XAM 300 //

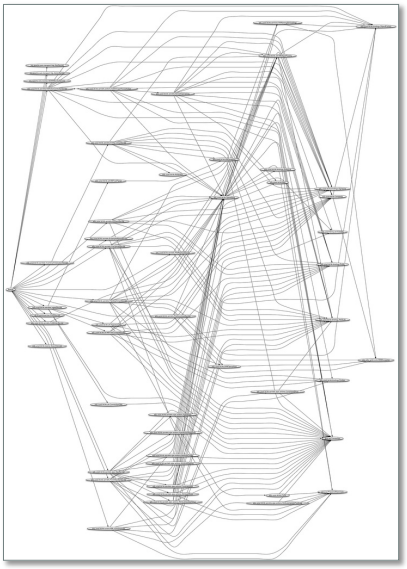
Advanced Cross Platform Mobile Development

- Lecture will begin shortly
- Download class materials from university.xamarin.com

Xamarin University

Objectives

1. Working with Dependencies in a Loosely-Coupled fashion



Xamarin University

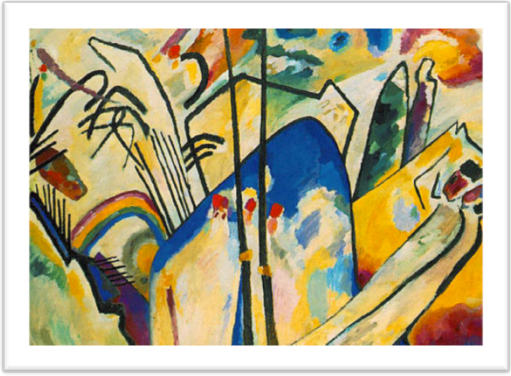
Objective 1

Working with Dependencies in a Loosely-Coupled fashion

Xamarin University

Learning Goals

- Abstractions via Bridge Pattern
- Inversion of Control
- Factory Pattern
- Service Locator
- Dependency Injection
- DI + IoC Containers

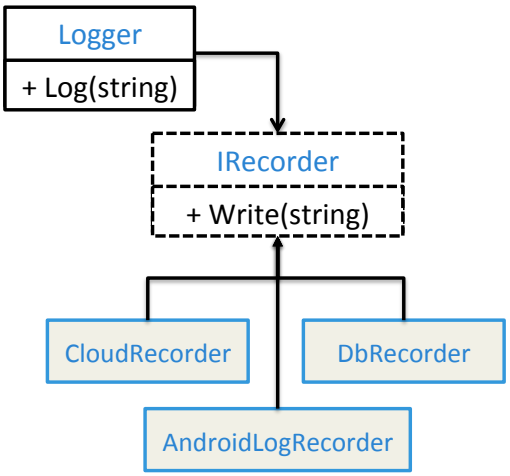


Composition IV, 1911 Kandinsky[1]

Xamarin University

Using Platform Features

- Common problem to require APIs which are platform-specific
  - alerts / notifications
  - file I/O
  - UI marshaling
  - …
- Use *Bridge Pattern* to decouple implementation; this also enables testing



Xamarin University

Example: Alert Service

- Must define abstraction to represent alert across all platforms
- Shared code will use the *abstraction*
- Platform(s) must *implement abstraction*

```
public interface IUIAlertService
{
   bool Show(string title,
             string message,
             string yesButton,
             string noButton);
}
```

Xamarin University

Using Services from our Shared Code

- Once we have abstractions and implementations we need to tie them together

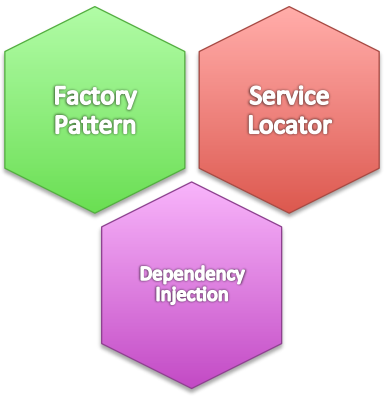
Need to provide the `IUIAlertService` to the class or method

```
public class EmployeeViewModel
{
    ...
    public void FireEmployee()
    {
        IUIAlertService alert = ??;
        if (!alert.Show("Fire Employee?",
              "Are you sure you want to fire " + Name,
              "Yes", "No")) { ... }
    }
}
```

Xamarin University

Locating Services – Inversion of Control

- Several well-known patterns can be used to break dependencies and loosely-couple components together
- These patterns are referred to as "Inversion of Control" (IoC); they allow reusable components to call into platform-specific code (vs. the other way around)
- No one-size-fits-all solution – pick the one(s) that works best with your team and project



