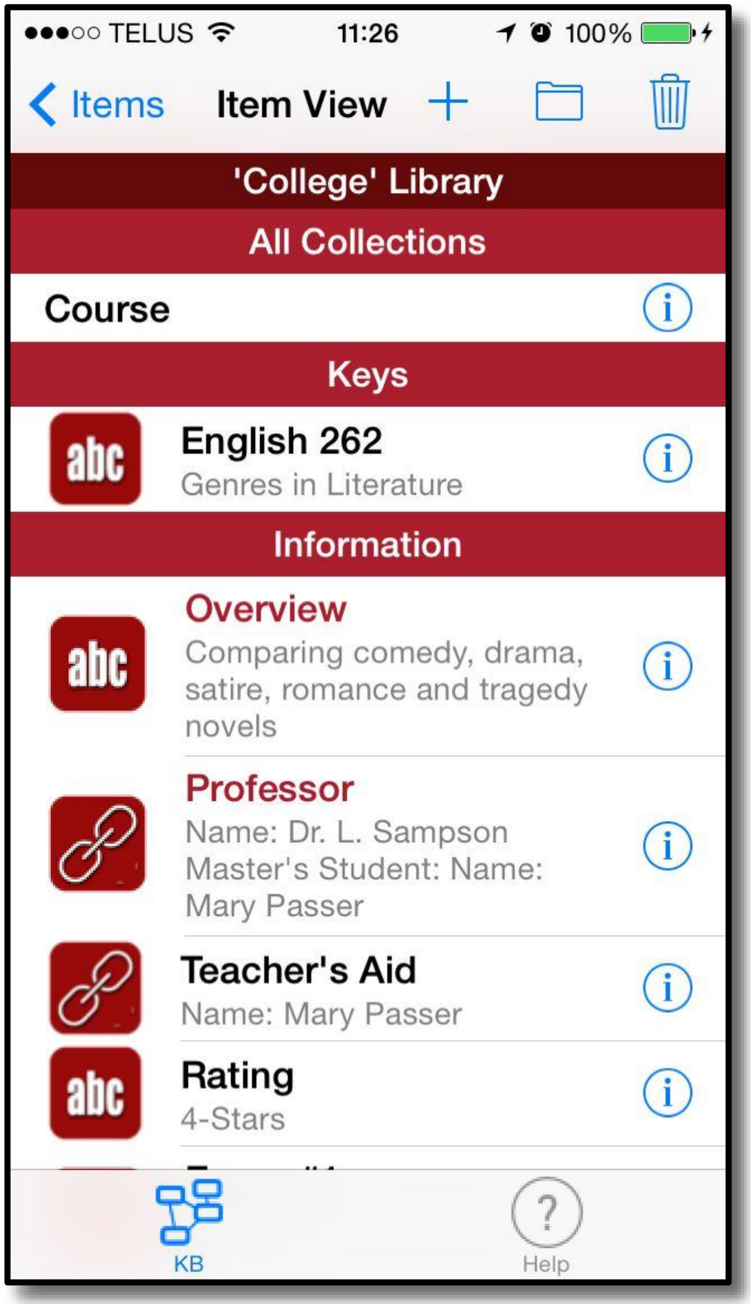
- Languages, O/S
- Platforms
- Likes, Dislikes and Hates

The App

- Architecture and Patterns

The Demo

- The Code Editor
- Demo of the App



'College' Library

All Collections

Course

Keys

English 262
Genres in Literature

Information

Overview
Comparing comedy, drama, satire, romance and tragedy novels

Professor
Name: Dr. L. Sampson
Master's Student: Name: Mary Passer

Teacher's Aid
Name: Mary Passer

Rating
4-Stars

Useful Xamarin Links

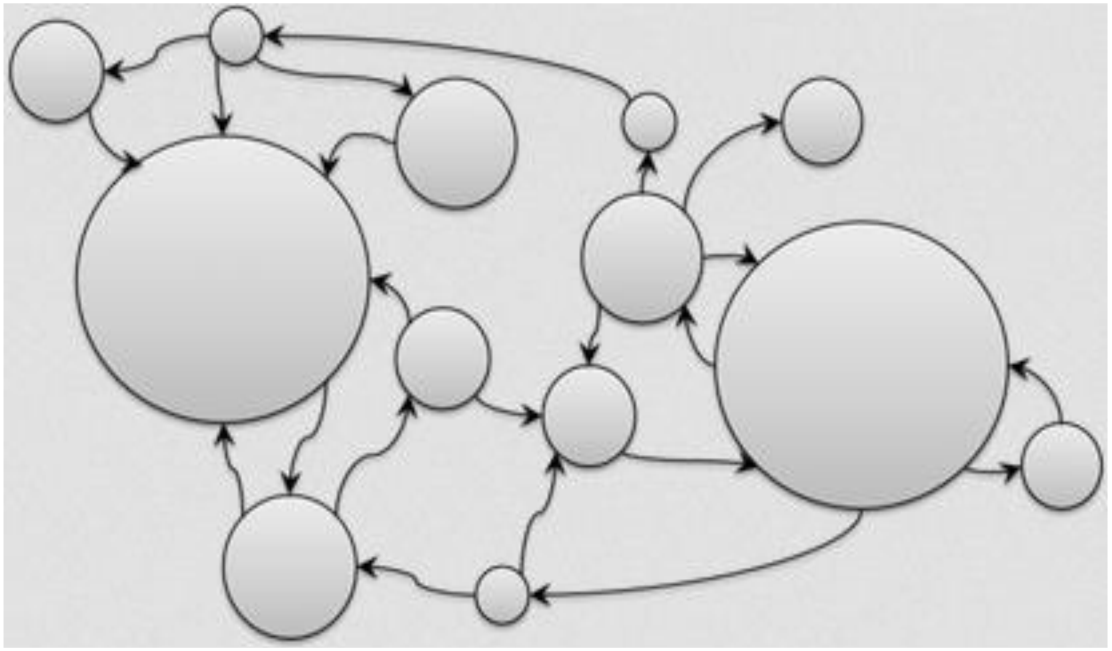
http://github.com/Knowtie/Links

32 Useful Links

The App

KnowTie Knowledge Base, aka, KB

Create Knowledge & Tie it to Anything



Create Knowledge and Tie it to Anything

Knowledge is a set of Items

- Each Item has a unique identifier (a key)
- Any number of fields (14 types are supported)

Items are organized into Collections

KB supports multiple libraries

- Libraries can be shared read-only
- Downloaded to the app (with link or QR Code)

Why?

What compelled me to write this app?

Why did I write the app?

Interested in Knowledge Applications

- Thinking about unstructured data and knowledge typing/acquisition for several years.

Needed an app where it forced me to use all the KEY mobile APIs, to get experience

- Make apps fast that works on all platforms?
  - Can I write code generators, to make it faster?

Mono & Xamarin

10,000 Foot View

What is Mono

Microsoft released C# standard to ECMA

Mono is:

- Open source implementation of Microsoft's .NET Framework based on the C# ECMA standard and the Common Language Runtime
- Currently sponsored by Xamarin.
  - Xamarin uses Mono compilers on Windows and Mac to compile to IL (Intermediate Language) which is transformed to native code on iOS and Android

What/Who is Xamarin?

A company that

- Sponsors Mono – Currently C# 5.0 standard

Created Frameworks to allow C# development, that generates native code

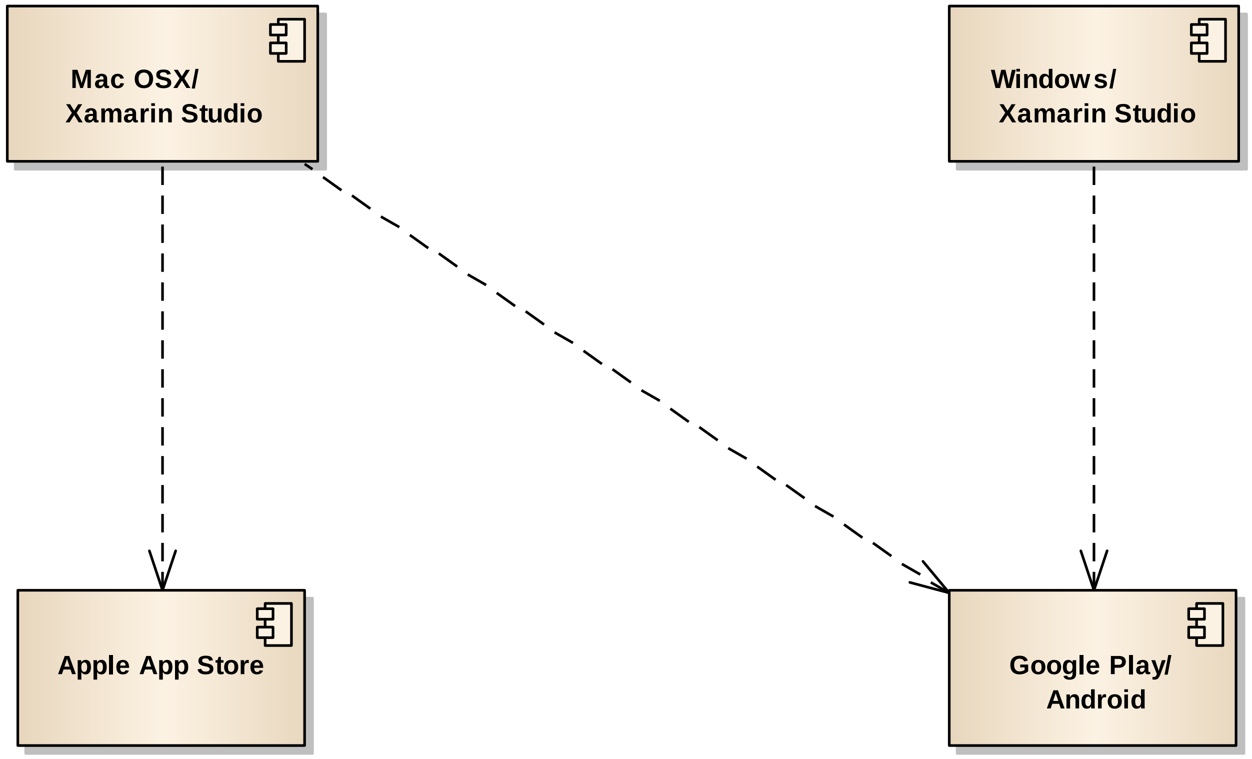
- Created Xamarin.iOS
  - Formerly MonoTouch
- Created Xamarin.Mac
- Created Xamarin.Android
  - Formerly MonoAndroid

High Level Platform Configuration and Pricing

With the Xamarin Platform

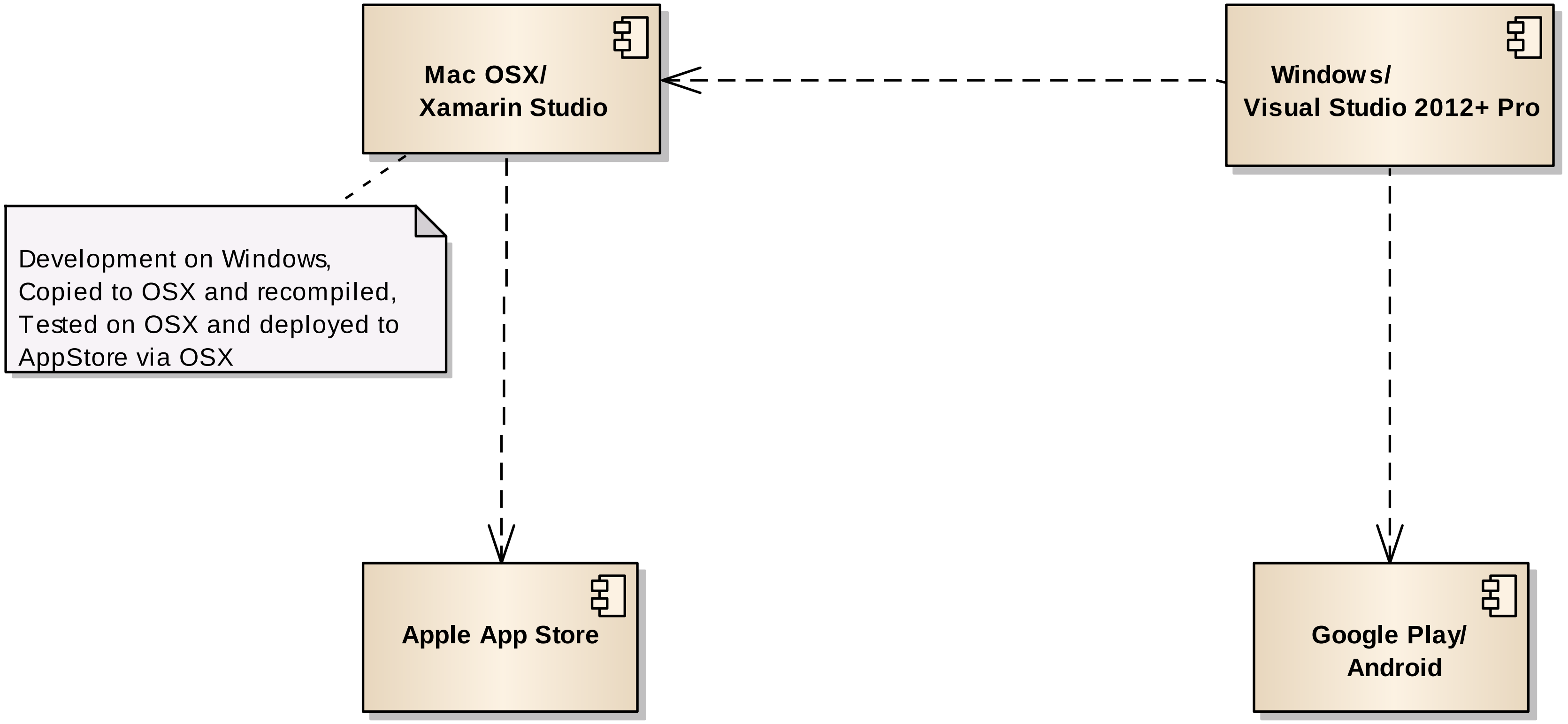
Platform Pricing - Configuration

STARTER/INDIE



Platform Pricing - Configuration

BUSINESS/ENTERPRISE



Development on Windows,
Copied to OSX and recompiled,
Tested on OSX and deployed to
AppStore via OSX

Costs (per Platform per Seat)

Starter

Free

- Deploy to App Stores
- Xamarin Studio
- No 3rd party libraries

Indie ★

$25/mo

- Starter +
- Unlimited App Size
- Xamarin Forms
- 3rd party libraries

Business

$83/mo

- Indie+
- VS2013
- SQLData
- Continuous Integration
- Email Support

Enterprise

$158/mo

- Business +
- Prime Comps
- One Business Day SLA
- Hotfixes
- Acct Mgr.

Apple Deployment Costs

Deploying to Apple

- Requires $99/year to register as developer
- If you own a company, plan for at least 4 weeks to get your paperwork done; Banking, GST, etc.
  - Canadian Buyers get charged 5% GST on your apps.
- Apple takes 30% on your app cost
- Fees are by tier and price varies by App Store

Apple provisioning is extremely complicated

- While not a $$ cost, it’s time consuming

Android Deployment Costs

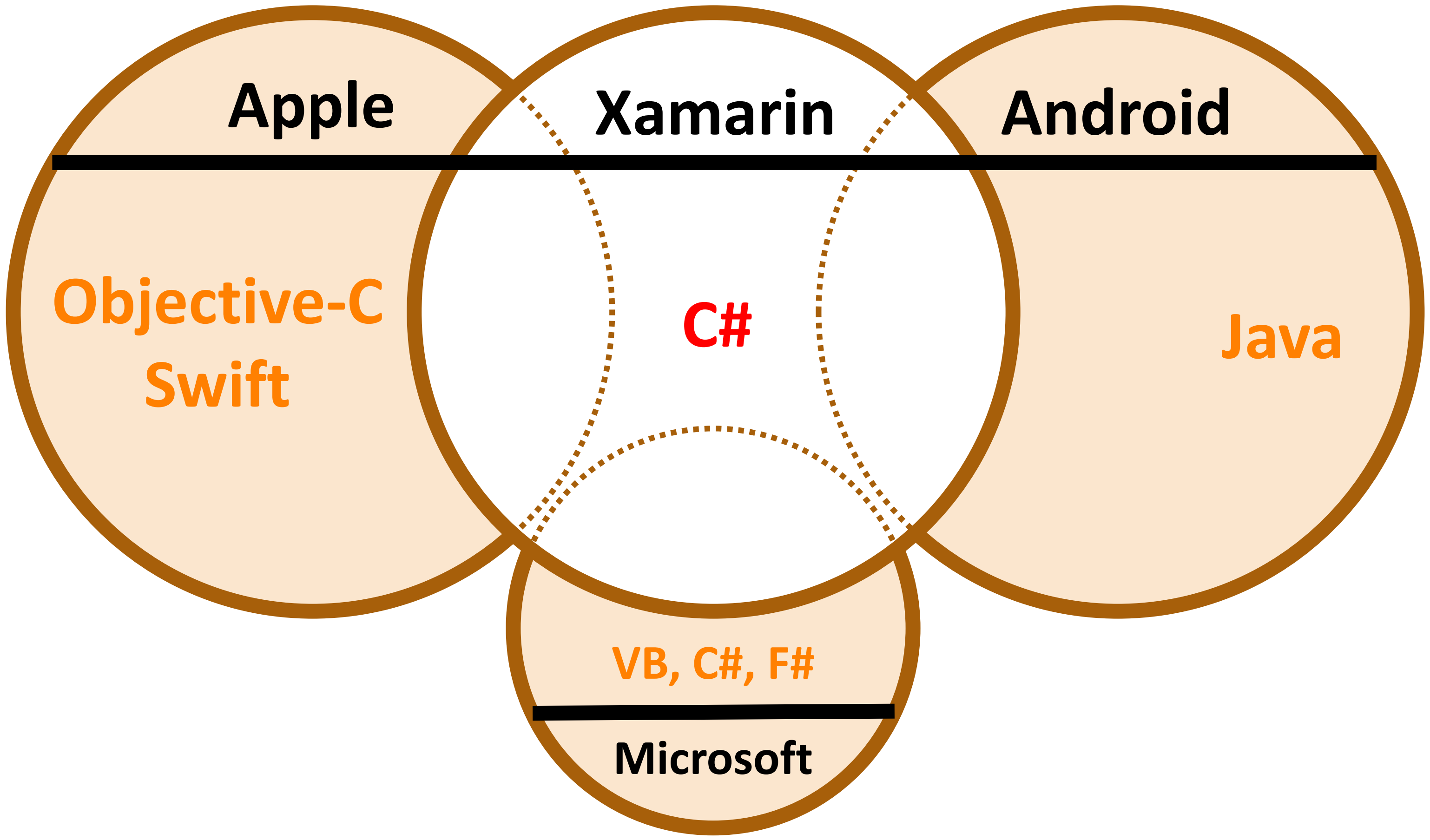
Deploying to Google Play

- Requires $69/year to register as Developer
- Don’t know how long it takes to prep a company to deploy to Google.
- Google takes 30%.
  - Don’t know how pricing model works.

