Interfaces and Dynamic Loading

The "Why" (Part 2)

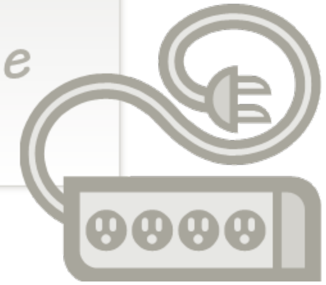
Jeremy Clark
www.jeremybytes.com
jeremy@jeremybytes.com

pluralsight
hardcore developer training

Why Interfaces?

Maintainable

Extensible



Easily Testable

Interfaces help us get there

Best Practice

Program to an abstraction rather than a concrete type

Best Practice

Program to an interface
rather than a concrete class

Program to an Interface

```
private void FetchData(string repositoryType)
{
    ClearListBox();

    IPersonRepository repository =
        RepositoryFactory.GetRepository(repositoryType);
    var people = repository.GetPeople();
    foreach (var person in people)
        PersonListBox.Items.Add(person);

    ShowRepositoryType(repository);
}
```

No Reference to Concrete Types

Compile-Time Factory

```
public static class RepositoryFactory
{
    public static IPersonRepository GetRepository(
      string repositoryType)
    {
      IPersonRepository repo = null;
      switch (repositoryType)
      {
          case "Service": repo = new ServiceRepository();
              break;
          case "CSV": repo = new CSVRepository();
              break;
          case "SQL": repo = new SQLRepository();
              break;
          default:
              throw new ArgumentException("Invalid Repository Type");
      }
      return repo;
    }
}
```

Factory Comparison

Compile-Time Factory

- **Has a Parameter**
  - The caller decides which repository to use
- **Compile-Time Binding**
  - Factory needs references to repository assemblies

Dynamic Factory

- **No Parameter**
  - The factory returns a repository based on configuration
- **Run-Time Binding**
  - Factory has no compile-time references to repository assemblies

Dynamic Loading

- Get Type and Assembly from Configuration
- Load Assembly through Reflection
- Create a Repository Instance with the Activator

```
public static class RepositoryFactory
{
  public static IPersonRepository GetRepository()
  {
    string typeName =
      ConfigurationManager.AppSettings["RepositoryType"];
    Type repoType = Type.GetType(typeName);
    object repoInstance = Activator.CreateInstance(repoType);
    IPersonRepository repo = repoInstance as IPersonRepository;
    return repo;
  }
}
```

Unit Testing

- **Testing small pieces of code**
  - Usually on the method level

- **Testing in isolation**
  - Eliminate outside interactions that might break the test
  - Reduce the number of objects needed to run the test

- **Note: We still need Integration Testing**
  - Testing that the pieces all work together

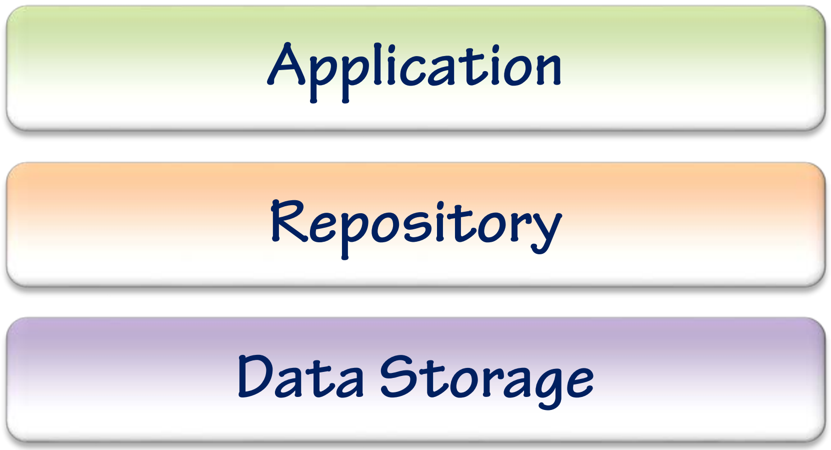
What We Want to Test

```
public partial class MainWindow : Window
{
  private void FetchButton_Click(object sender, RoutedEventArgs e)
  {
    ClearListBox();

    IPersonRepository repository = RepositoryFactory.GetRepository();

    var people = repository.GetPeople();
    foreach (var person in people)
      PersonListBox.Items.Add(person);

    ShowRepositoryType(repository);
  }

  public MainWindow()...
  private void ClearButton_Click(object sender, RoutedEventArgs e)...
  private void ClearListBox()...
  private void ShowRepositoryType(IPersonRepository repository)...
}
```