Xamarin University

Factory Pattern

- Dependencies can be located through factories which are responsible for creating the abstractions

Xamarin University

Factory Example

Delegate is set by platform which returns implementation of the defined `AlertService`

```
public abstract class AlertService
{
    public static Func<AlertService> Create { get; set; }

    abstract bool Show(string title,
                       string message,
                       string yesButton,
                       string noButton);
}
```

Note: this is just one way to build a Factory, as with any pattern, the implementation can be tailored to the language and platform capabilities

Xamarin University

Factory Pros and Cons

Pros	Cons
▪ Hides the implementation ▪ Easy to use and understand ▪ Can decide implementation at runtime and return specific version based on environment	▪ Requires separate "factory" for each abstraction (possible maintenance issue) ▪ Client must take dependency against factory ▪ Missing dependencies are not known until runtime

Xamarin University

Class Worksheet

- Bridge Pattern
- Inversion of Control principle
- Factory Pattern

Xamarin University

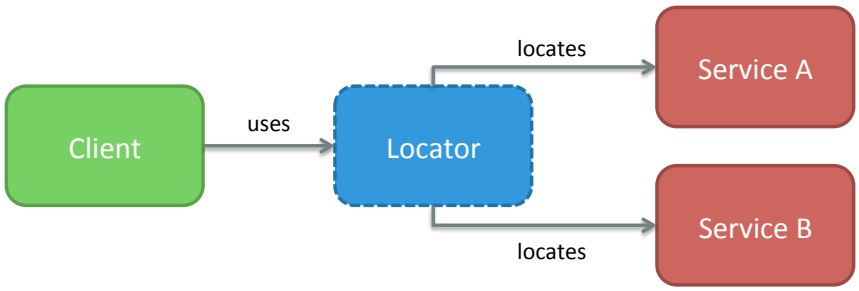
Individual Exercise

Use the Factory Pattern to create a Dependency

Xamarin University

Service Locator

- Service Locator pattern uses a container that maps abstractions (interfaces) to concrete, registered types – client then uses locator to find dependencies



Xamarin University

Service Locator Example Definition

Uses Singleton pattern to provide global accessibility

```
public sealed class ServiceLocator
{
    public static ServiceLocator Instance { get; set; }

    public void Add(Type contractType, object value);
    public void Add(Type contractType, Type serviceType);
    public object Resolve(Type contractType);
    public T Resolve<T>();
}
```

Provide capability to register and locate types

Xamarin University

Registering Dependencies

```
public partial class AppDelegate
{
   ...
   public override void FinishedLaunching(UIApplication application)
   {
      ...
      ServiceLocator.Instance.Add<IUIAlertService,MyAlertService>();
   }
}
```

Platform-specific code *registers* implementation for the abstraction

Xamarin University

Using the Service Locator

```
public class EmployeeViewModel
{
   ...
   public void FireEmployee()
   {
      var alert = ServiceLocator.Instance.Resolve<IUIAlertService>();
      if (!alert.Show("Holiday Bonus",
           "Would you like to give a bonus to " + Name,
           "Yes", "No")) { ... }
   }
}
```

Client then *requests* the abstraction and locator returns the registered implementation

Xamarin University

Service Locator implementations

- Easy to roll your own, but there are many usable implementations out there if you are already using 3rd party libraries
  - Common Service Locator [`commonservicelocator.codeplex.com`]
  - Most Mvvm/Pattern libraries have a Service Locator
  - Xamarin.Forms `DependencyService`

Xamarin University

Service Locator Pros and Cons

Pros	Cons
▪ Easy to use and understand ▪ Clients can JIT-request services ▪ Can be used with any client	▪ Clients must all have access to Locator ▪ Harder to identify dependencies in code ▪ Missing dependencies harder to detect

Xamarin University

Class Worksheet

- Service Locator Pattern

Xamarin University

Group Exercise

Build a Service Locator

Xamarin University

Dependency Injection

- Another option is to have the platform-specific code "inject" the dependency through some form of initialization code; this avoids the dependency against a global service locator

Platform code implements `PlatformServices` class and sets the static `Instance` property to provide access to all the known services