Why Xamarin?

Platforms and Device Distribution

Why Use Xamarin?



Languages

C#, Java, Objective-C and Swift

What is C#

C#

- Language created by Microsoft, with “C”-like syntax
  - To compete with Java
  - Created by Anders Hejlsberg, who also created Turbo Pascal and Delphi
- Major Language Features:

No Pointers	LINQ	var
Garbage Collection	Generics	dynamic
Pure Objects	Lambda Expressions	Optional and Named Parameters
Delegates	Extension Methods	Async/Await
	Anonymous Types	

What is Java

Java

- Language created by Sun and owned by Oracle
  - The Language Standard is used by Google for Android

- Language Features in comparison to C#:

No Pointers	LINQ	var
Garbage Collection	Generics	dynamic
Pure Objects	Lambda Expressions	Optional and Named Parameters
Delegates	Extension Methods	Async/Await
	Anonymous Types	

Source - http://en.wikipedia.org/wiki/Comparison_of_C_Sharp_and_Java

What is Objective-C

Objective-C

- Language used by Apple
  - Has been around since the mid-1980's, and hasn't evolved much

- Language Features in comparison to C#:

No Pointers

Garbage Collection

Pure Objects

Delegates

LINQ

Generics

Lambda Expressions

Extension Methods

Anonymous Types

var

dynamic

Optional and Named Parameters

Async/Await

What is Objective-C

Objective-C

- Language used by Apple

Language Features

- Small Talk extensions to provide a messaging
  - Enable routes dynamically
    - Similar to MQ or SOA but on a smaller internal scale
- Can be object-oriented with the right discipline

New Language Swift

- To start to “catch up” to Java
- Based on my analysis, still far behind C#

What is Swift

Swift

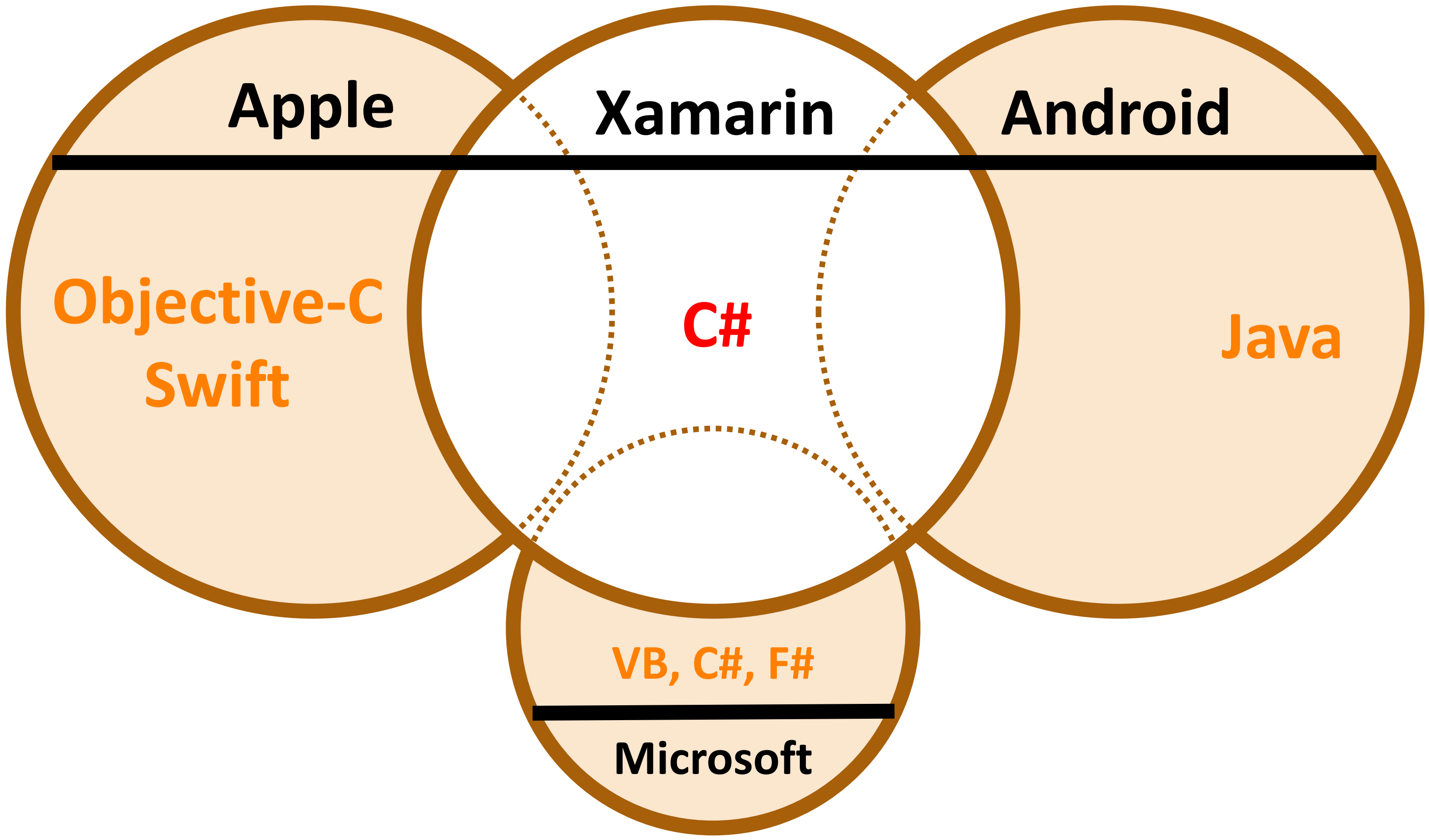
- Language used by Apple
  - To start to “catch up” to Java
  - Based on my analysis, still far behind C#
- Language Features in comparison to C#:

No Pointers	LINQ	var
Garbage Collection	Generics	dynamic
Pure Objects	Lambda Expressions	Optional and Named Parameters
Delegates	Extension Methods	Async/Await
	Anonymous Types	

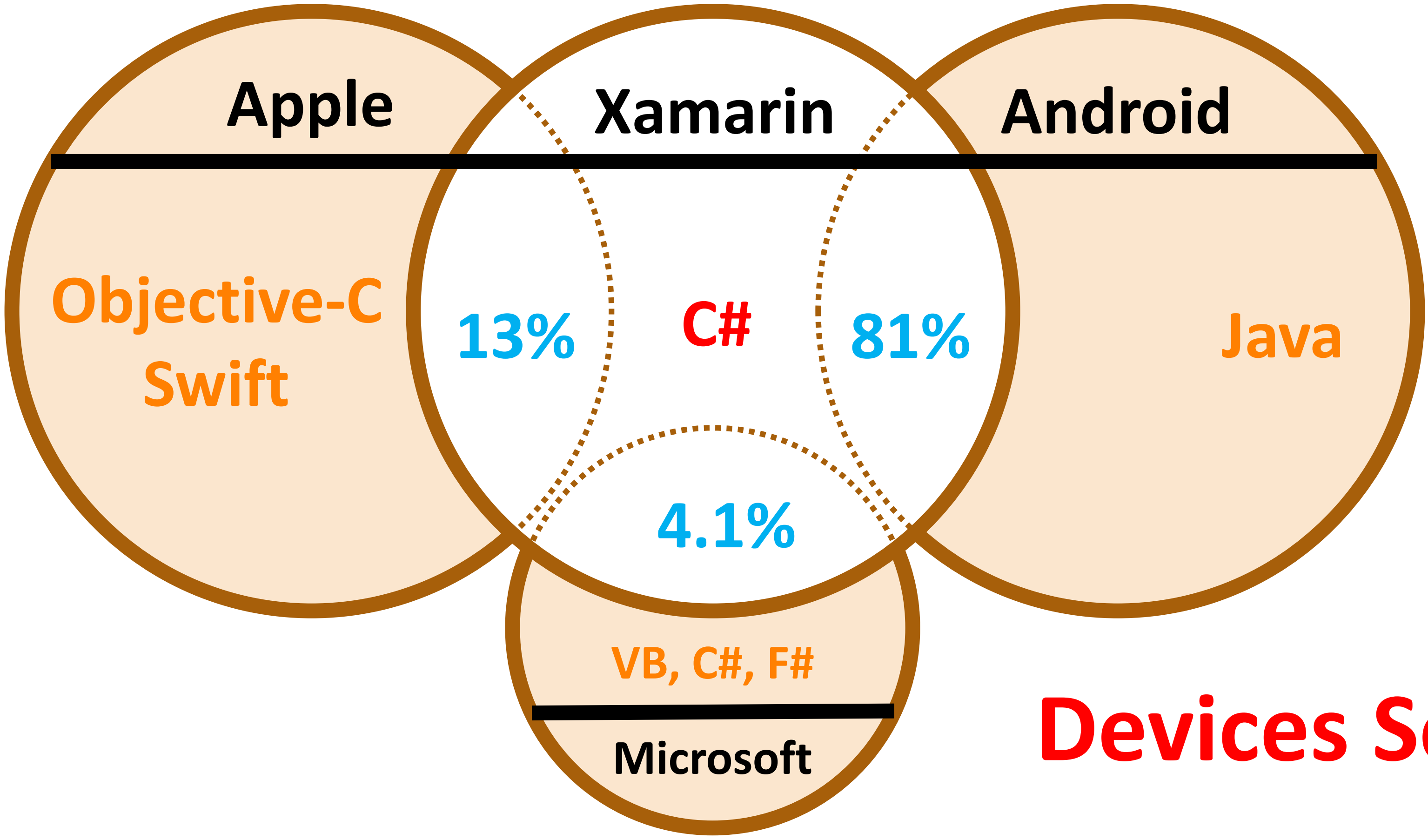
Why Xamarin?

Platforms and Usage

Why Use Xamarin?

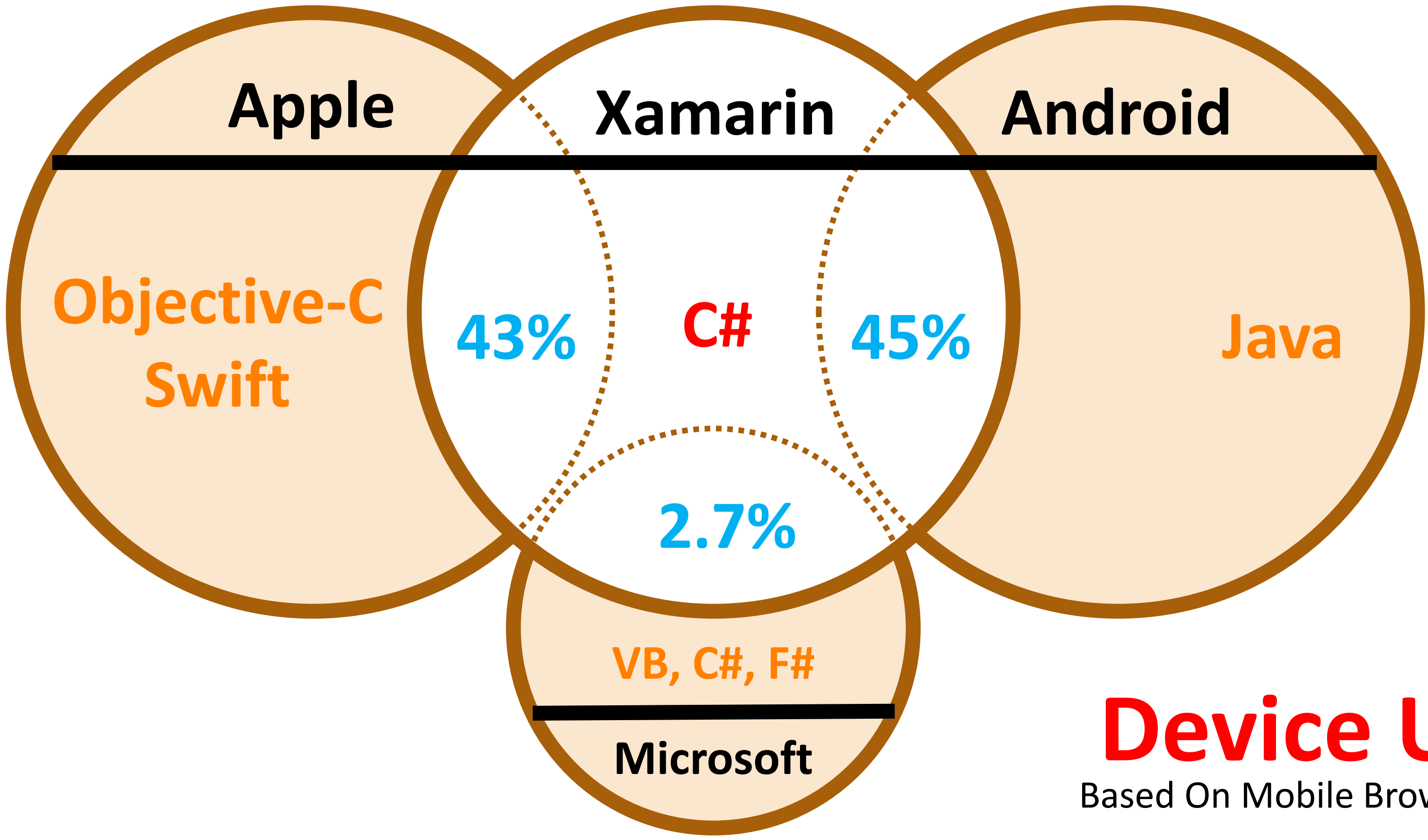


Why Use Xamarin?



Devices Sold

Why Use Xamarin?



Device Use

Based On Mobile Browser Use

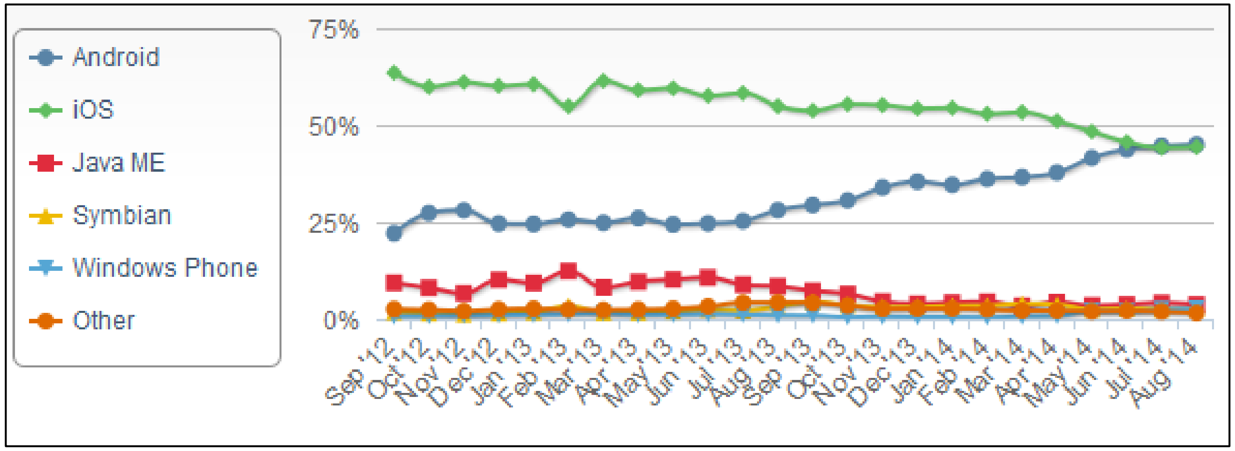
Mobile Devices Counts

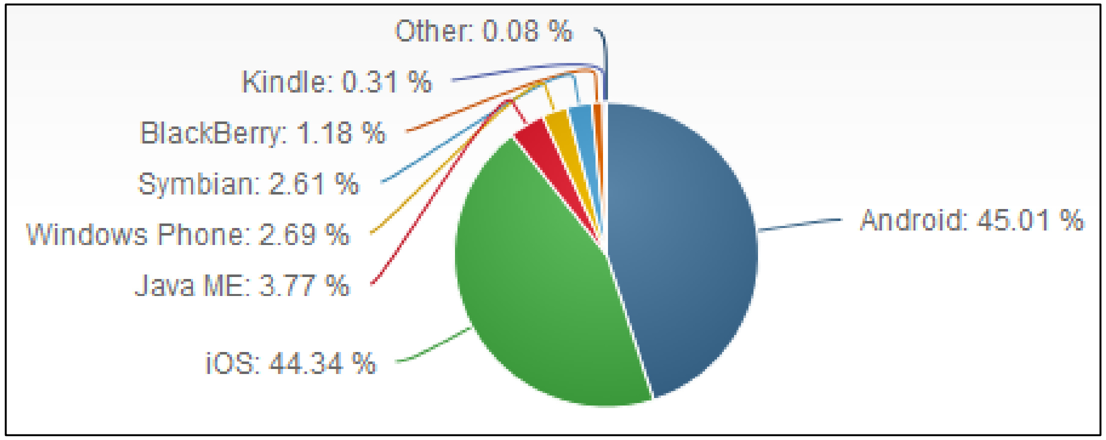
Global Shipments	Q3 ’12	%	Q3 ’13	%	Use (‘14)
Android	129.6	75.0%	204.4	**81.3%**	45%
Apple	26.9	15.6%	33.8	**13.4%**	43%
Microsoft	3.7	2.1%	10.2	**4.1%**	2.7%
BlackBerry	7.4	4.3%	2.5	**1.0%**	
Others	5.2	3.0%	0.5	**0.2%**	
Total (in millions)	**172.8**	**100.0%**	**251.4**	**100.0%**	

