Interfaces in Frameworks and Patterns

Jeremy Clark
DEVELOPER BETTERER

@jeremybytes www.jeremybytes.com

Where

Dependency injection

Design patterns

- Repository
- Factory method
- Decorator

Mocking

Dependency Injection (DI)

A set of software design principles and patterns that enable us to develop loosely coupled code.

van Deursen and Seeman. *Dependency Injection in .NET*. Manning, 2018.

Interfaces help us create loose coupling.

```
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);
}
```

Delegating Details

No references to concrete repository types

"Seam" allows easy swapping of repositories

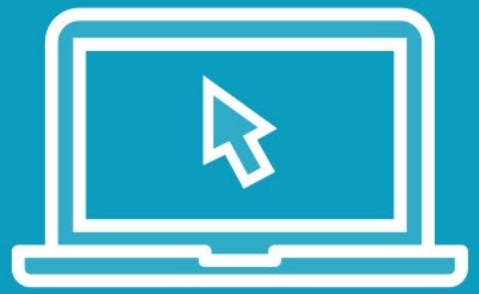
Getting a Dependency

```
public class PeopleViewModel : INotifyPropertyChanged
{
    private IPersonRepository repository;

    public PeopleViewModel()
    {
        repository = RepositoryFactory.GetRepository();
    }

    public void FetchData()
    {
        People = repository.GetPeople();
    }
    ...
}
```

Injecting a Dependency

```
public class PeopleViewModel : INotifyPropertyChanged
{
    private IPersonRepository repository;

    public PeopleViewModel(IPersonRepository injectedRepo)
    {
        repository = injectedRepo;
    }

    public void FetchData()
    {
        People = repository.GetPeople();
    }
    ...
}
```

Demo



Injecting a repository

- Manual construction
- Dependency injection container

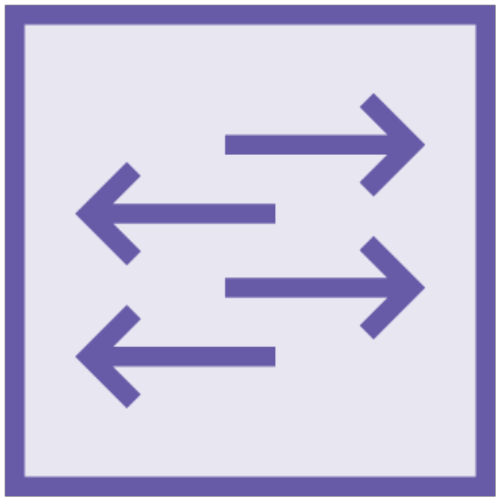
Unit tests with DI

Interfaces help with
implementing design patterns.

Repository Pattern

Separates our application from the data storage technology

Application



Repository

Data store

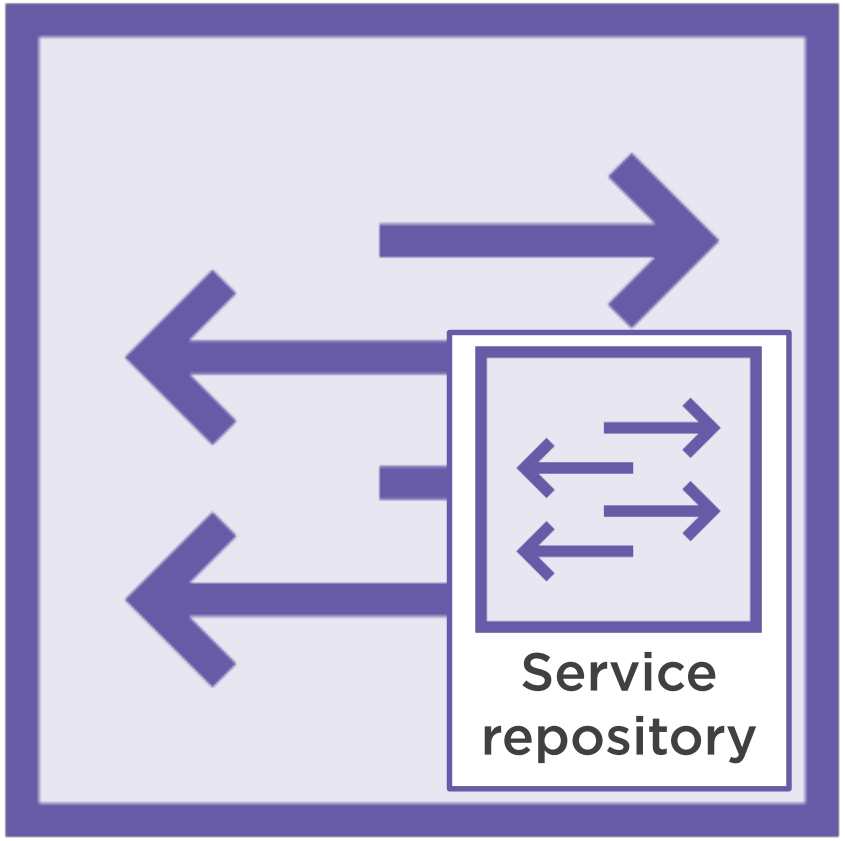
Factory Method Pattern

```
IPersonRepository GetRepository(string repositoryType) {

  IPersonRepository repository = null;

  switch (repositoryType) {
    case "Service": repository = new ServiceRepository();
      break;
    case "CSV": repository = new CSVRepository();
      break;
    case "SQL": repository = new SQLRepository();
      break;
  }
  return repository;
}
```

Decorator

Wrap an existing interface to add functionality

Repository Decorator

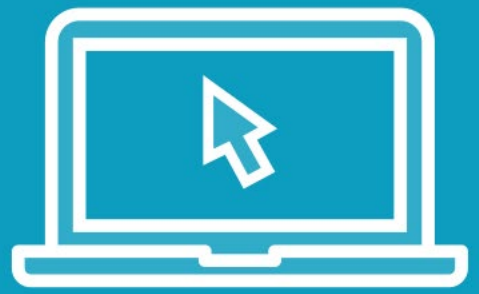


Caching repository

{JSON}

Web service

Demo



Caching decorator

Mocking

Creating an in-memory object for testing purposes

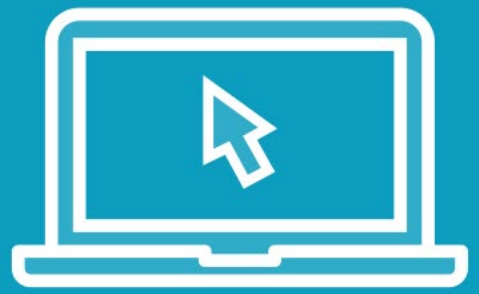
Fake Repository

```
public class FakeRepository : IPersonRepository
{
    public IEnumerable<Person> GetPeople()
    {
        var people = new List<Person>() {...};
        return people;
    }

    public Person GetPerson(int id)
    {
        var people = GetPeople();
        return people.FirstOrDefault(p => p.Id == id);
    }
}
```

In-Memory Repository with MOQ

```
private IPersonRepository GetMockRepository()
{
    var testPeople = new List<Person>() {...};

    var mockRepo = new Mock<IPersonRepository>();

    mockRepo.Setup(m => m.GetPeople()).Returns(testPeople);

    return mockRepo.Object;

}
```

Demo



Test with mock repository

Where

Dependency injection

Design patterns

- Repository
- Factory method
- Decorator

Mocking