Interfaces

Simon Robinson
Software Developer

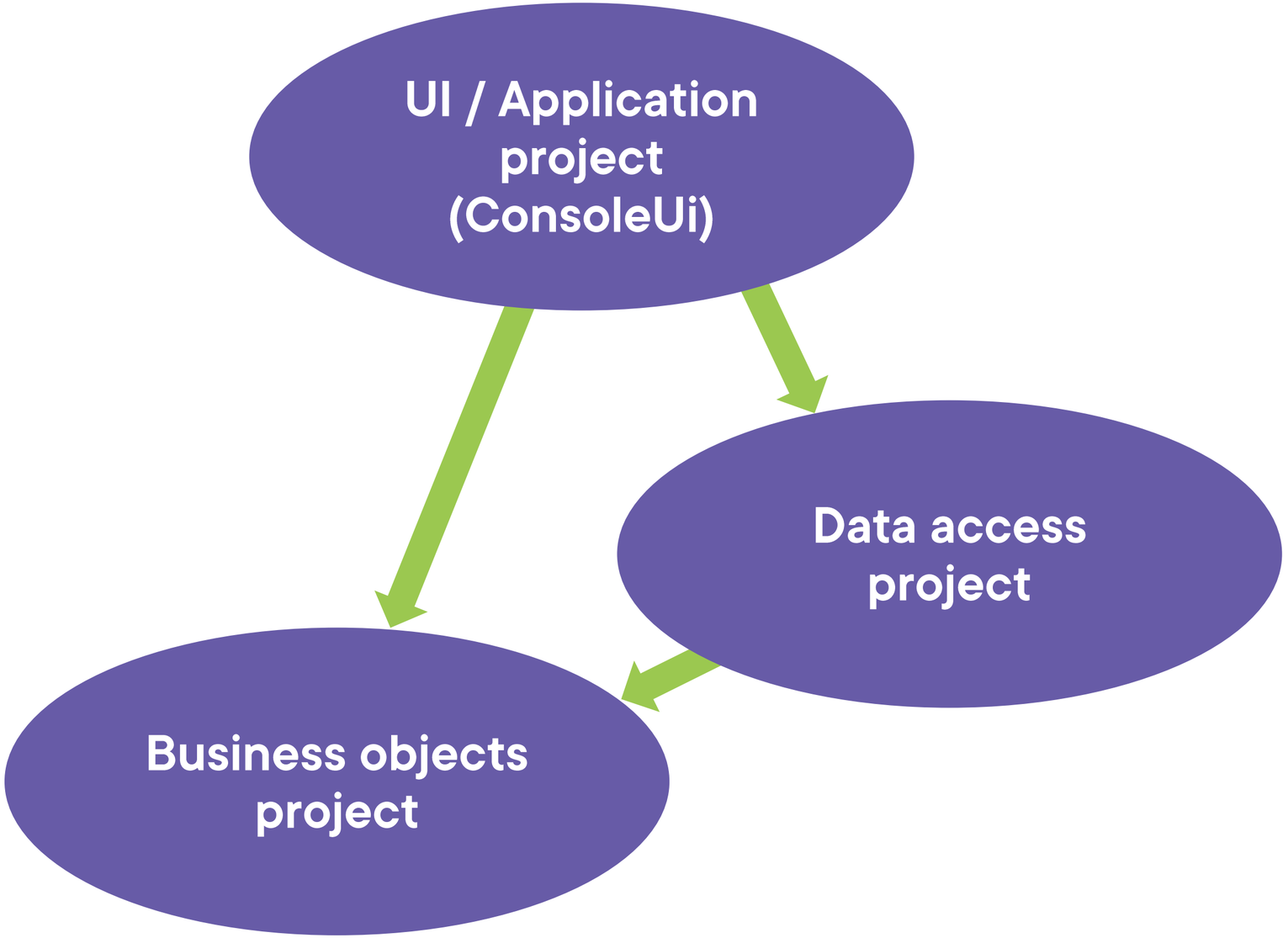
@TechieSimon www.SimonRobinson.com

Overview

Solving Problems with Interfaces

- Enable dependency-injection
  - Interfaces are essential
- Hide some interface methods
  - Why do that?
- Extend interfaces over time

Architecture for the Demo



Client shopping carts
- Garden product store

The business objects have no other dependencies

Interfaces will enable a clean architecture

Demo

Setting up the demo

- Focus on a business object that represents clients
- Get ready for dependency injection

Interfaces and Dependency Injection

There’s a problem!

