Coming Up

A few words on generics

Working with generics and reflection

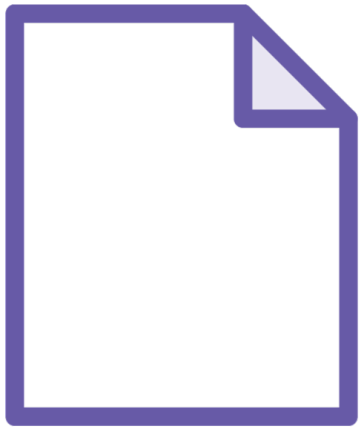
- **Inspecting generic instances**
- **Instantiating generic types**
- **Calling generic methods**

Building an IoC container

Generics

Generics let you tailor a method, class, structure or interface to the precise data type it acts upon

Generics



HashTable

Key-value pairs of any type

Dictionary<TKey,TValue>

Typed key-value pairs

Advantages of Generics

Type safety: compiler will throw an error on unsafe cast

Reusability: one generic class can be used on a variety of types

Improved performance

Boxing

The process of converting a type to an object

Unboxing

The process of converting an object to a type

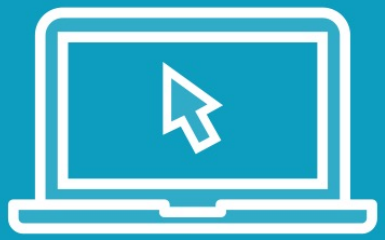
Advantages of Generics

Type safety: compiler will throw an error on unsafe cast

Reusability: one generic class can be used on a variety of types

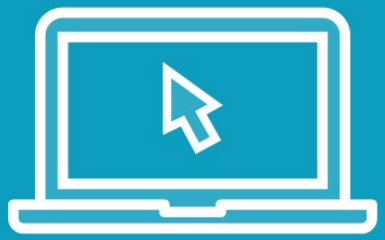
Improved performance: (un)boxing is avoided

Demo



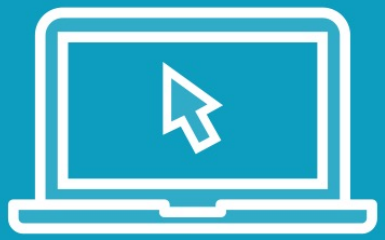
Inspecting generic instances

Demo



Creating generic instances

Demo



Invoking generic methods

Building an IoC Container

An inversion of control container extensively uses reflection & generics

- **Perfect demo example ☺**

```
public class IoCContainer
{
    public void Register<TContract, TImplementation>(){ … }
    public void Register(Type contract, Type implementation){ … }
    public TContract Resolve<TContract>() { … }
    private TContract Create<TContract>(Type implementationType) { … }
}
```

Building an IoC Container

```
public class IoCContainer
{
    public void Register<TContract, TImplementation>(){ … }
    public void Register(Type contract, Type implementation){ … }
    public TContract Resolve<TContract>() { … }
    private TContract Create<TContract>(Type implementationType) { … }
}
```

Building an IoC Container

```
public class IoCContainer
{
    public void Register<TContract, TImplementation>(){ … }
    public void Register(Type contract, Type implementation){ … }
    public TContract Resolve<TContract>() { … }
    private TContract Create<TContract>(Type implementationType) { … }
}
```

Building an IoC Container

```
public class IoCContainer
{
    public void Register<TContract, TImplementation>(){ … }
    public void Register(Type contract, Type implementation){ … }
    public TContract Resolve<TContract>() { … }
    private TContract Create<TContract>(Type implementationType) { … }
}
```

Building an IoC Container

```
public class IoCContainer
{
    public void Register<TContract, TImplementation>(){ … }
    public void Register(Type contract, Type implementation){ … }
    public TContract Resolve<TContract>() { … }
    private TContract Create<TContract>(Type implementationType) { … }
}
```

Building an IoC Container

Support constructor injection

Support generics

Building an IoC Container - Demo Scenario

```
public class CoffeeService : ICoffeeService {

    public CoffeeService(IWaterService waterService,
                         IBeanService<Catimor> beanService) { … }

}

public interface ICoffeeService { … }

public class TapWaterService : IWaterService { … }

public interface IWaterService { … }

public class ArabicaBeanService<T> : IBeanService<T> { … }

public interface IBeanService<T> { … }

public class Catimor { … }
```

Building an IoC Container - Demo Scenario

```
public class CoffeeService : ICoffeeService {

    public CoffeeService(IWaterService waterService,
                         IBeanService<Catimor> beanService) { … }

}

public interface ICoffeeService { … }

public class TapWaterService : IWaterService { … }

public interface IWaterService { … }

public class ArabicaBeanService<T> : IBeanService<T> { … }

public interface IBeanService<T> { … }

public class Catimor { … }
```

Building an IoC Container - Demo Scenario

```
public class CoffeeService : ICoffeeService {

    public CoffeeService(IWaterService waterService,
                         IBeanService<Catimor> beanService) { … }

}

public interface ICoffeeService { … }

public class TapWaterService : IWaterService { … }

public interface IWaterService { … }

public class ArabicaBeanService<T> : IBeanService<T> { … }

public interface IBeanService<T> { … }

public class Catimor { … }
```

Building an IoC Container - Demo Scenario

```
public class CoffeeService : ICoffeeService {

    public CoffeeService(IWaterService waterService,
                         IBeanService<Catimor> beanService) { … }

}

public interface ICoffeeService { … }

public class TapWaterService : IWaterService { … }

public interface IWaterService { … }

public class ArabicaBeanService<T> : IBeanService<T> { … }

public interface IBeanService<T> { … }

public class Catimor { … }
```

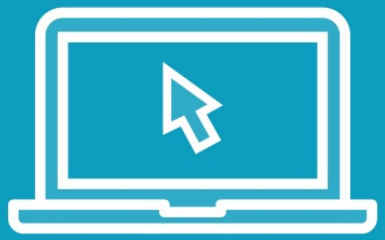
Building an IoC Container - Demo Scenario

```
public class CoffeeService : ICoffeeService {

    public CoffeeService(IWaterService waterService,
                         IBeanService<Catimor> beanService) { … }

}

public interface ICoffeeService { … }

public class TapWaterService : IWaterService { … }

public interface IWaterService { … }

public class ArabicaBeanService<T> : IBeanService<T> { … }

public interface IBeanService<T> { … }

public class Catimor { … }
```

Building an IoC Container - Demo Scenario

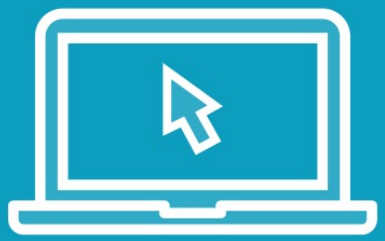
```
public class CoffeeService : ICoffeeService {

    public CoffeeService(IWaterService waterService,
                         IBeanService<Catimor> beanService) { … }

}

public interface ICoffeeService { … }

public class TapWaterService : IWaterService { … }

public interface IWaterService { … }

public class ArabicaBeanService<T> : IBeanService<T> { … }

public interface IBeanService<T> { … }

public class Catimor { … }
```

Demo



Building an IoC container – the basics

Demo



Building an IoC container – supporting constructor injection and unbound generics

Summary

Advantages of generics

- **Type safety**
- **Reusability**
- **Improved performance**

Summary

Creating generic instances

- **MakeGenericType**

Invoking generic methods

- **MakeGenericMethod**