Source - http://www.engadget.com/2013/10/31/strategy-analytics-q3-2013-phone-share/

Source - http://www.netmarketshare.com

Mobile Device Use Distribution

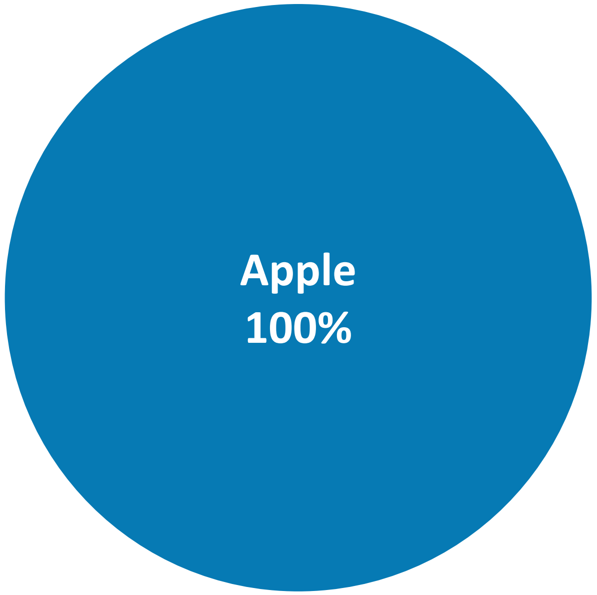


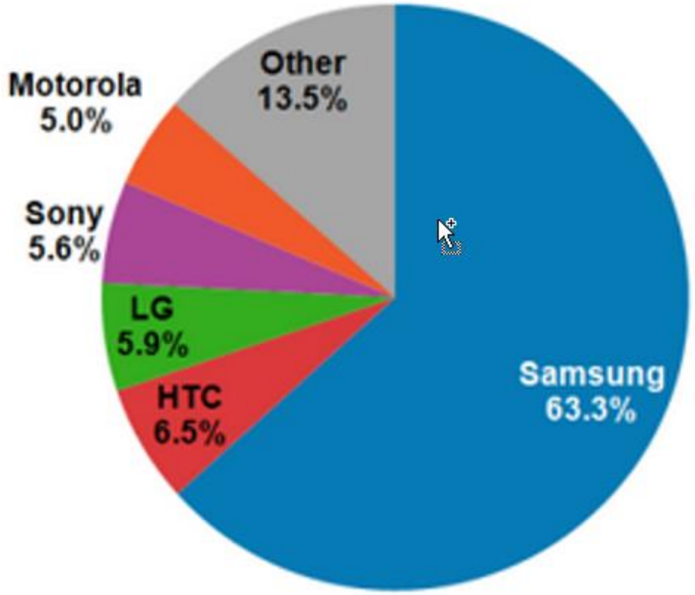


Comparing Apple and Android Hardware & O/S

Device Distribution by Maker

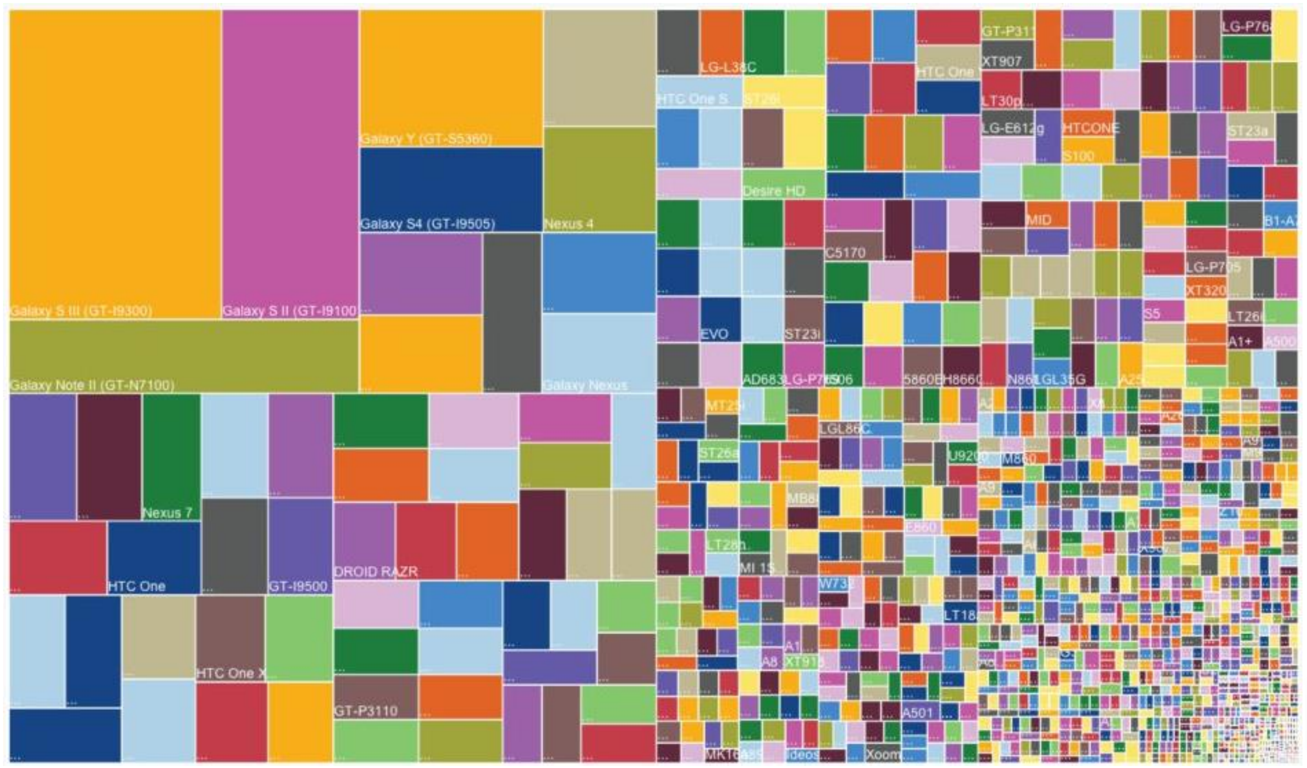
Global iOS Share





Source - http://bgr.com/2013/11/08/samsung-android-devices-market-share/

Number of Android Devices



Only 11,868

http://qz.com/109657/here-are-the-11868-devices-and-counting-that-every-android-app-has-to-work-on/

Number of iOS Devices

iPhone

- 10 Devices
  - Source: http://support.apple.com/kb/HT3939

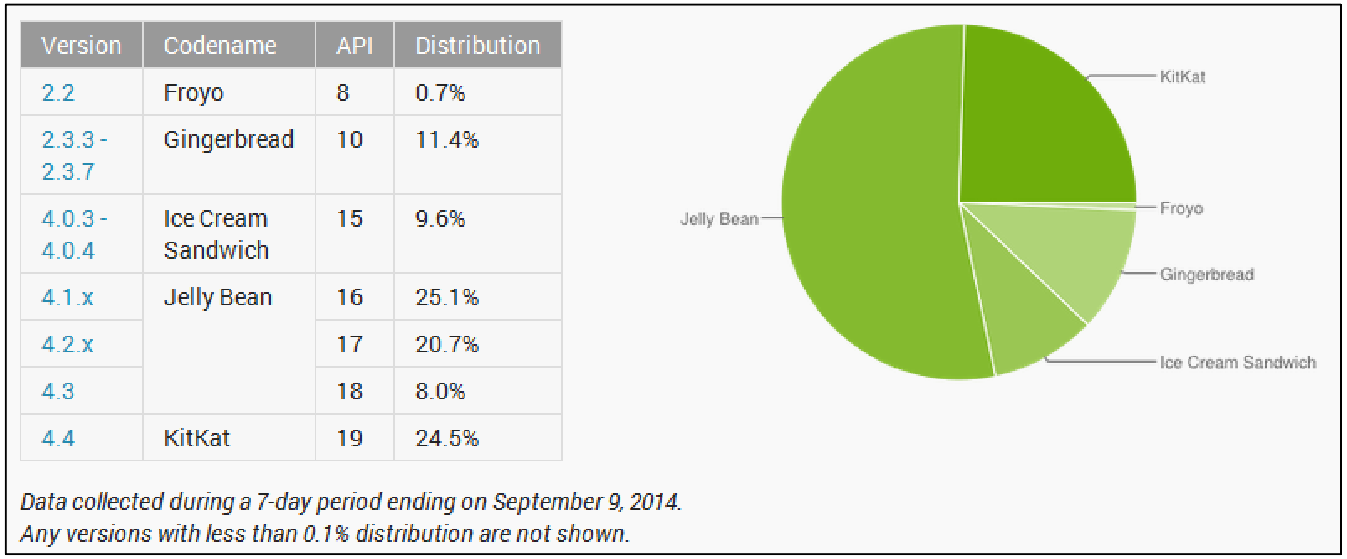
iTouch

- 11 Devices
  - Source: http://support.apple.com/kb/HT1353

iPad

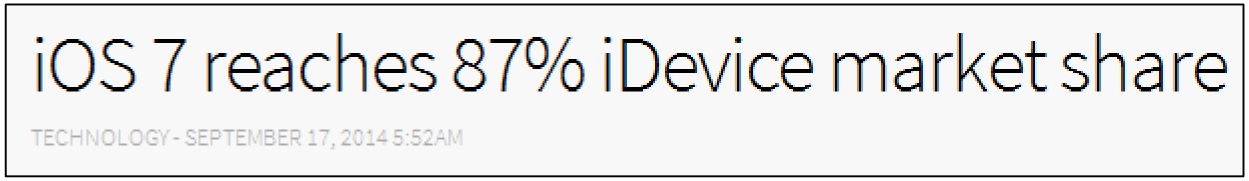
- 20 Devices (Wifi + 3G combined)
  - Source: http://support.apple.com/kb/HT5452

Android OS Distribution



Version	Codename	API	Distribution
2.2	Froyo	8	0.7%
2.3.3 - 2.3.7	Gingerbread	10	11.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	9.6%
4.1.x	Jelly Bean	16	25.1%
4.2.x		17	20.7%
4.3		18	8.0%
4.4	KitKat	19	24.5%

Data collected during a 7-day period ending on September 9, 2014.
Any versions with less than 0.1% distribution are not shown.



iOS 7 reaches 87% iDevice market share

TECHNOLOGY - SEPTEMBER 17, 2014 5:52AM

Source - https://developer.android.com

Source - http://cir.ca/news/apple-releases-ios-7

Device Resolutions

ANDROID

- 2560X1600
  1366X768
- **1920X1200**
  1280X800
  1280X768
  1024X800
  1024X768
  1024X600
- 960640
  960X540
  854X480
  800X600
  800X480
  800X400

APPLE

- **1136 x 640**
- 1334 x 750
- 2208 x 1242
- 1080 x 1920
- 1024 x 768
- **2048 x 1536**

Conclusion

Android

- 12,000 Devices and fractured O/S environment
  - Make development and deployment more difficult.
- Apps compatible with
  - Only KitKat address 25% of the devices
  - KitKat + JellyBean address 75% of devices
    - JellyBean was Released in July 2012

Apple

- 41 Devices with 90% using iOS7
  - iOS 5.x was released May 2012

Mobile Platform and Xamarin Architecture

Mobile Platform Architecture

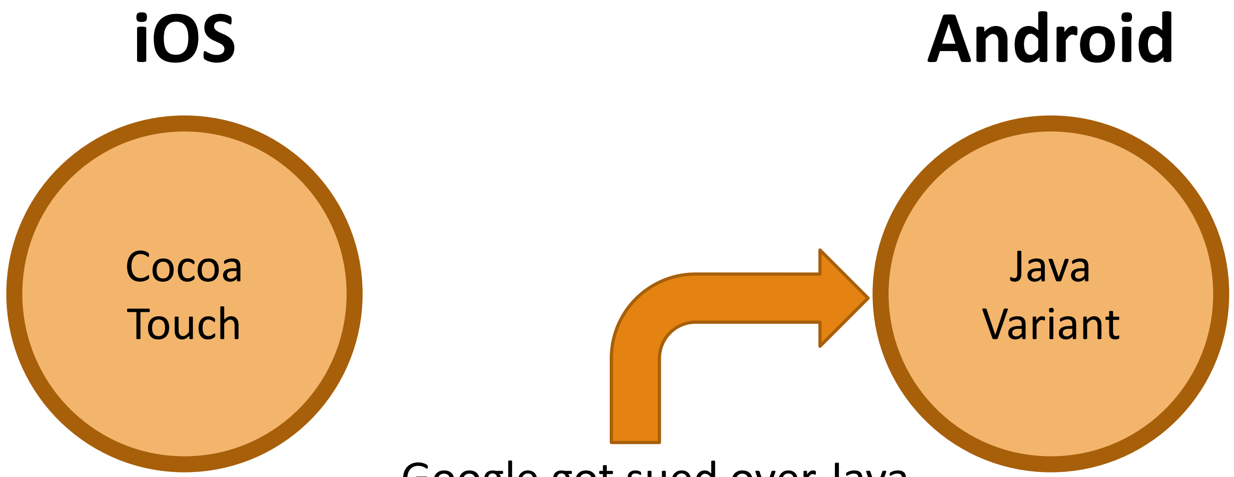
iOS

Cocoa Touch

Android

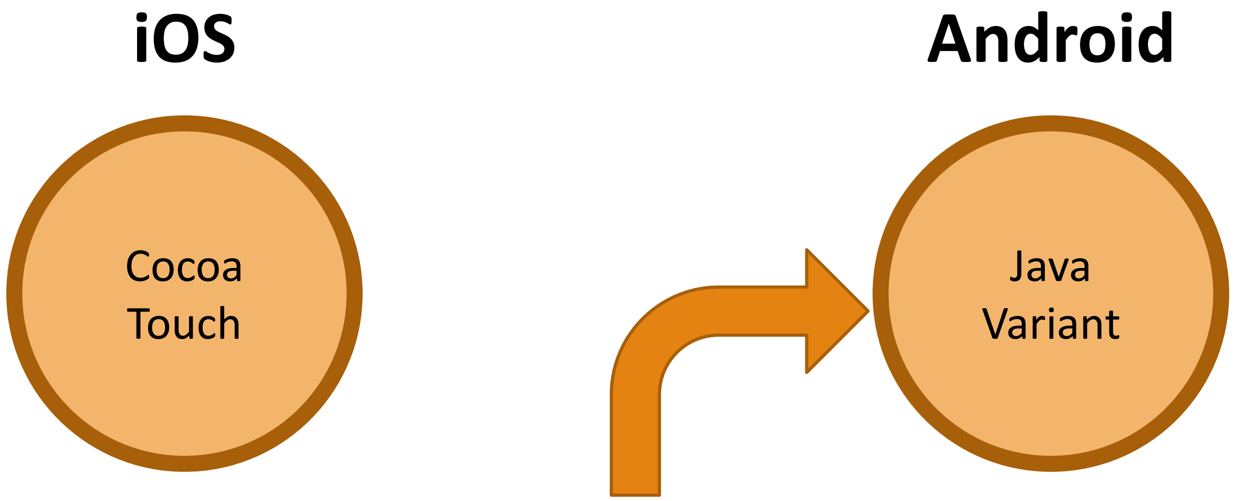
Java Variant

Mobile Platform Architecture



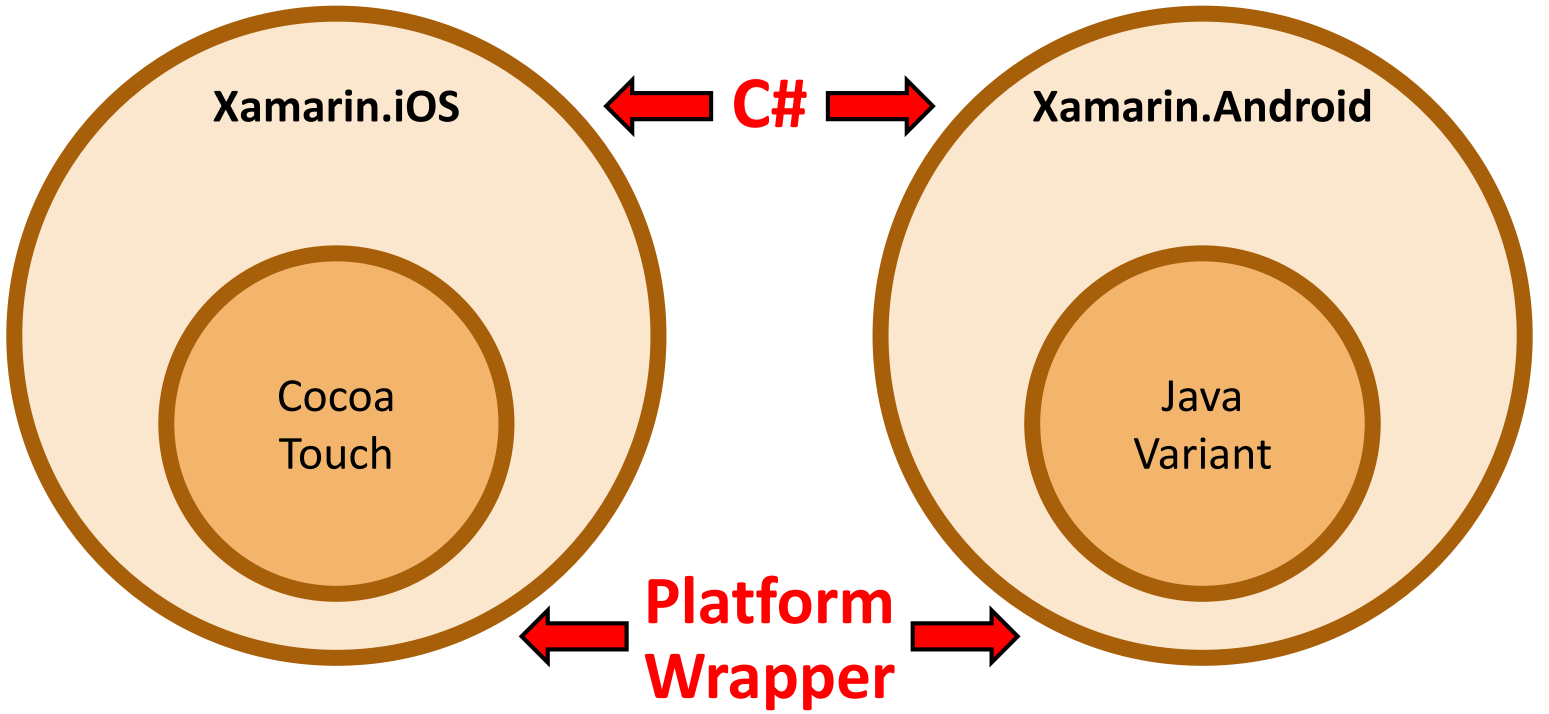
Google got sued over Java Patent infringement and had to prove it wasn't using Java, just compiling to the Java language standard