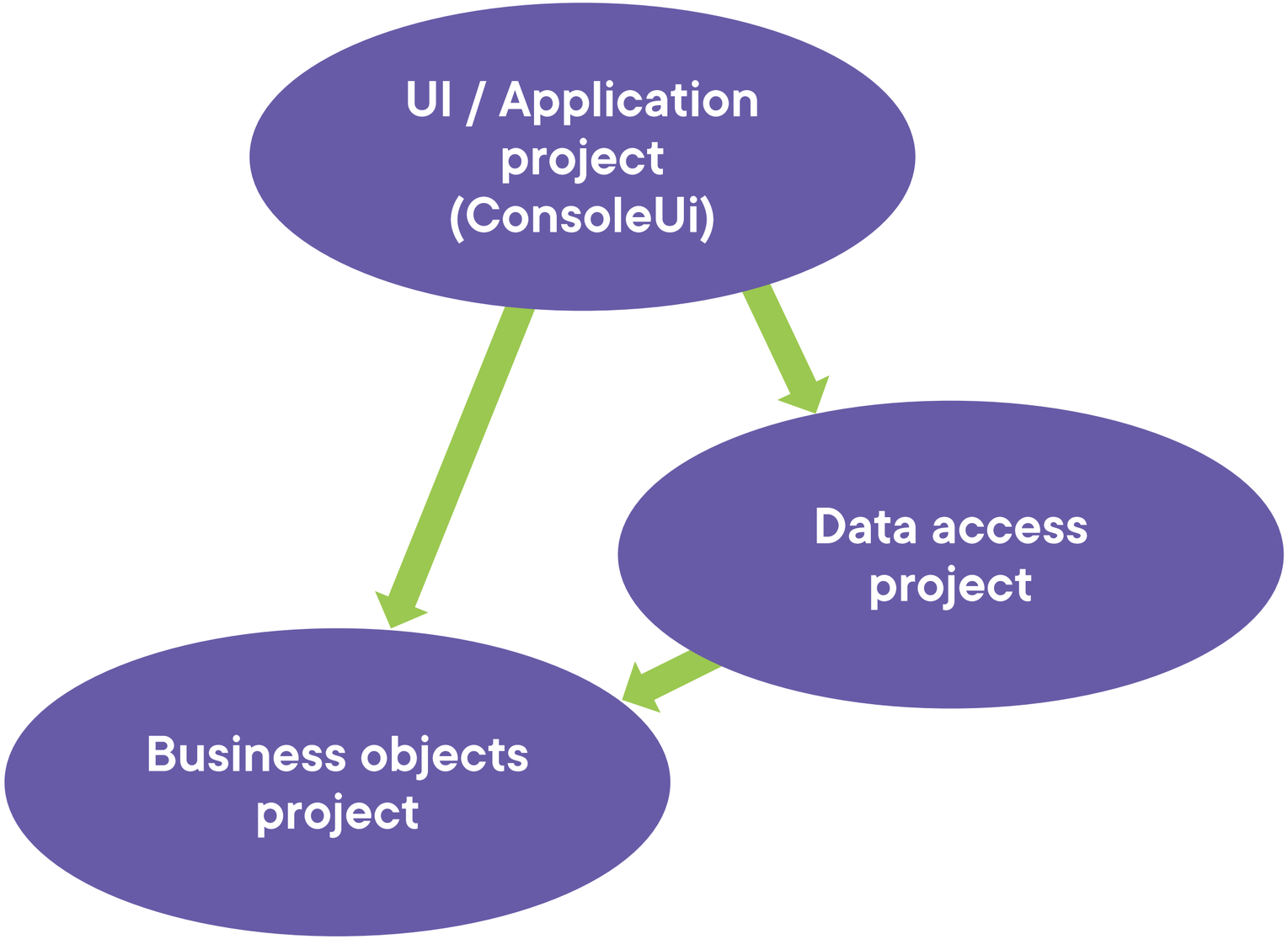
- Shopping carts aren’t being saved
- So clients who return later are losing what they added to the carts

Demo

Add ability to save carts

- Keep good architecture
- Interfaces will be essential!