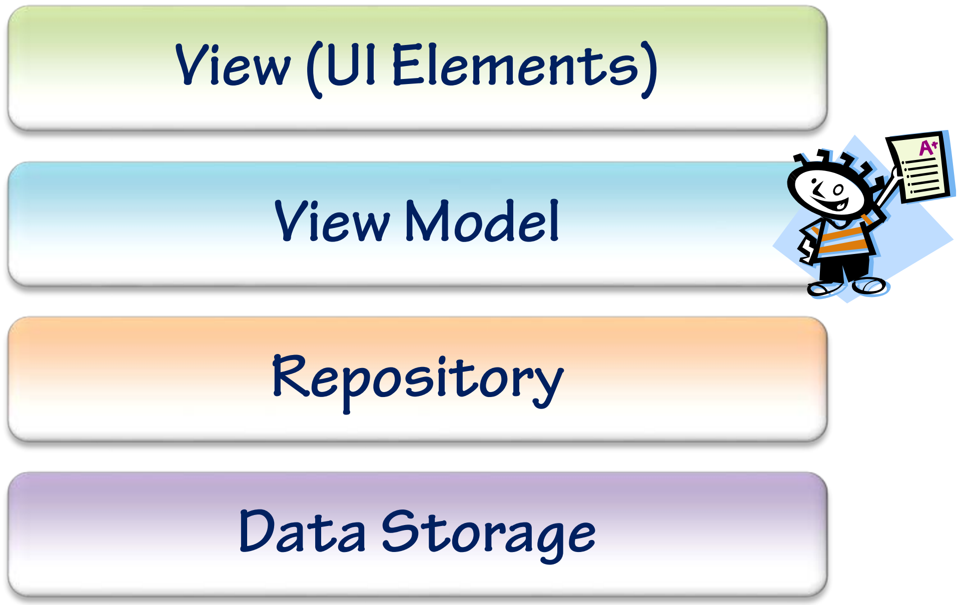
Dependent Objects

```
public partial class MainWindow : Window
{
  private void FetchButton_Click(object sender, RoutedEventArgs e)
  {
    ClearListBox();

    IPersonRepository repository = RepositoryFactory.GetRepository();

    var people = repository.GetPeople();
    foreach (var person in people)
      PersonListBox.Items.Add(person);

    ShowRepositoryType(repository);
  }

  public MainWindow()...
  private void ClearButton_Click(object sender, RoutedEventArgs e)...
  private void ClearListBox()...
  private void ShowRepositoryType(IPersonRepository repository)...
}
```

Additional Layering



Additional Layering



Very Simple MVVM Implementation

Isolating Code

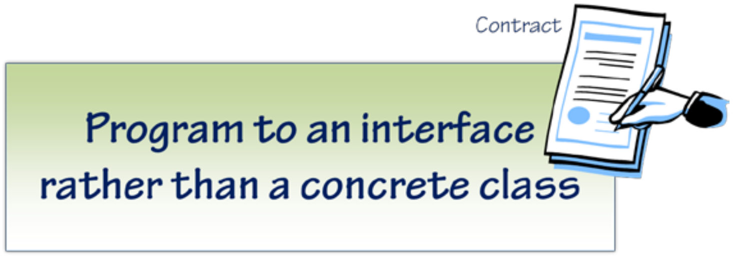
- **Move Functionality to a View Model**
  - Eliminates dependency on UI objects
- **Add a Fake Repository**
  - Eliminates dependency on network, file system, or SQL database
  - Ensures consistent behavior

Remember: We are not testing the Repository here.

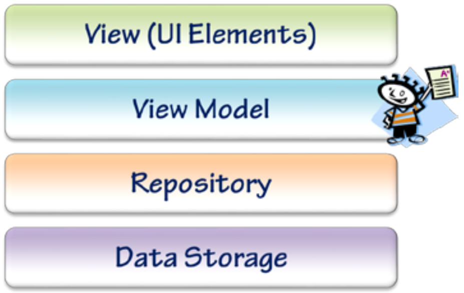
We are testing the “Fetch Data” functionality in our application code.

Summary

- **Program to an Interface only**



- **Dynamic Loading / Late Binding**
- **Unit Testing**
  - Application Layering
  - Fake Repository



- **Next up: Where to go Next**