Mobile Platform Architecture

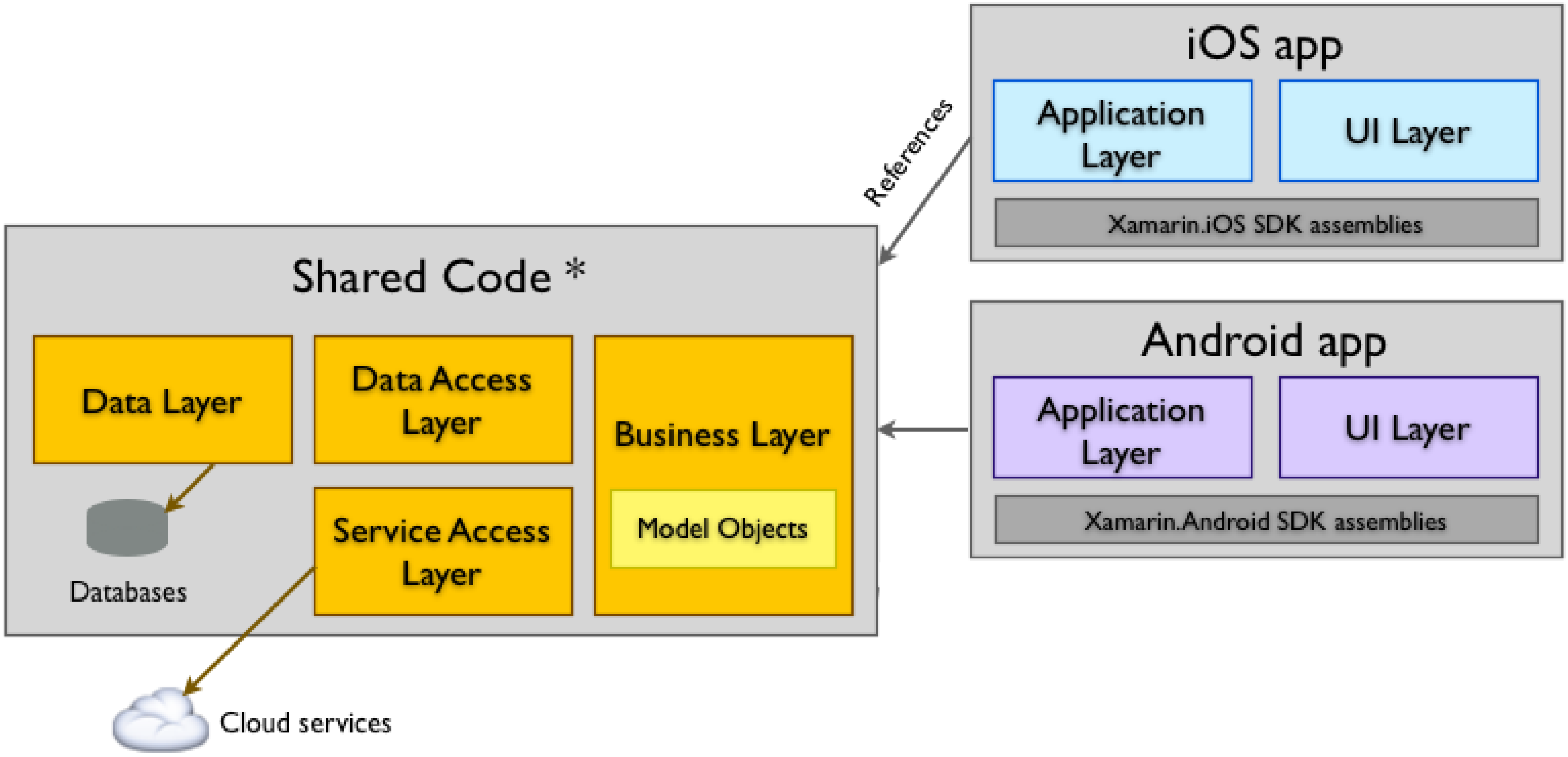


Each hardware manufacturer has to write there own JRE which introduces subtle implementation variations in the JDK.

Xamarin Architecture



Code Sharing – Non-Forms



** Portable Class Library or Shared Asset Project*

Code Sharing – Non-Forms

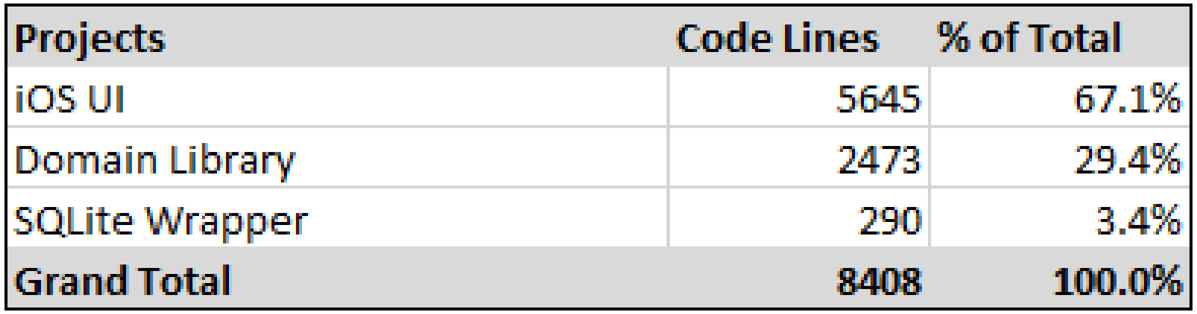
I don't know what the split is because lack of discipline can put more code in the UI than is required

- Guesstimate 60% will be UI code
  - Mine is 67.1%, but doing it again, there are many places I could have optimized if I knew what I was doing the first time.
  - Not much Domain code because features like LINQ reduces code
- There is a lot of transformational code to/from the view Model <-> View.

By the Numbers

The Bits

- 4 Projects
  - SQL Lite Wrapper
  - Domain Library
  - iOS User Interface
  - Android User Interface (tbd)
- 3 Main Tables + 1 Settings Table
- 40 User Interfaces
- 100 Classes
- 8400 lines of code (w/o blanks or comment lines)

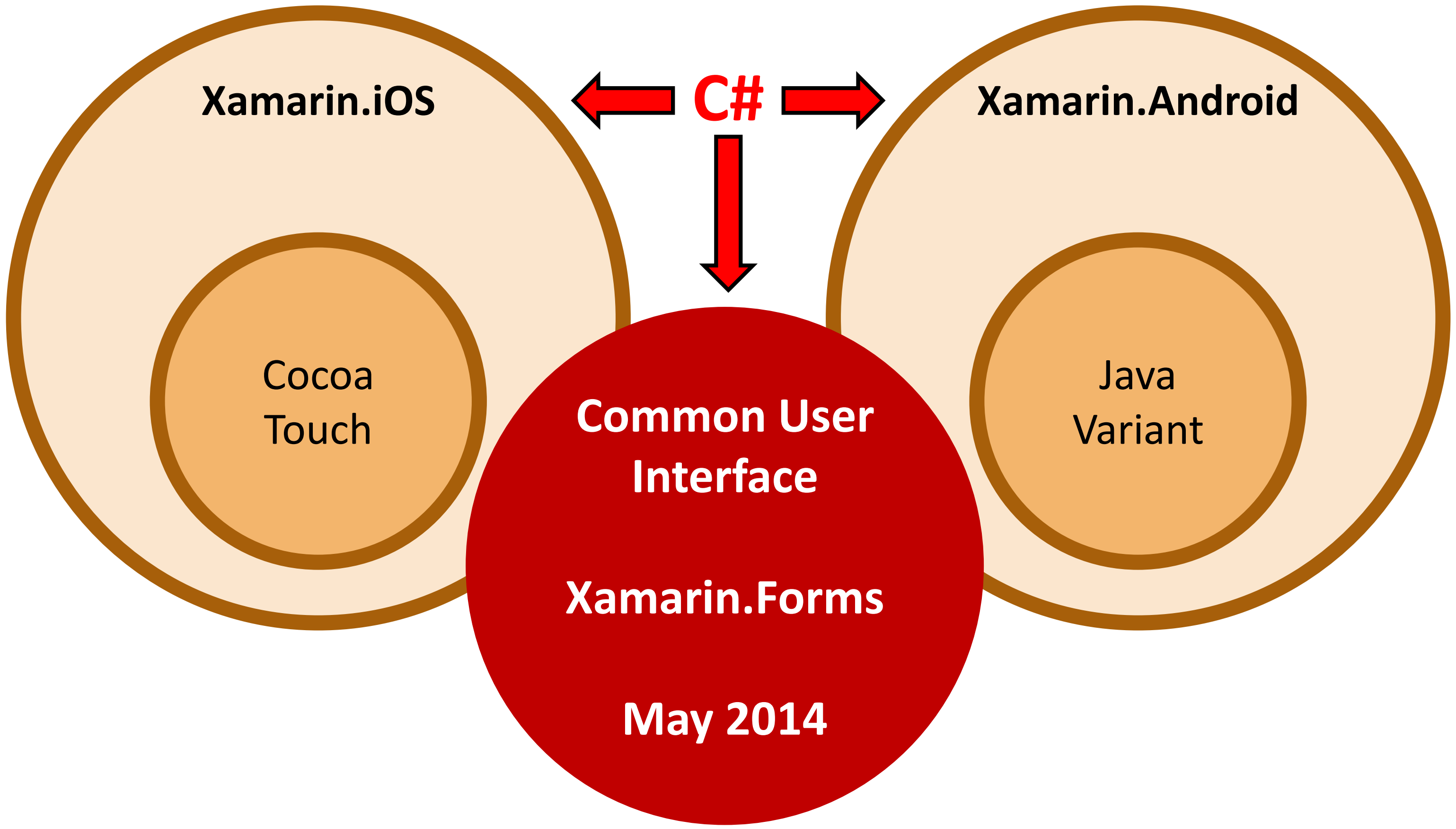


Projects	Code Lines	% of Total
iOS UI	5645	67.1%
Domain Library	2473	29.4%
SQLite Wrapper	290	3.4%
Grand Total	**8408**	**100.0%**

Xamarin.Forms

Addressing the silos by enabling a generic User Interface

Xamarin Architecture



Xamarin.Forms uses XAML

XAML is Microsoft's XML-Based UI specification protocol

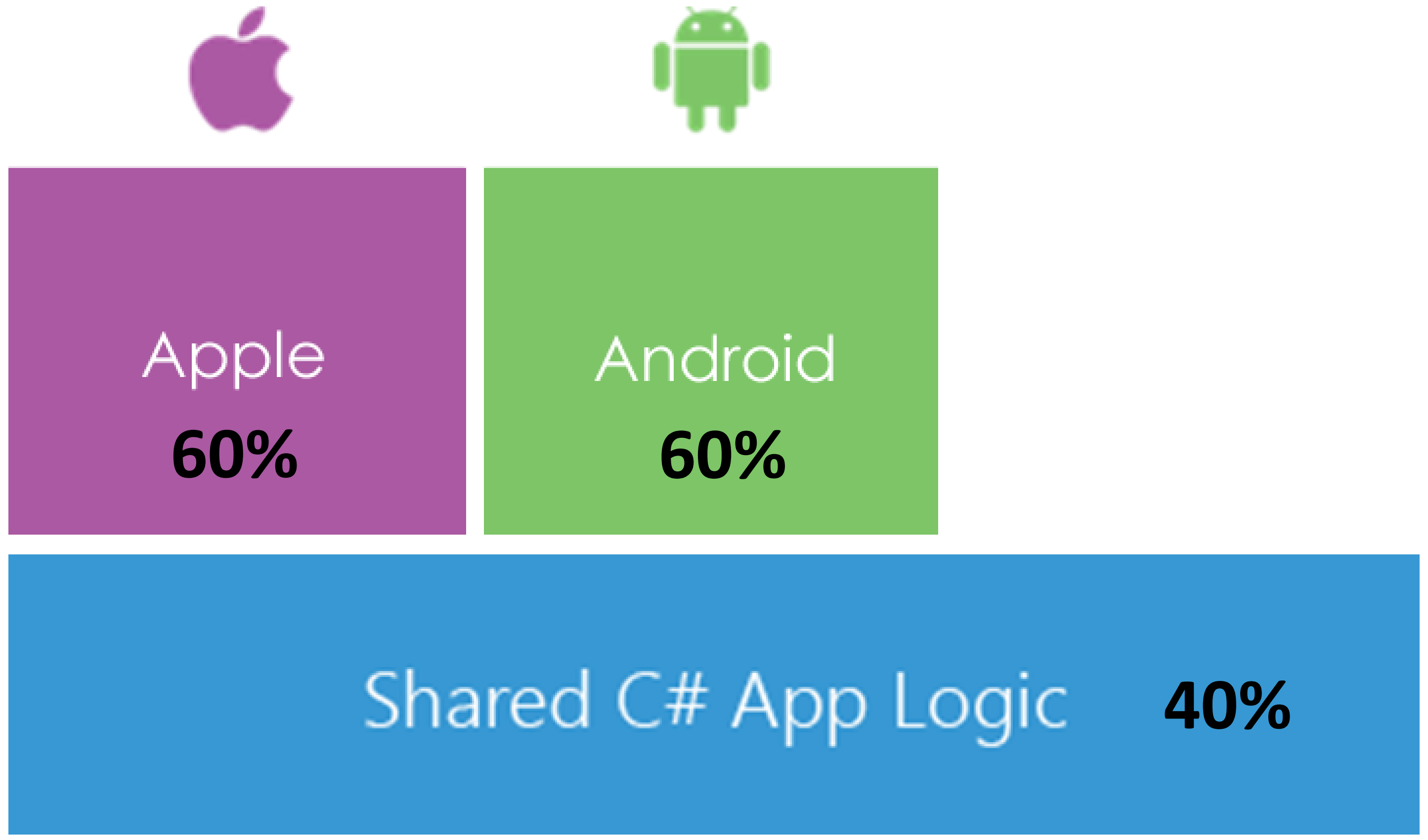
- You can also build programmatically

There are no XAML visual designers within Xamarin Studio.

