

# Container Dependency Injection

Adrian Mularczyk

# Description of the problem

# Dependency injection

# Dependency injection

```
public class Foo {  
  
    public void DoSomeWork() {  
        Bar bar = new Bar();  
        bar.DoSomething();  
    }  
  
}
```

# Dependency injection

```
public class Foo {  
  
    public void DoSomeWork() {  
        Bar bar = new Bar();  
        bar.DoSomething();  
    }  
  
}
```

# Dependency injection

```
public class Foo {  
  
    public void DoSomeWork(Bar bar) {  
        bar.DoSomething();  
    }  
  
}
```

# Dependency injection

```
public class Foo {  
  
    Bar _bar;  
  
    public Foo(Bar bar) {  
        _bar = bar;  
    }  
  
    public void DoSomeWork() {  
        _bar.DoSomething();  
    }  
  
}
```

# Dependency injection



# Dependency injection kinds

# Dependency injection kinds

- Injecting by the constructor
- Injecting by the method
- Injecting by the property

# Dependency injection kinds

- Injecting by the constructor
- Injecting by the method
- Injecting by the property

# Container of dependency injection

# Basic operations

- Register,
- Resolve.

# Registration types

- Transient,
- Singleton,
- Scope (Thread, HttpRequest),
- Factory Method.

# Container

```
public class Container {  
  
    private List<Type> RegisteredTypes;  
  
    public Container() {  
        RegisteredTypes = new List<Type>();  
    }  
  
    ...  
}
```

# Register

```
public void Register(Type type) {  
    RegisteredTypes.Add(type);  
}
```



# Resolve

```
public object Resolve(Type type) {  
    if(!RegisteredTypes.Contains(type)) {  
        throw new TypeNotRegisteredException();  
    }  
  
    return CreateObjectOfType(type);  
}
```

# RegisteredTypes

```
public class Container {  
  
    private List<RegisteredType>  
    RegisteredTypes;  
  
    public Container() {  
        RegisteredTypes = new  
List<RegisteredType>();  
    }  
  
    ...  
}
```

# RegisteredType

```
public class RegisteredType {  
    public Type InputType;  
    public Type OutputType;  
    public RegistrationKind RegistrationKind;  
    public object Value;  
    public List<Scope, object> Scopes;  
    ...  
}
```

# Register

```
public void Register(  
    Type inputType, Type outputType) {  
  
    RegisteredTypes.Add(  
        new RegisteredType {  
            InputType = inputType,  
            OutputType = outputType  
        }  
    );  
}
```

# Register

```
public void RegisterTransient(  
    Type inputType, Type outputType) {  
  
    RegisteredTypes.Add(  
        new RegisteredType {  
            InputType = inputType,  
            OutputType = outputType,  
            RegistrationKind =  
RegistrationKind.Transient  
        }  
    );  
}
```

# Resolve

```
public object Resolve(Type type) {  
    if(!RegisteredTypes.Contains(type)) {  
        throw new TypeNotRegisteredException();  
    }  
  
    return CreateObjectOfType(type);  
}
```

# Summary

## Questions?

Thank you!