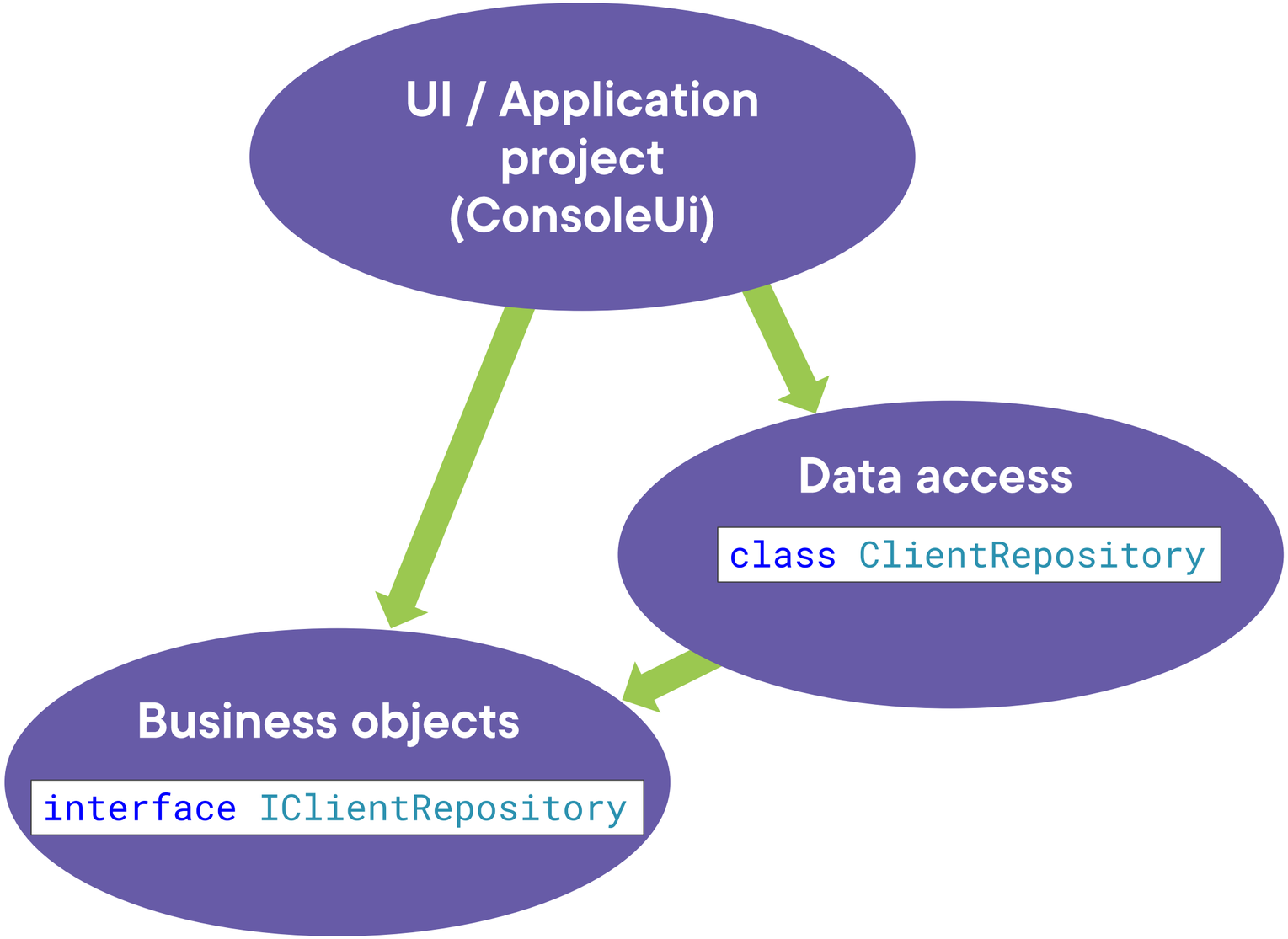
Architecture for the Demo



Business objects should not know about the external database

But data access types usually do need to know about the business objects

Architecture for the Demo

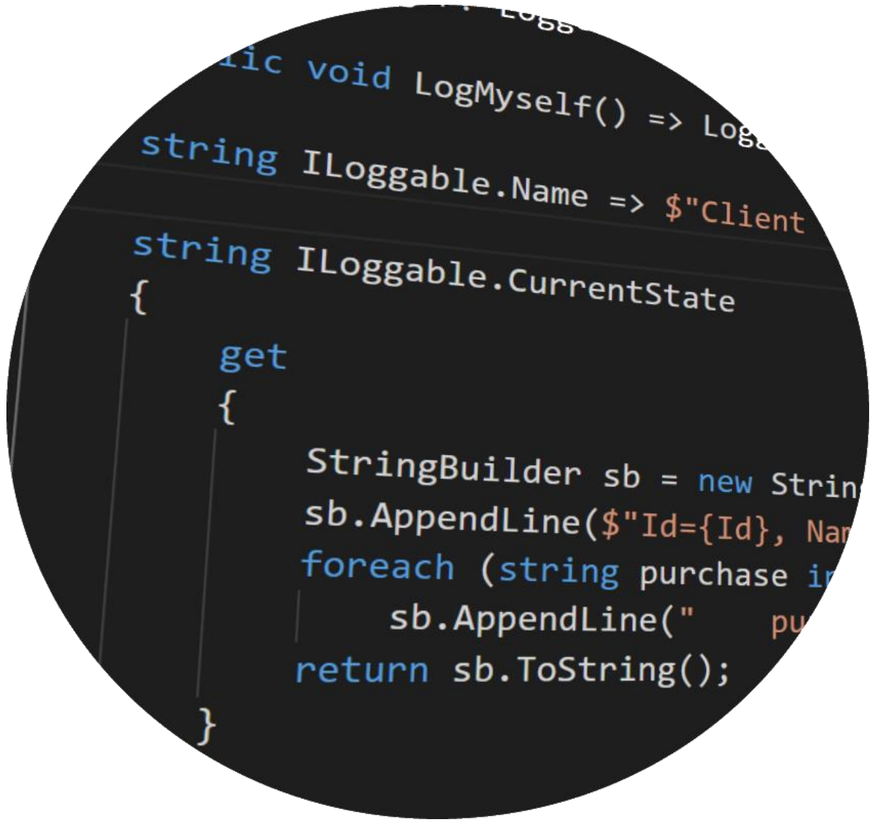


Define interfaces
in the lowest level project
where they
might be required

Even if concrete classes
must be defined
further up

Then do
dependency injection
with the interfaces

Hiding Some Interface Methods



How do you hide interface members?

- Implement them explicitly

But why?

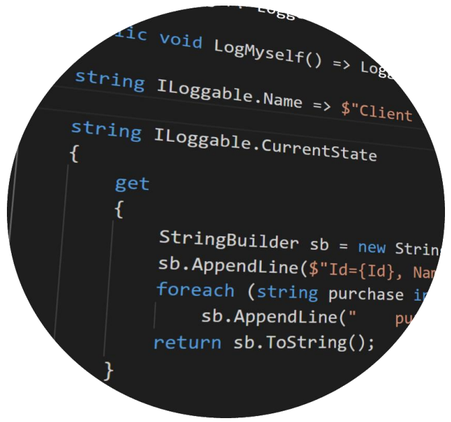
- Stay tuned...

Demo

New requirement: Diagnostic logging

- Two loggers
  - A file logger
  - A console logger
- The task: Implement these
  - Explicit interface members will be necessary

Explicit Interface Implementation




```
string ILoggable.CurrentState
```

- ✔ **Avoids name clashes**
- ✔ **Avoids publicizing members not related to core purpose of type**

Extend an Interface over Time

Demo

New requirement:

- Logging state of an object is no longer enough
- You must be able to log method invocations
- You must add this ability to the logging infrastructure

Default Interface Member Implementation

```
public interface ILogger
{
    void LogMethodCall(ILoggable source, string methodName) { // etc.
```

Basic implementation of a member

Covers if any interface implementers haven't implemented this member

Not necessarily ideal – just a fallback

Summary

Dependency injection

- Interfaces enable DI
- Even when you'd otherwise run against project dependencies

Member name clashes

- Solve with explicit member implementation
- Also useful to hide members not relevant to the concrete type's purpose

To extend an interface

- Provide default implementations