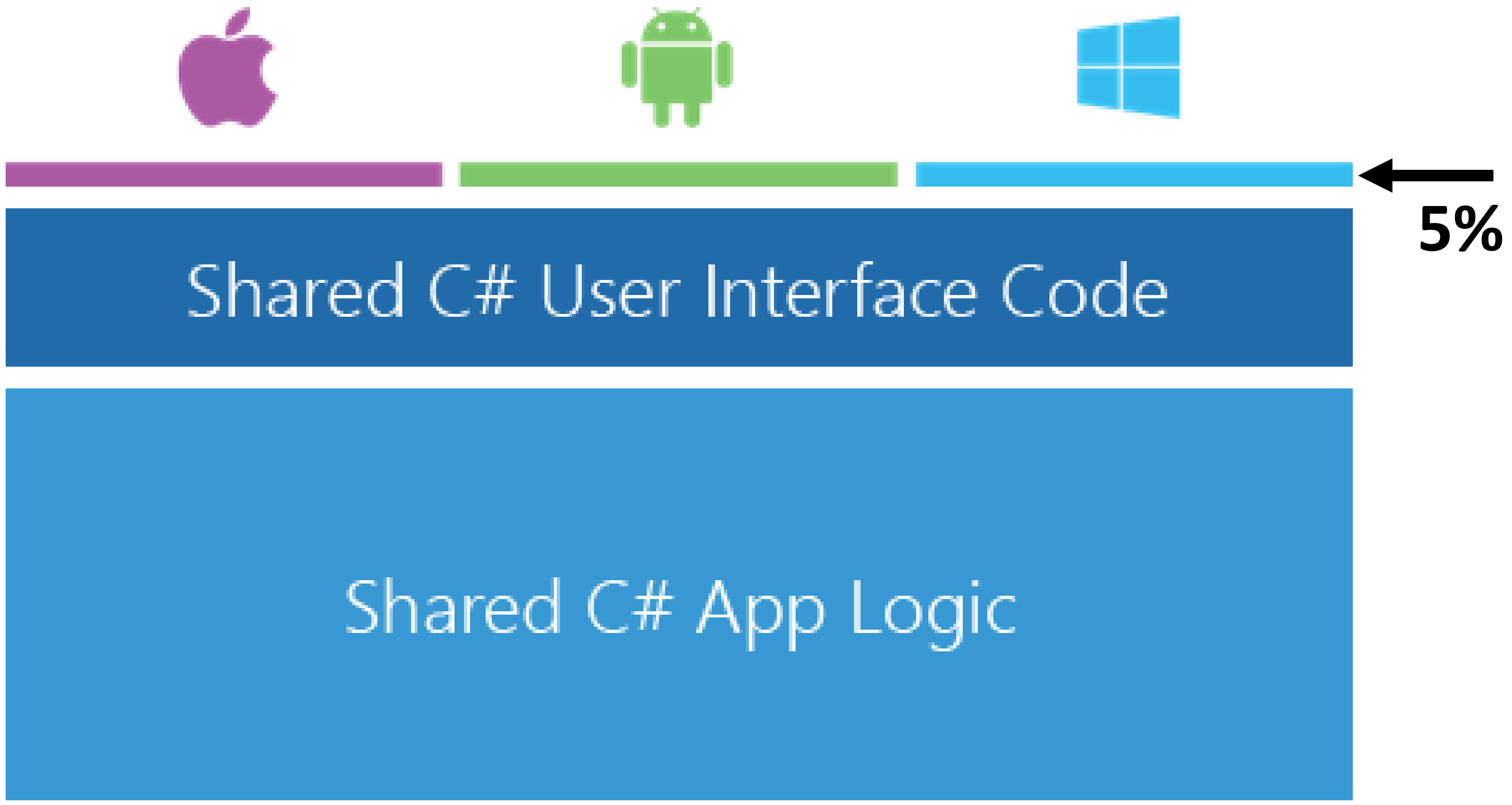
- http://kaxaml.com/ - Visual Tool

XAML is Windows Phone Compatible

Code Sharing – without Forms

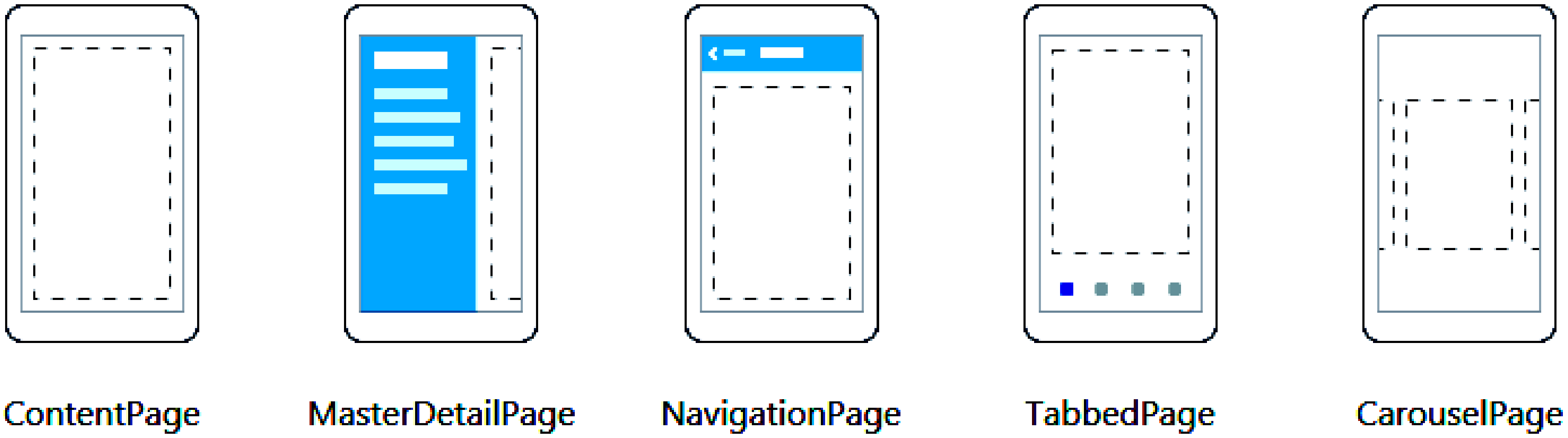


Code Sharing – with Forms

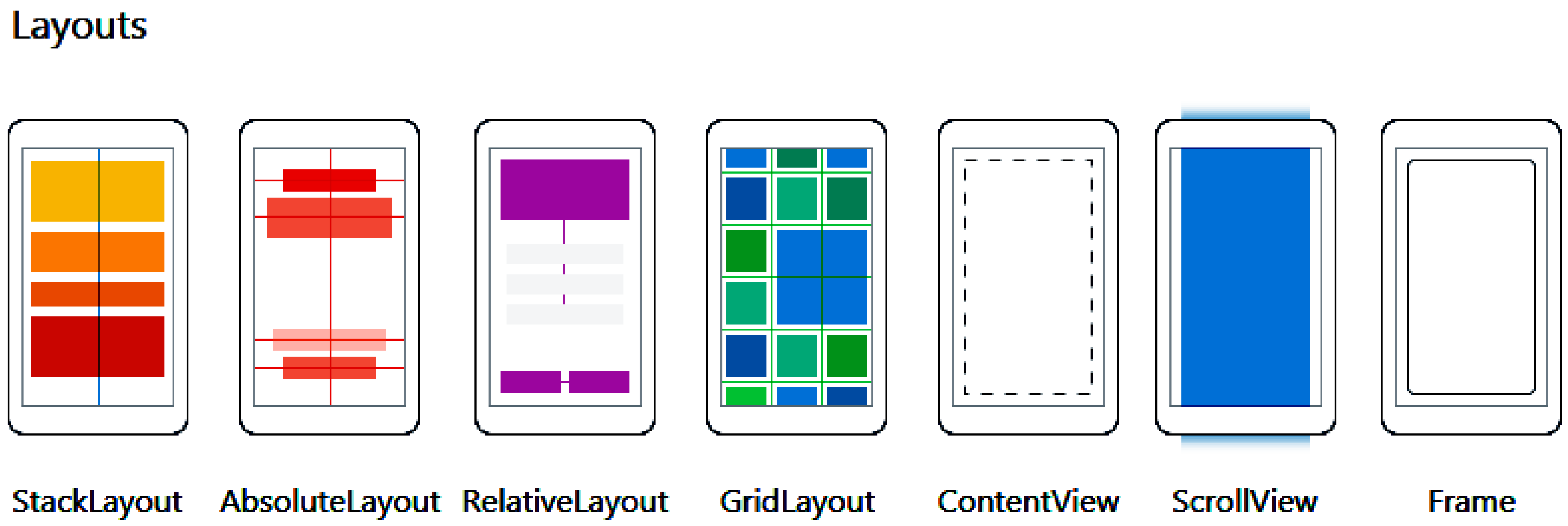


Xamarin.Forms – Pages

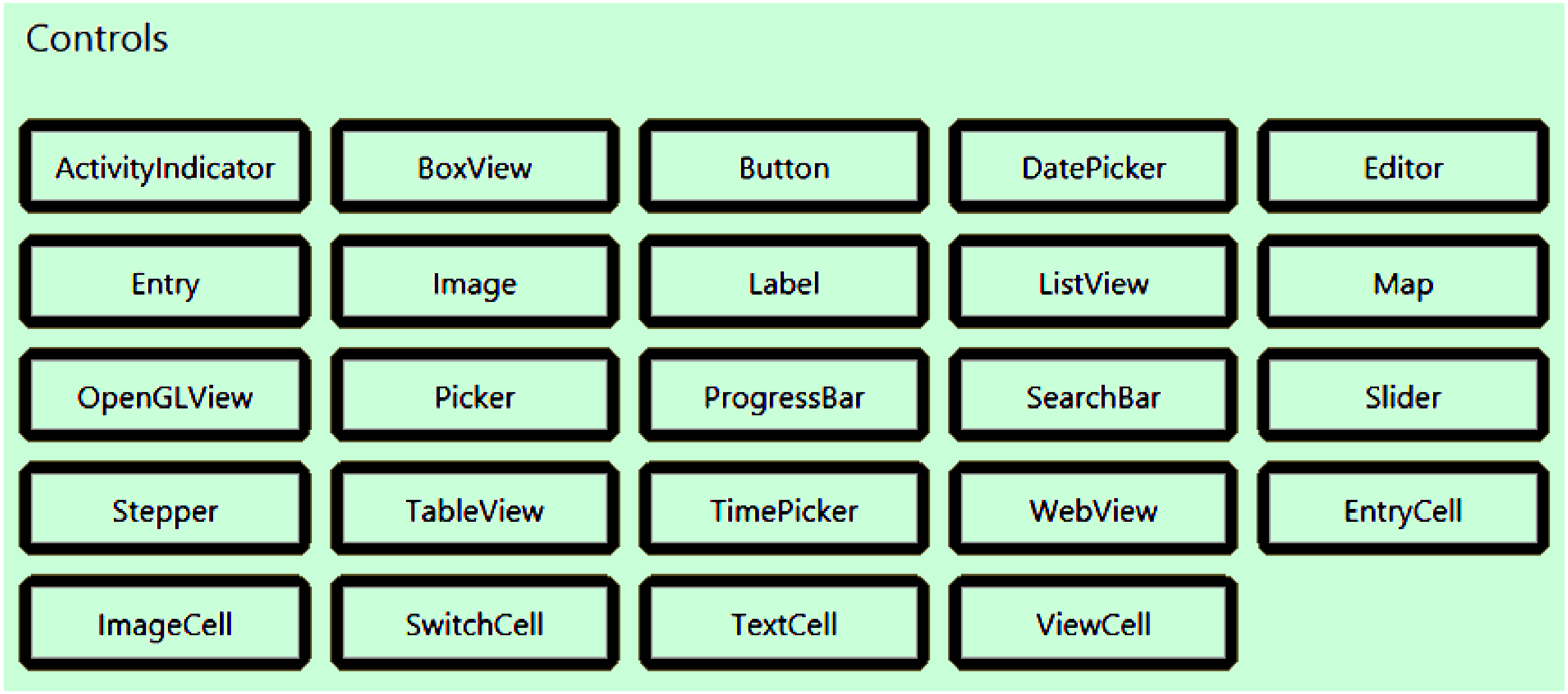
Pages



Xamarin.Forms – Layouts



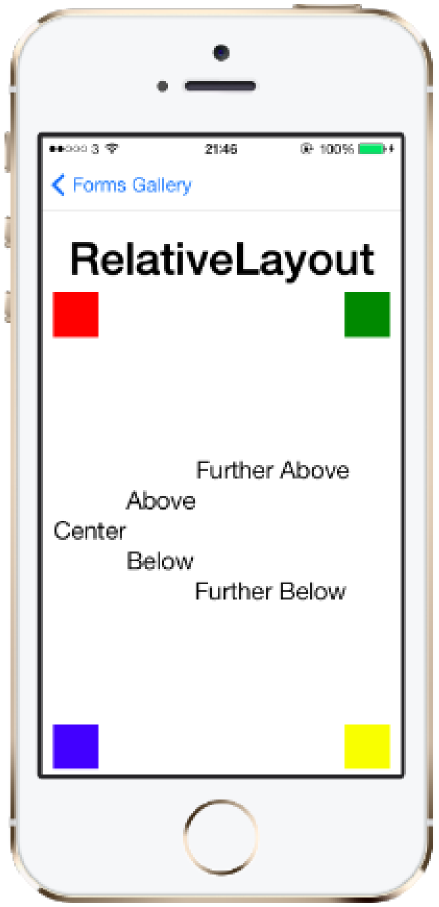
Xamarin.Forms – Controls

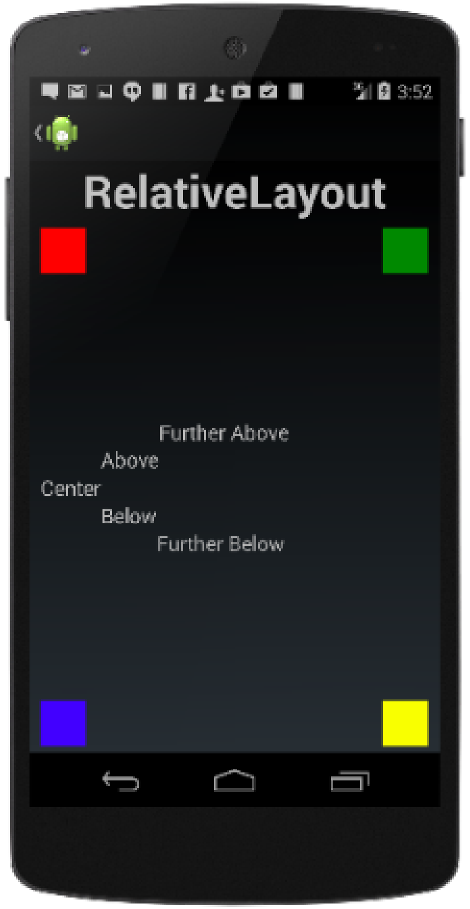


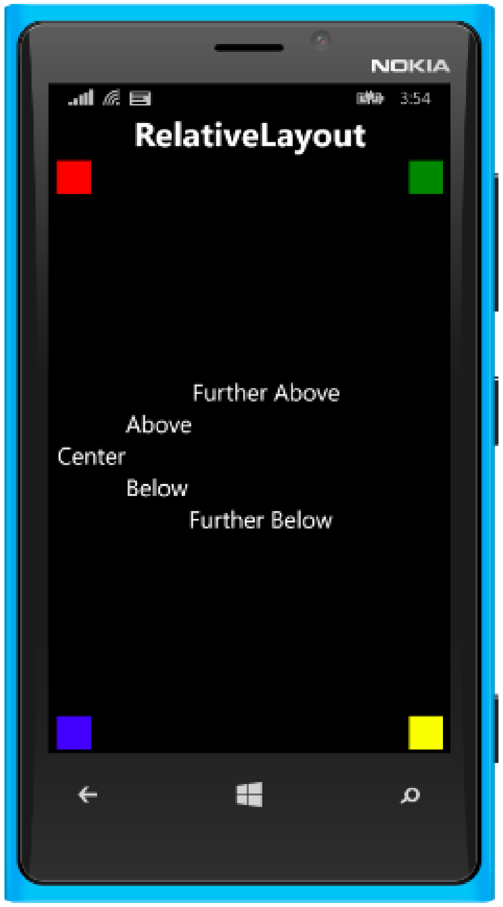
Controls

ActivityIndicator	BoxView	Button	DatePicker	Editor
Entry	Image	Label	ListView	Map
OpenGLView	Picker	ProgressBar	SearchBar	Slider
Stepper	TableView	TimePicker	WebView	EntryCell
ImageCell	SwitchCell	TextCell	ViewCell	

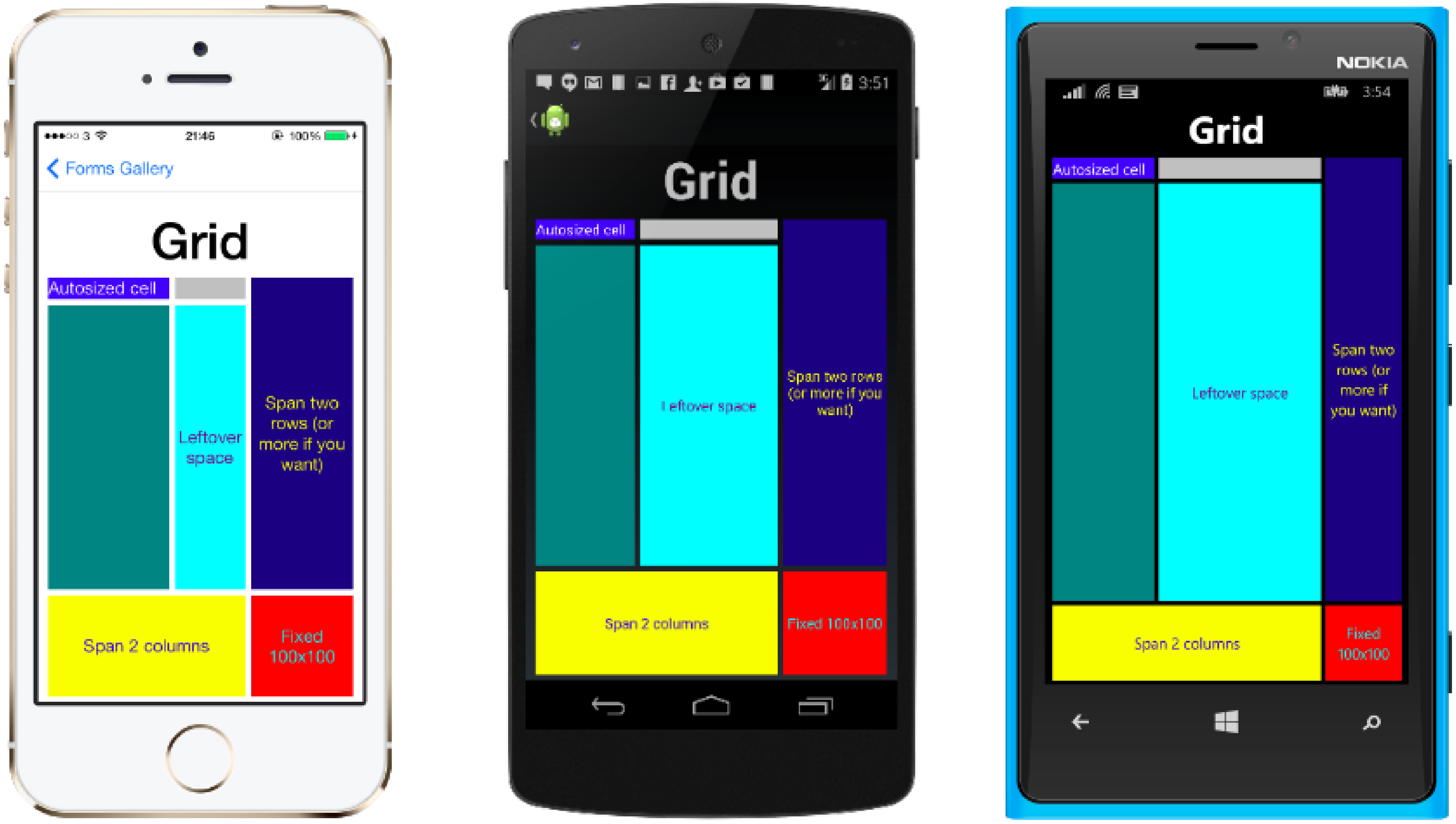
Result


●●○○○ 3
21:46
100%
Forms Gallery
RelativeLayout
Further Above
Above
Center
Below
Further Below