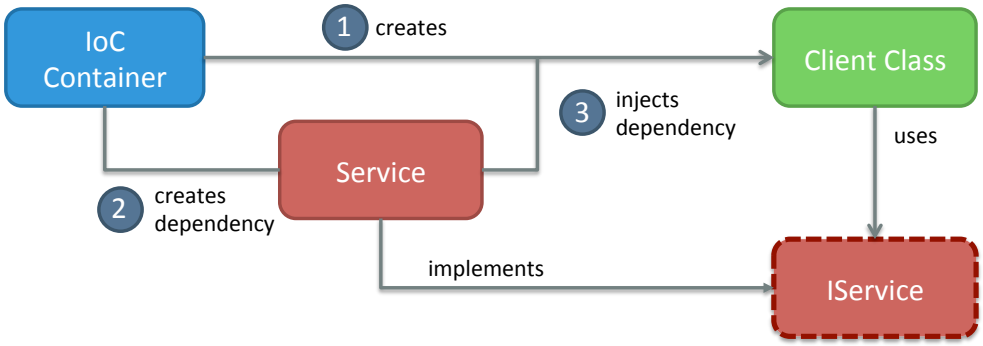
```
public abstract class PlatformServices
{
    public static PlatformServices Instance { get; set;}

    public IUIAlertService AlertService { get; }
    public INavigationService NavigationService { get; }
    ...
}
```

Xamarin University

DI with an IoC Container

- Can automate DI with a *container* that dependencies are registered with which then *create* types – automatically supplying the dependencies



Xamarin University

DI Container Example

```
public class MainViewModel
{
    public MainViewModel(IUIAlertService alertSvc)
    {
      ...
    }
}
```

Alert Service is passed in as a constructor parameter

Container creates MainViewModel, passing the alert service to the constructor automatically

```
container.Register<IUIAlertService,MyAlertService>();
var vm = container.Create<MainViewModel>();
```

Xamarin University

DI + Containers Pros and Cons

Pros	Cons
▪ Created client only needs real dependencies, no container reference necessary ▪ Easy to identify dependencies being used since they are often passed to constructors or filled in properties ▪ Missing dependencies found a little earlier than service locator	▪ Involves a bit of magic (!), the big picture can be harder to understand (what depends on what). ▪ Often requires some form of reflection, not generally a performance issue but could be. ▪ Clients cannot request dependencies once they've been created / initialized

Xamarin University

DI / IoC Containers

- Many containers out there – this is a popular approach
  - TinyIoC
  - Ninject
  - AutoFac
  - Castle Windsor
  - Spring.NET
  - …

Xamarin University

Class Worksheet

- Dependency Injection
- DI/IoC Containers

Xamarin University

Individual Exercise

Using Dependency Injection

Xamarin University

Flash Quiz

① Key to all these patterns is _____.

a) Custom attributes
b) Containers
c) Singletons
d) Abstractions

Xamarin University

Flash Quiz

① Key to all these patterns is _____.

a) Custom attributes
b) Containers
c) Singletons
d) Abstractions

Xamarin University

Flash Quiz

② Service Locator is where _____.

a) Services are found and set into properties on the client
b) Client request specific abstraction through a shared locator
c) Client creates service directly
d) You use *Accio* summoning charm to create the service.

Xamarin University

Flash Quiz

② Service Locator is where _____.

a) Services are found and set into properties on the client
b) Client request specific abstraction through a shared locator
c) Client creates service directly
d) You use *Accio* summoning charm to create the service.

Xamarin University

Flash Quiz

③ To inject dependencies the IoC container will often need to create the type that uses those dependencies

a) True

b) False

Xamarin University

Flash Quiz

③ To inject dependencies the IoC container will often need to create the type that uses those dependencies

a) True

b) False

Xamarin University

Flash Quiz

④ The best technique to manage dependencies is _____.

a) Factory Pattern
b) Service Locator Pattern
c) Dependency Injection
d) Depends on the project, team, and personal preference.

Xamarin University

Flash Quiz

④ The best technique to manage dependencies is _____.

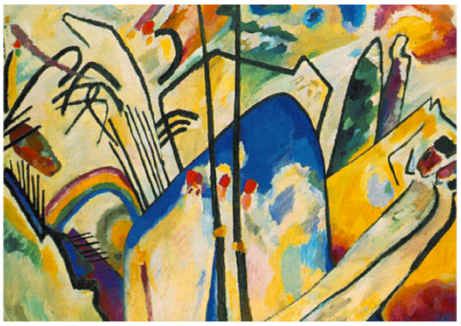
a) Factory Pattern
b) Service Locator Pattern
c) Dependency Injection
d) It depends on the project, team, and personal preference.

Xamarin University

Summary

- Abstractions via Bridge Pattern
- Inversion of Control
- Factory Pattern
- Service Locator
- Dependency Injection
- DI + IoC Containers

QUESTIONS?



Composition IV, 1911 Kandinsky[1]

Xamarin University

Xamarin University

XAM300 – Advanced Cross Platform Mobile Development

Thank You

Please complete the class survey in your profile:
university.xamarin.com/profile