Coding Dojo 2016-07-15: Dependency Injection

The product owner of your current project has realised the benefits of using IoC via Dependency Injection, but the framework you are currently using wasn't developed in-house so it is time to build your own IoC container in C# named "TinyIoC" (or Java if you prefer – adapt function calls accordingly). Here are the User Stories he gave you:

1. I want to register a concrete instance of an interface as singleton and be able to resolve it by providing only the interface, e.g.:

   ```
   IoC.RegisterAsSingleton<Interface>(instance);
   IoC.Resolve<Interface>();
   ```

   Details: Resolving an interface that hasn't been registered should throw an exception. Registering an interface multiple times overwrites previous registrations (only last one is valid)

2. I want to register a class as singleton provider and be able to resolve it by providing only the interface, e.g.:

   ```
   IoC.ConstructAndRegisterSingleton<Interface, Implementation>();
   IoC.Resolve<Interface>();
   ```

   Assume that the implementation has only a single constructor that takes no arguments. Ensure that the resolved interface is a singleton.

3. I want to register a class as type provider for an interface and be able to resolve it by providing only the interface.

   ```
   IoC.RegisterType<Interface, Implementation>();
   IoC.Resolve<Interface>();
   ```

   Assume that the implementation has only a single constructor that takes no arguments. Ensure that the resolved interface is in fact a new instance every time it is resolved.

4. I want to be able to register singleton and type providers that have dependencies to other interfaces and have them be resolved lazily when they are resolved. Their dependencies are injected via the constructor (see page 2 for example interfaces and implementations):

   ```
   IoC.LazyConstructAndRegisterSingleton<Interface, Implementation>();
   IoC.Resolve<Interface>();
   ```

   Lazy construction means Resolve builds the instance when first called and returns this exact instance in this and every successive call. Assume that there are no cyclic dependencies. If an implementation has multiple constructors use the one that has the most arguments, that can be resolved.

5. I want to have type registration (3) and singleton providers (2) extended to be able to resolve dependent interfaces too.

   Assume that there are no cyclic dependencies. If an implementation has multiple constructors use the one that has the most arguments, that can be resolved.

6. I want to have all registrations (2, 3, 4) be able to detect cyclic dependencies and throw an exception when they are resolved (3,4)/registered (2).

Example dependent interfaces and implementations:

```
public interface ITestInterface2
{
    int TestMethod();
}

public interface ITestInterface
{
    int ExampleMethod();
}

public class TestDependencyImplementation : ITestInterface
{
    public int ExampleMethod()
    {
        return 7;
    }
}

public class TestImplementationWithDependencies : ITestInterface2
{
    private readonly ITestInterface _testInterface;

    public TestImplementationWithDependencies(ITestInterface testInterface)
    {
        _testInterface = testInterface;
    }

    public int TestMethod()
    {
        return _testInterface.ExampleMethod();
    }

}
```