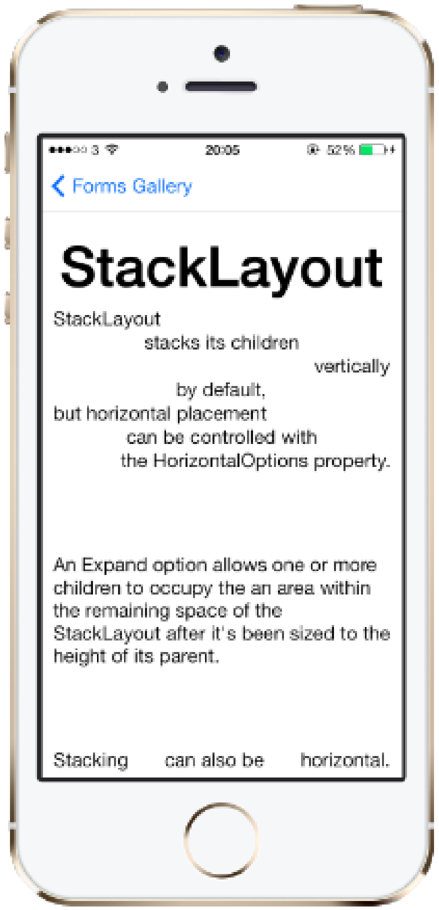

3:52
RelativeLayout
Further Above
Above
Center
Below
Further Below


NOKIA
3:54
RelativeLayout
Further Above
Above
Center
Below
Further Below

Result


Forms Gallery
Grid
Autosized cell
Leftover space
Span two rows (or more if you want)
Span 2 columns
Fixed 100x100
Grid
Autosized cell
Leftover space
Span two rows (or more if you want)
Span 2 columns
Fixed 100x100
NOKIA
Grid
Autosized cell
Leftover space
Span two rows (or more if you want)
Span 2 columns
Fixed 100x100

Result

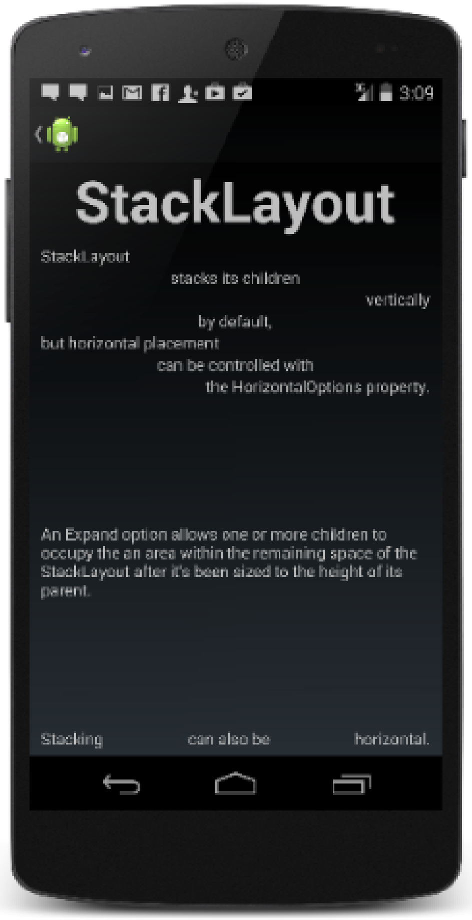


StackLayout

StackLayout
stacks its children
vertically
by default,
but horizontal placement
can be controlled with
the HorizontalOptions property.

An Expand option allows one or more children to occupy the an area within the remaining space of the StackLayout after it's been sized to the height of its parent.

Stacking can also be horizontal.

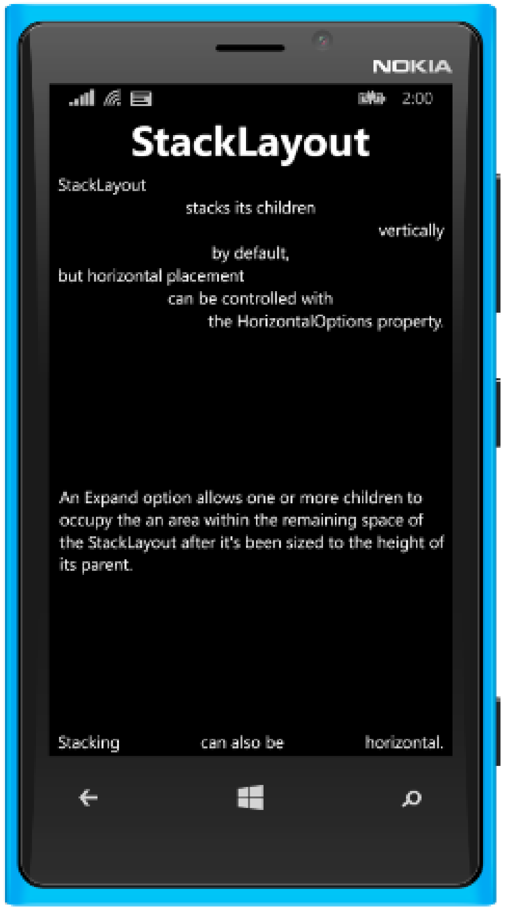


StackLayout

StackLayout
stacks its children
vertically
by default,
but horizontal placement
can be controlled with
the HorizontalOptions property.

An Expand option allows one or more children to occupy the an area within the remaining space of the StackLayout after it's been sized to the height of its parent.

Stacking can also be horizontal.



StackLayout

StackLayout
stacks its children
vertically
by default,
but horizontal placement
can be controlled with
the HorizontalOptions property.

An Expand option allows one or more children to occupy the an area within the remaining space of the StackLayout after it's been sized to the height of its parent.

Stacking can also be horizontal.

Xamarin Love/Hate

The Love / Hate Relationship

(aka, Likes, Dislikes and Hates)

Important Point

There are a lot of dislikes, don’t conclude that I really don’t like the environment

- *I really do like Xamarin and the platform*
- *The dislikes are more about pains you’ll encounter as you develop, rather than thinking this makes it unbareable to use.*

Likes

Xamarin Studio on Mac

- Works almost as well as VS2013
- It's a decent baby brother of ReSharper
- It's only US$25/month (could be free too)

Xamarin Frameworks are Open Source

- So it can help understand what it's actually doing
- Make changes or enhancements for your projects

iOS URLs Scheme

- A carefully crafted URL allows one app to start another app and pass parameters

iOS Model-View-Control is well implemented

Little Dislikes

You have to do it ALL yourself

- Expected Xamarin.iOS to be more abstract
- The “share” community isn’t that large
- Stack Overflow better “staffed” than Forums

Xamarian.iOS’s rename from MonoTouch

- Google searches are impossible:
  - Google strips the “.” and fetches mostly Apple iOS results
  - Objective-C is different enough from Xamarian.iOS to make results somewhat worthless.

Big Dislikes

Windows Xamarin Studio --> Mac is yucky

- Insanely complex to configure
- Flaky as heck; they say it’s better (think BETA)

Xamarin Studio doesn’t have Visualizers

- You can construct View Controllers, but there is no way to visualize what it looks like.

No Resolution, Portrait, Landscape support

- You have to do it yourself.
- Xamarin.Forms will help.

Little Hates

Mac Keyboard

- Working in Windows and Mac is confusing as much as it is frustrating. CTRL, WIN, and ALT equivalents are all moved around, as well as keyboard short cuts are wildly different

Simulator and iOS Device Differences

- APIs unavailable; cannot install apps (i.e. DropBox), so testing can only occur on the device
- Filenames are case insensitive on simulator
- Debugging times on devices are substantially longer and sometimes fail to start; wasted time

Major Hates

Threading

- When and why does threading take place?
- Communications between threads are confusing
  - Sometimes they are impossible to coordinate.
  - One such thing took a ½ day rewrite to overcome.
- Silent Failures
  - Code simply stops without any notification or exceptions

17 Seconds to Freedom

- Debugging device startup issues … GRRRRRRRR!
  - How frustrating? Throw the device at the wall frustrating !

Alternative Mobile Development Platforms

Other platforms I looked at before deciding on Xamarin

Alternative #1

Phone Gap

Phone Gap

JavaScript Framework

- Uses HTML5 and CSS3 for specification
- But is transformed to a Native view

I Considered Phone Gap

Framework

- It felt like jumping through hoops just to do the basics
- Almost impossible to write abstract decoupled code

JavaScript

- No Types
- Isn’t Object Oriented
- Can create very buggy and bug elusive code

Many report that:

- Phone Gap apps are very non-native looking
- Phone Gap apps are very slow; limiting the functionality you can put in them

Ultimately rejected it as a platform

Alternative #2 - Unity 3D

Unity 3D

Uses C# as it’s key scripting language

- But it’s scripts, not programs/classes

Unity is a 3D Visualization Engine

- Used primarily as a gaming platform

It’s a high quality 3D engine and is used for business apps requiring 3D visualization

- I’m seen app to visualize oil & gas reservoirs based on GPS locations

Unity 3D

I'd use Unity over Xamarin if I had to make any kind of 3D or 2D game or 3D App

- It has multiple deployment platforms including iOS, Android, Mac, Windows and Flash (web)

It's more complicated to develop in it

- However there is a ton of support.

Prices:

- Free or $75/mo/platform for advanced pro features

Design Choices

Design Choices

For iOS

- Use a Cocoa Touch look and feel:
  - Use Xamarin.iOS as the development framework
  - 87% of devices use iOS7, therefore, can provide universal feel

For Android

- Use Xamarin.Forms
  - This abstracts the mobile UI (decouples it)
  - Assuming, this makes it far easier to deal with the 1,000s of devices

App Architecture

Architecture Highlights

- Identity Field
- Single Table Inheritance
  - Memento
- Factories
- Value Database
  - RDBMS and Data Warehouse Hybrid
- Lazy Loading
- Identity Map

Open Sourced – Soon

https://github.com/Knowtie/SQLiteRepository

- Identity Field (sort of) – It’s implied
- Lazy Loading
- Identity Map

Identity Field

Fowler Enterprise Architecture Pattern

Identity Field

Using a single field consistently to identify object values.

- Typically a GUID or an Integer
- Post Relational Era, referred to as a Surrogate Key

KB uses an Integer as the Identity Field

- Every table has an OID
- Foreign Keys are TABLE_OID

Single Table Inheritance

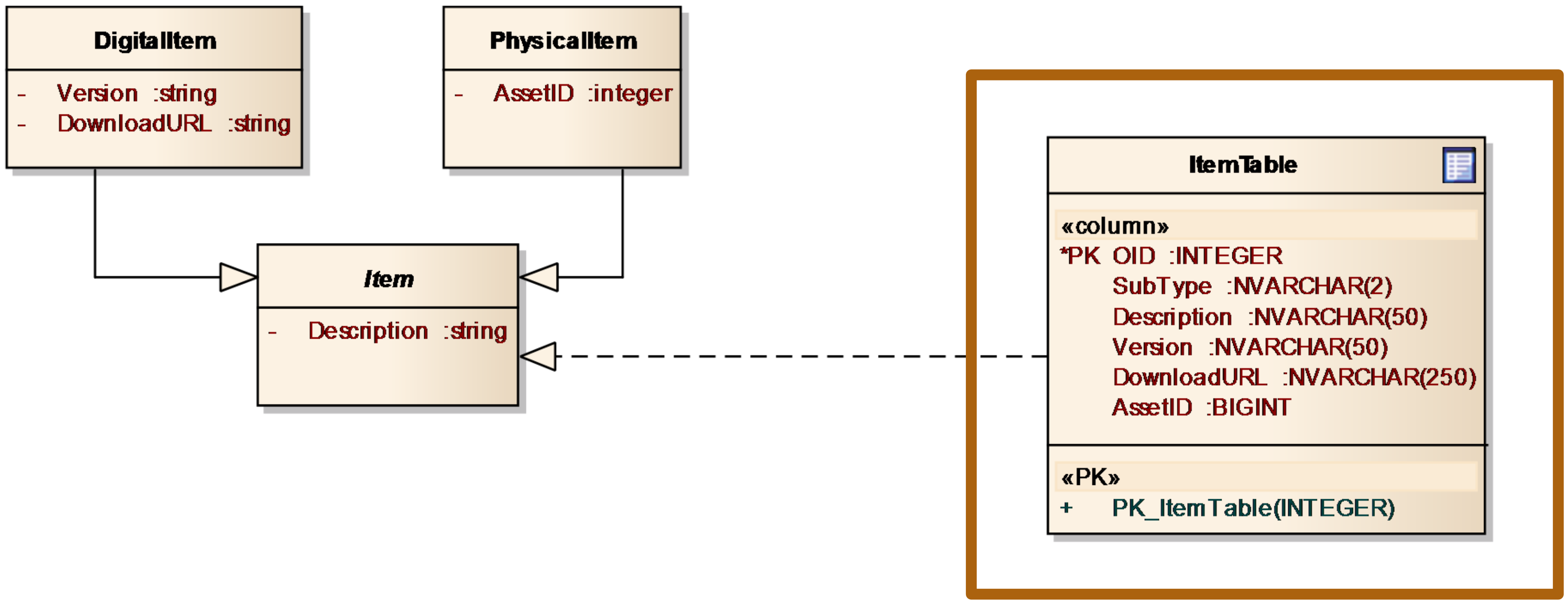
Fowler Enterprise Architecture Pattern

+ Memento – Design Patterns GoF

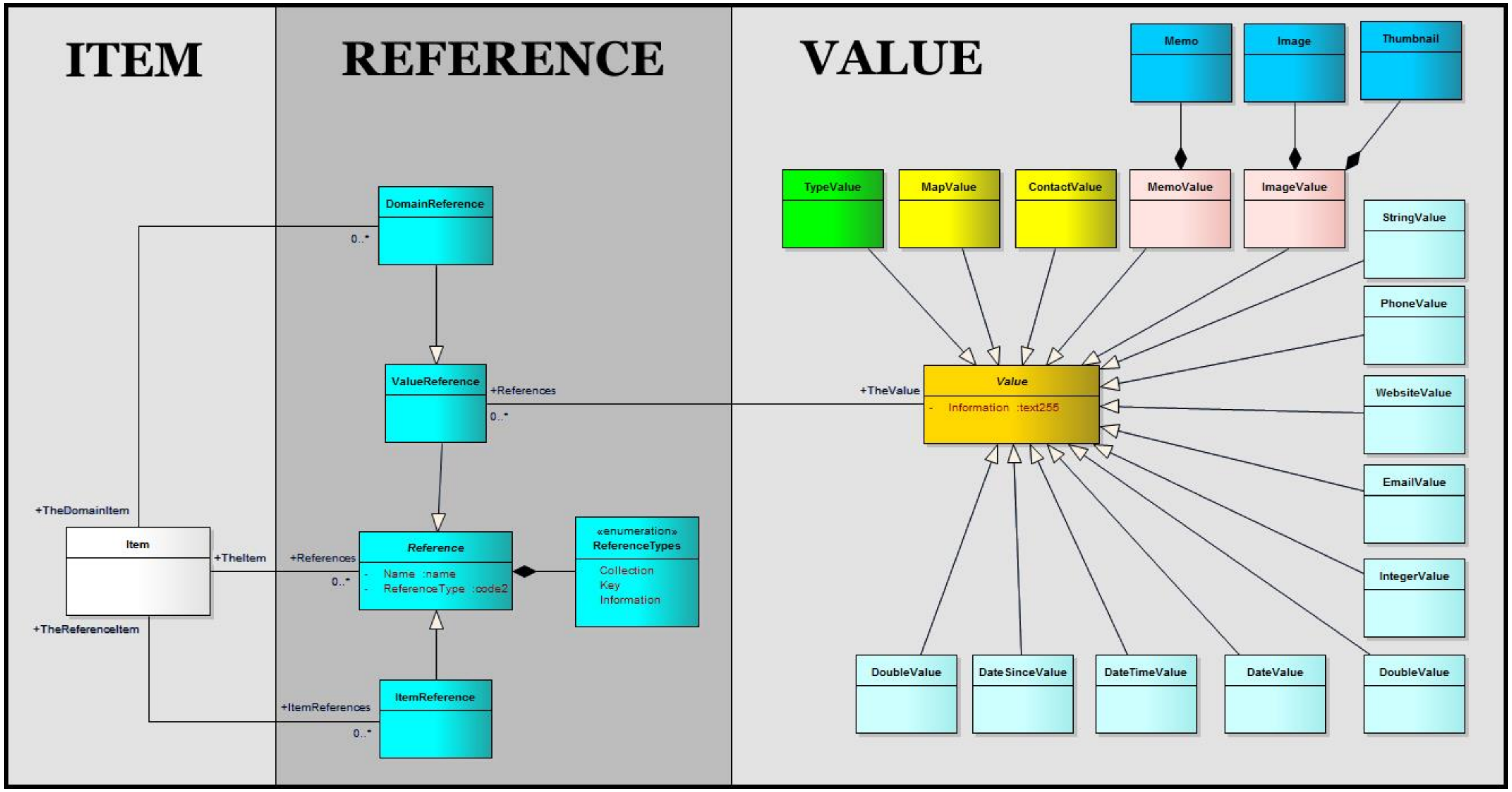
Single Table Inheritance

A single table is used to store a base class and all subclasses.

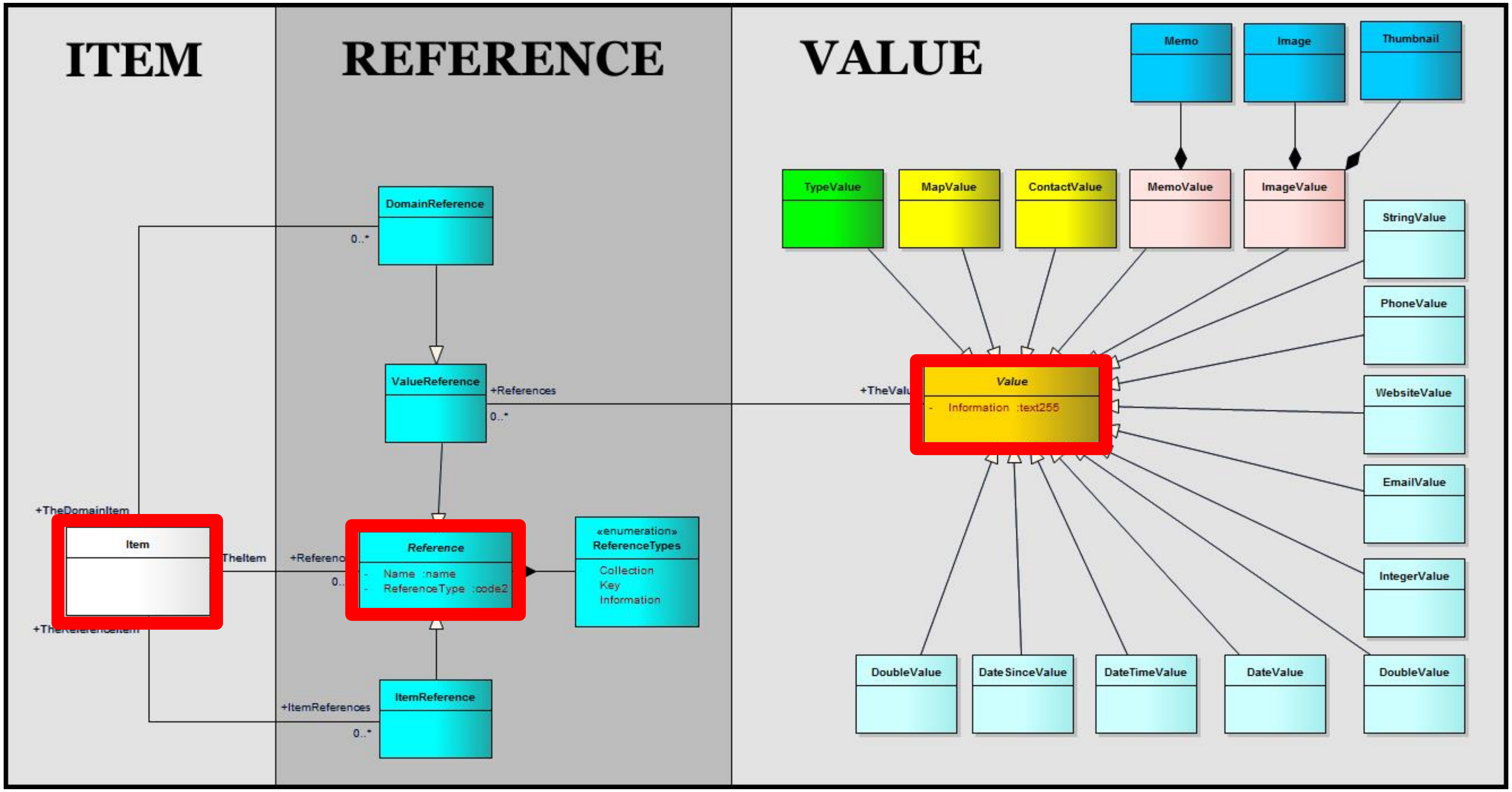
- A “SubType” differentiates on subclass from another



Single Table Inheritance



Single Table Inheritance

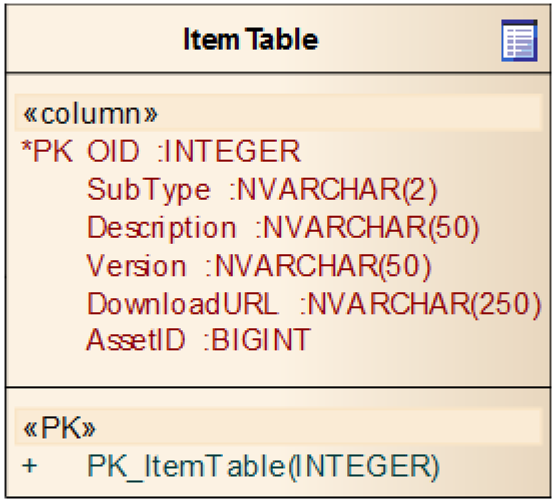


Factories

Design Patterns/Gang of Four

Factories / Factory Method

Deferring the construction of an object to something else.



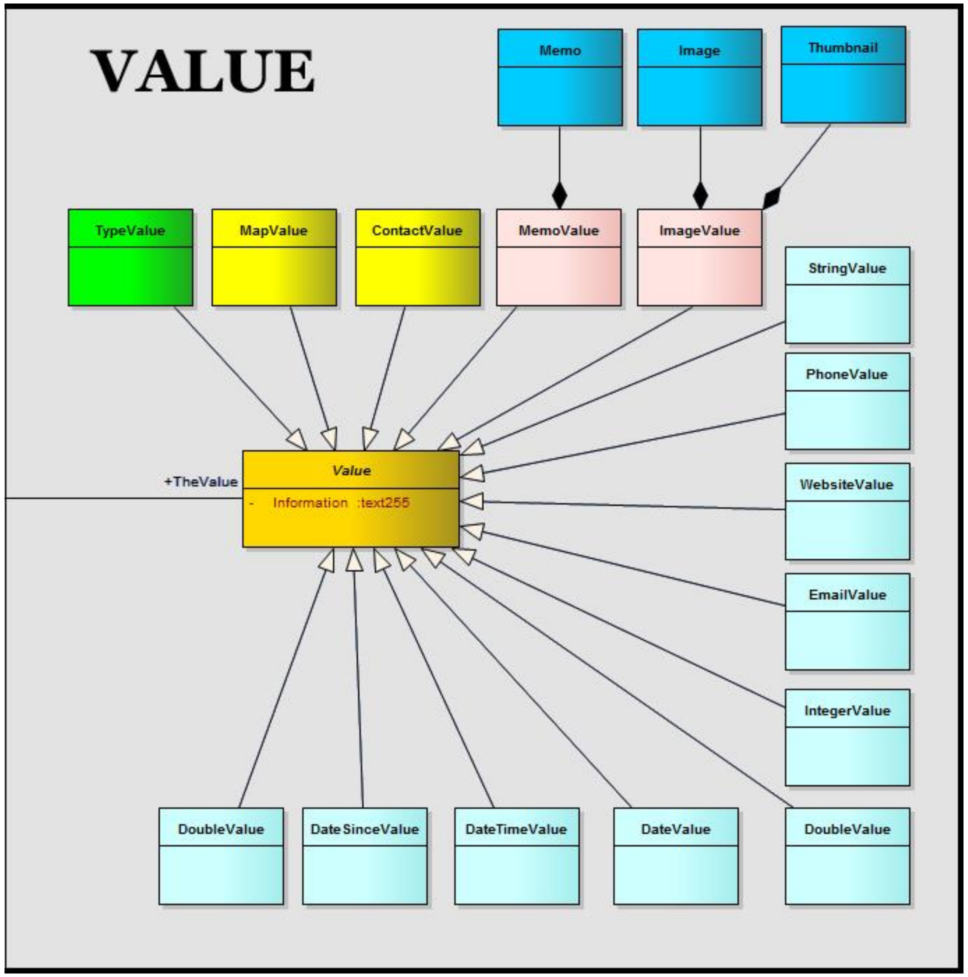
```
Item Table
«column»
*PK OID :INTEGER
    SubType :NVARCHAR(2)
    Description :NVARCHAR(50)
    Version :NVARCHAR(50)
    DownloadURL :NVARCHAR(250)
    AssetID :BIGINT
«PK»
+   PK_ItemTable(INTEGER)
```

```
If ( SubType == "PI" ) {
    construct PhysicalItem
          from ItemTable;

} else if ( SubType == "DI" ) {
    construct DigitalItem
          from ItemTable;

}
```

Factories / Factory Method

Value Factory

- Is implemented as a delegate dictionary.
- Removes a large IF block with something more efficient.
  - Functionally, 2 lines of code
  - 1, if your not defensive



Delegate Factory Method

```
public delegate Value ValueFactory( ValueTab aValueRow );
```

Delegate Factory Method

```
public delegate Value ValueFactory( ValueTab aValueRow );
```

```
Dictionary<string, ValueFactory> ValueFactories = new ...
```

Delegate Factory Method

```
public delegate Value ValueFactory( ValueTab aValueRow );

Dictionary<string, ValueFactory> ValueFactories = new ...

public static Value StringValueFactory( ValueTab aValueRow ) {
    return new StringValue( aValueRow );
}
```

Delegate Factory Method

```
public delegate Value ValueFactory( ValueTab aValueRow );

Dictionary<string, ValueFactory> ValueFactories = new ...

public static Value StringValueFactory( ValueTab aValueRow ) {
    return new StringValue( aValueRow );
}

ValueFactories.Add( “STRING” , StringValueFactory );
```

Delegate Factory Method

```
public delegate Value ValueFactory( ValueTab aValueRow );

Dictionary<string, ValueFactory> ValueFactories = new ...
```

```
public static Value StringValueFactory( ValueTab aValueRow ) {
    return new StringValue( aValueRow );
}

ValueFactories.Add( "STRING" , StringValueFactory );
```

```
ValueFactories.Add( "STRING" , delegate( ValueTab aRow ) {
     return new StringValue( aRow ); } );
```

Anonymous Delegate

Delegate Factory Method

```
public delegate Value ValueFactory( ValueTab aValueRow );

Dictionary<string, ValueFactory> ValueFactories = new ...

public static Value StringValueFactory( ValueTab aValueRow ) {
    return new StringValue( aValueRow );
}

ValueFactories.Add( "STRING" , StringValueFactory );
```

```
ValueFactories.Add( "STRING" , delegate( ValueTab aRow ) {
     return new StringValue( aRow ); } );
```

```
ValueFactories.Add( "STRING" , aRow => new StringValue( aRow ) );
```

Lambda Expression

Delegate Factory Method

```
public delegate Value ValueFactory( ValueTab aValueRow );

Dictionary<string, ValueFactory> ValueFactories = new ...

public static Value StringValueFactory( ValueTab aValueRow ) {
    return new StringValue( aValueRow );
}

ValueFactories.Add( "STRING" , StringValueFactory );

ValueFactories.Add( "STRING" , aRow => new StringValue( aRow ) );

if ( ValueFactories.Contains( aValueRow.SubType ) {
    return ValueFactories[ aValueRow.SubType ]( aValueRow );
else {
    // Some kind of error occurs here
}
```

Finally the Use

Delegate Factory Method

```
public delegate Value ValueFactory( ValueTab aValueRow );

Dictionary<string, ValueFactory> ValueFactories = new ...

public static Value StringValueFactory( ValueTab aValueRow ) {
    return new StringValue( aValueRow );
}

ValueFactories.Add( "STRING" , StringValueFactory );

ValueFactories.Add( "STRING" , aRow => new StringValue( aRow ) );

if ( ValueFactories.Contains( aValueRow.SubType ) {
    return ValueFactories[ aValueRow.SubType ]( aValueRow );
else {
    // Some kind of error occurs here
}
```

Defensive Programming

Lazy Load

Design Patterns/Gang of Four

Lazy Loading

Lazy Loading is all about loading objects into memory on an "as needed" basis.

- Easier to code, but more expensive I/O
  - You aren't custom designing SQL statements per screen
- Decouples the object model from the data model
  - How and Where objects are stored is independent of the object model, and moveable so long as the behavior is maintained.
- Loads objects into memory as you reference them
  - If (Objects NOT loaded) then { load objects }
  - It appears as if the object were always loaded

Lazy Loading (Children)

```
public class LazyLoadChildren<CHILDCLASS,MASTERCLASS,CHILDTABLE>
                           : IList<CHILDCLASS>
where CHILDCLASS : BaseClass
where MASTERCLASS : BaseClass
where CHILDTABLE : IBaseTable,new() { }
```

Declaration

Lazy Loading (Children)

```
public class LazyLoadChildren<CHILDCLASS,MASTERCLASS,CHILDTABLE>
                            : IList<CHILDCLASS>
where CHILDCLASS : BaseClass
where MASTERCLASS : BaseClass
where CHILDTABLE : IBaseTable,new() { }
```

Declaration

```
public LazyLoadChildren( MASTERCLASS aMaster
, string aQueryFieldOnChildTable ) {
    _Master = aMaster;
    _QueryFieldOnChildTable = aQueryFieldOnChildTable;
}
```

Lazy Loading (Children)

```
public class LazyLoadChildren<CHILDCLASS,MASTERCLASS,CHILDTABLE>
                            : IList<CHILDCLASS>
where CHILDCLASS : BaseClass
where MASTERCLASS : BaseClass
where CHILDTABLE : IBaseTable,new() { }
```

Declaration

```
public LazyLoadChildren( MASTERCLASS aMaster
, string aQueryFieldOnChildTable ) {
    _Master = aMaster;
    _QueryFieldOnChildTable = aQueryFieldOnChildTable;
}
```

Use

```
public LazyLoadChildren<ValueReference,Value,ReferenceTab>
                      References { get; private set; }
```

Lazy Loading (Children)

```
public class LazyLoadChildren<CHILDCLASS,MASTERCLASS,CHILDTABLE>
                              : IList<CHILDCLASS>
where CHILDCLASS : BaseClass
where MASTERCLASS : BaseClass
where CHILDTABLE : IBaseTable,new() { }
```

Declaration

```
public LazyLoadChildren( MASTERCLASS aMaster
, string aQueryFieldOnChildTable ) {
    _Master = aMaster;
    _QueryFieldOnChildTable = aQueryFieldOnChildTable;
}
```

Use

```
public LazyLoadChildren<ValueReference,Value,ReferenceTab>
                      References { get; private set; }
```

Lazy Loading (IList)

PROPERTIES	METHODS
Count	Add
IsFixedSize	Clear
IsReadOnly	Contains
IsSynchronized	CopyTo
Item	GetEnumerator
SyncRoot	IndexOf
	Insert
	Remove
	RemoveAt

Lazy Loading (Children)

```
public class LazyLoadChildren<CHILDCLASS,MASTERCLASS,CHILDTABLE>
                            : IList<CHILDCLASS>
where CHILDCLASS : BaseClass
where MASTERCLASS : BaseClass
where CHILDTABLE : IBaseTable,new() { }
```

Declaration

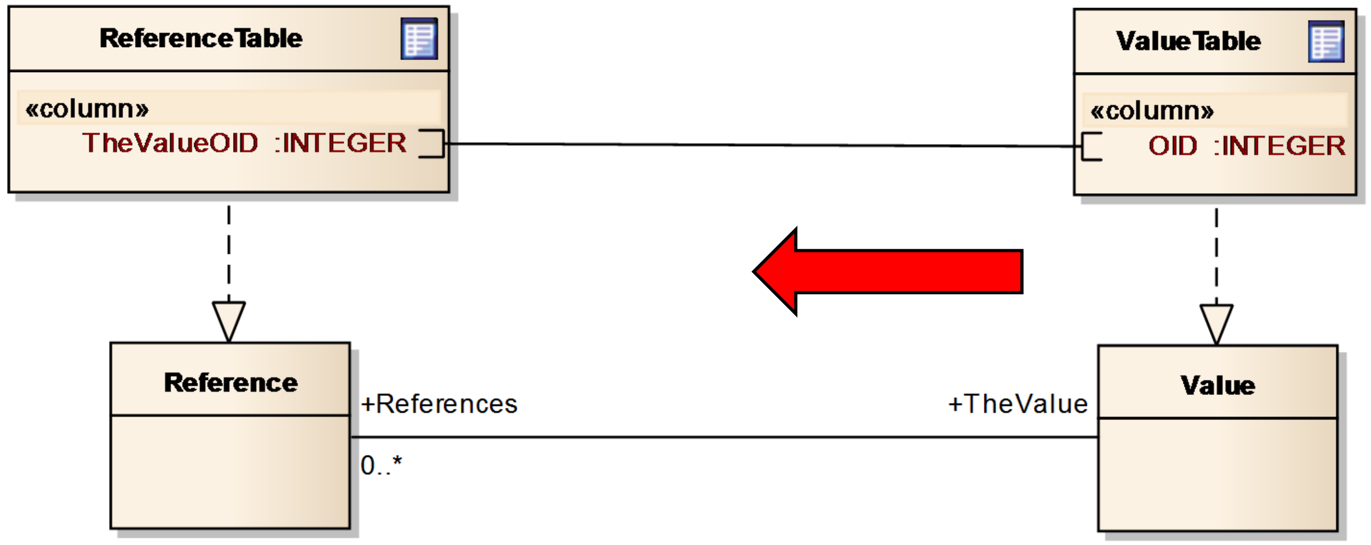
```
public LazyLoadChildren( MASTERCLASS aMaster
, string aQueryFieldOnChildTable ) {
    _Master = aMaster;
    _QueryFieldOnChildTable = aQueryFieldOnChildTable;
}
```

Use

```
public LazyLoadChildren<ValueReference,Value,ReferenceTab>
                       References { get; private set; }
```

```
References = new LazyLoadChildren<ValueReference, Value, ReferenceTab>
                 (this,"TheValueOID");
```

Lazy Loading (Children)

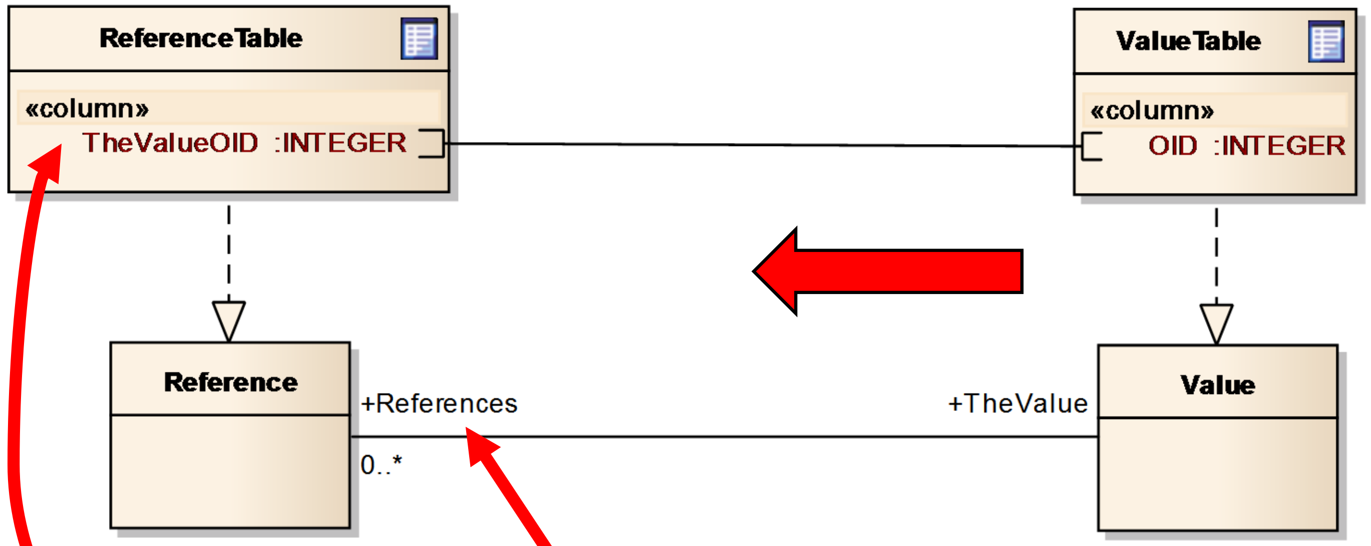


Use

```
public LazyLoadChildren<ValueReference,Value,ReferenceTab>
                                References { get; private set; }
```

```
References = new LazyLoadChildren<ValueReference, Value, ReferenceTab>
                 (this,"TheValueOID");
```

Lazy Loading (Children)

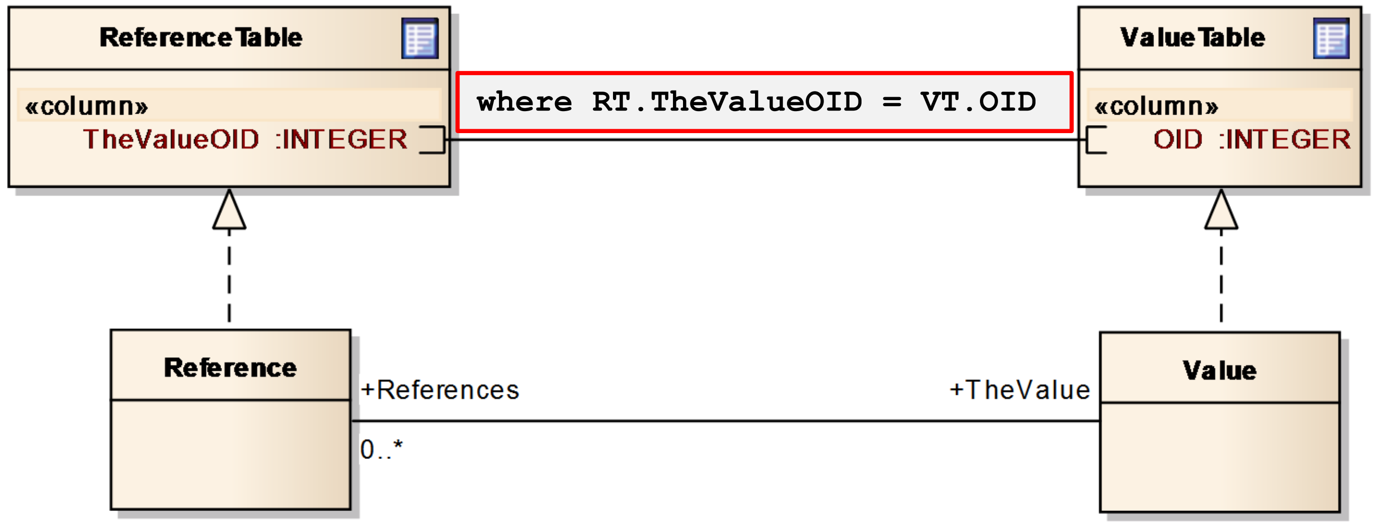


Use

```
public LazyLoadChildren<ValueReference,Value,ReferenceTab>
                          References { get; private set; }
```

```
References = new LazyLoadChildren<ValueReference, Value, ReferenceTab>
                 (this, "TheValueOID") ;
```

Lazy Loading (Children)



Use

```
public LazyLoadChildren<ValueReference,Value,ReferenceTab>
                        References { get; private set; }
```

```
References = new LazyLoadChildren<ValueReference, Value, ReferenceTab>
                 (this,"TheValueOID");
```

Lazy Loading (Master)

```
public class LazyLoadMaster<CLASS,TABLE>
where CLASS : BaseClass
where TABLE : IBaseTable { }
```

Lazy Loading (Master)

```
public class LazyLoadMaster<CLASS,TABLE>
where CLASS : BaseClass
where TABLE : IBaseTable { }
```

```
protected LazyLoadMaster<Value,ValueTab> _TheValue;
public Value TheValue {
     get { return _TheValue.Value; }
     set { _TheValue.Value = value; }
}
```

```
_TheValue = new LazyLoadMaster<Value,ValueTab>( aTheValueOID );
```

Lazy Loading (Master)

```
public class LazyLoadMaster<CLASS,TABLE>
where CLASS : BaseClass
where TABLE : IBaseTable { }
```

```
protected LazyLoadMaster<Value,ValueTab> _TheValue;
public Value TheValue {
     get { return _TheValue.Value; }
     set { _TheValue.Value = value; }
}
```

```
_TheValue = new LazyLoadMaster<Value,ValueTab>( aTheValueOID );
```

Initialized in the constructor

Lazy Loading (Master)

```
public class LazyLoadMaster<CLASS,TABLE>
where CLASS : BaseClass
where TABLE : IBaseTable { }
```

```
protected LazyLoadMaster<Value,ValueTab> _TheValue;
public Value TheValue {
     get { return _TheValue.Value; }
     set { _TheValue.Value = value; }
}
```

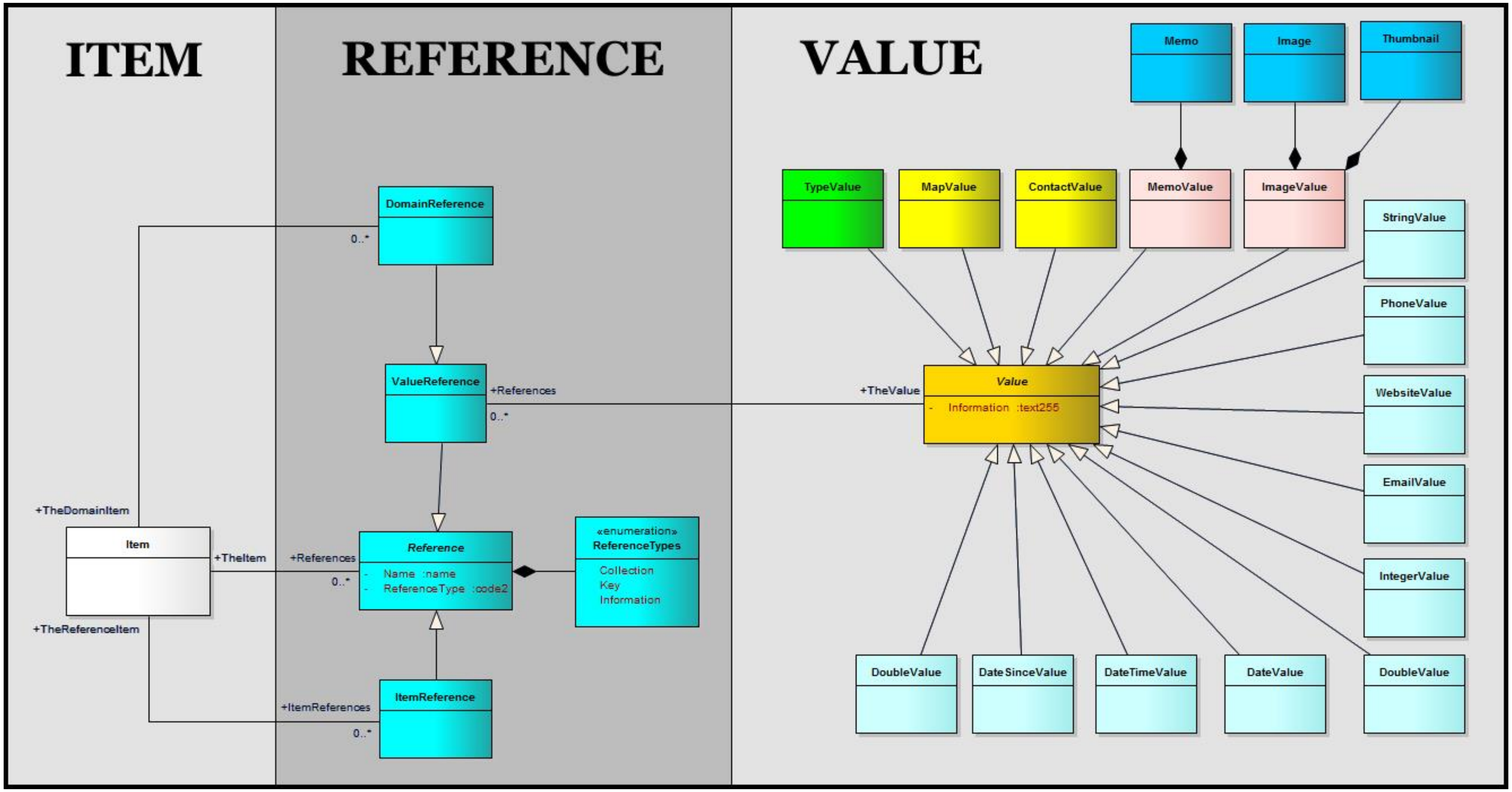
```
_TheValue = new LazyLoadMaster<Value,ValueTab>( aTheValueOID );
```

We don't need to identify the ValueTab's key because we know it's "OID"

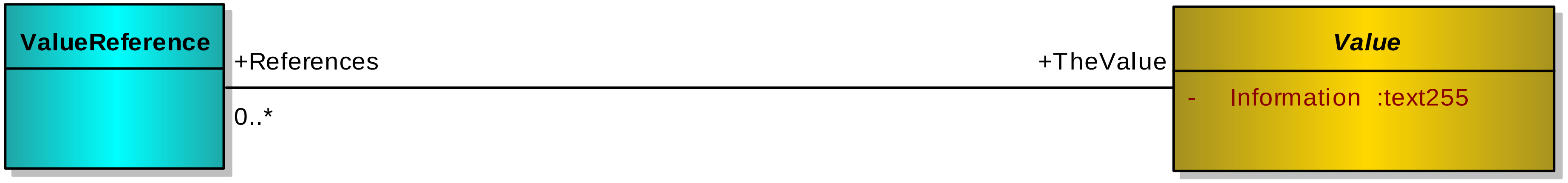
- We are using "Identity Field"

Value Database

Value Database



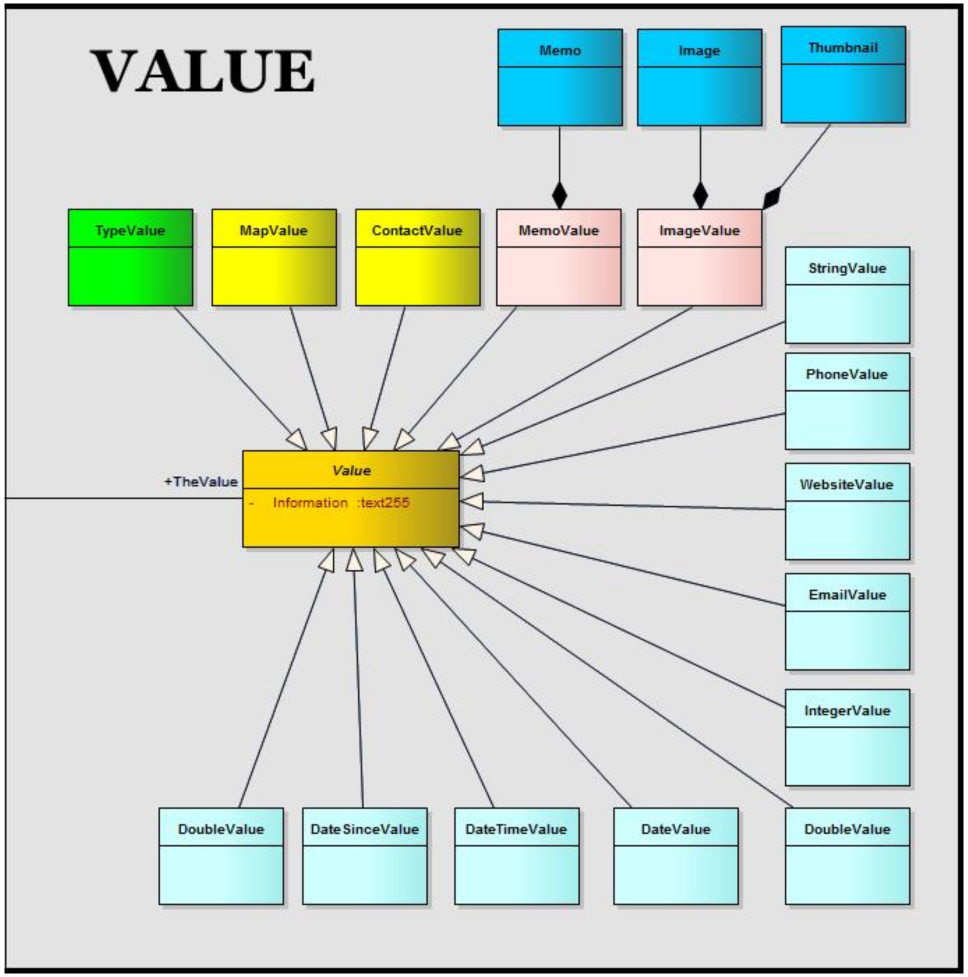
Value Database References



Easy to find all a values references

Value Database …

- Each value is stored only once.
- Why? Makes Search and cross referencing fast and easy.
- Storage is challenging:
  - 0 -> 1 (add, new)
  - 1 -> 2 (add, existing)
  - 1 -> 1 (change, existing)
  - 2 -> 1 (change, new)
  - 1 -> 0 (delete, one)
  - 2 -> 1 (delete, many)



Identity Map

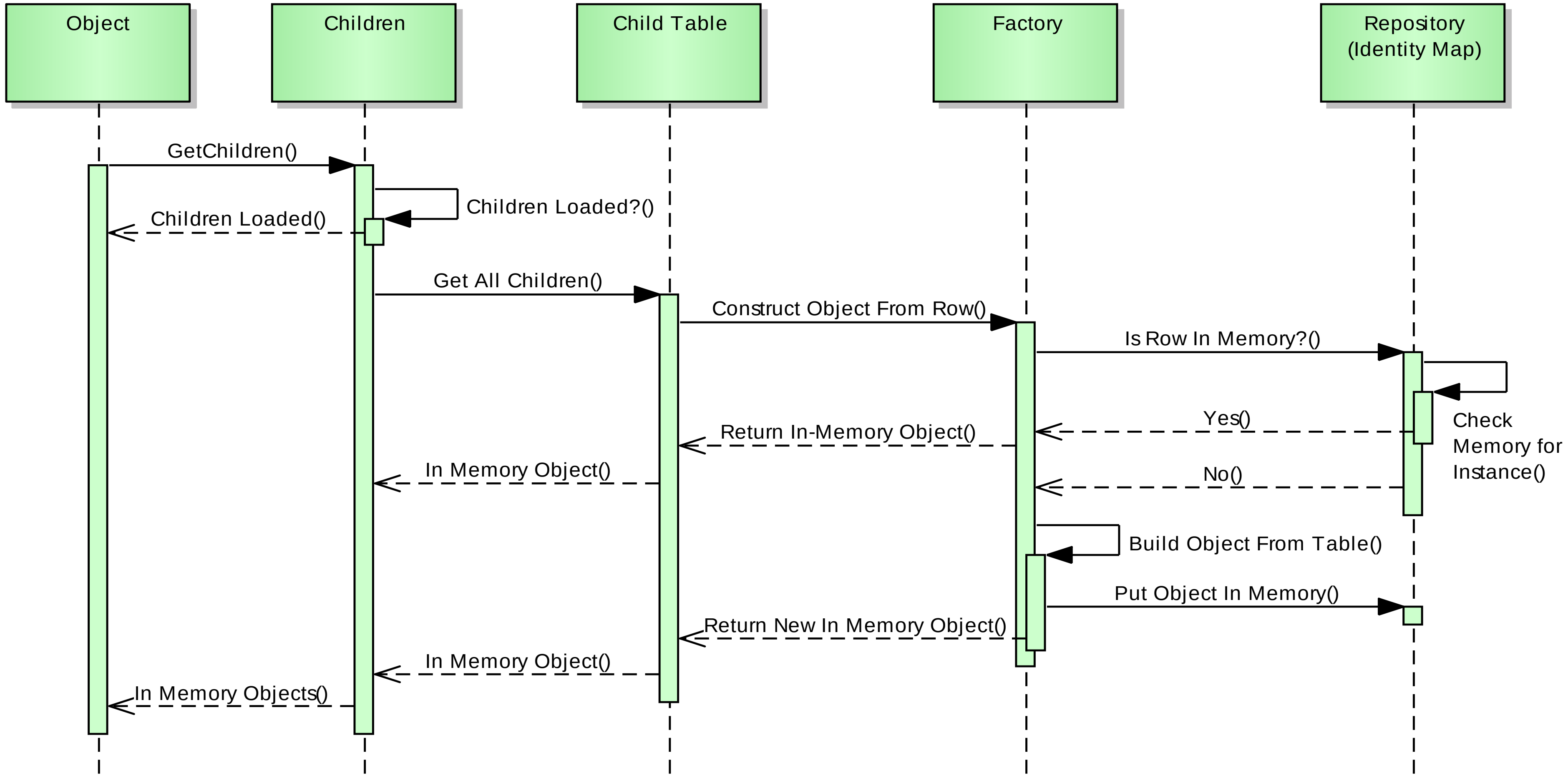
Fowler Enterprise Architecture Pattern

Identity Map

An identity map is essentially a dictionary of in-memory objects

- During the loading of objects, it checks which objects are in memory before loading other objects.
- It’s to ensure that two independent operations on the same objects affect the same in-memory objects without having to save/load objects to/from the database.

Identity Map, with Lazy Load, Factory, Identity Field, and STI



And Finally

The Demos