

Module 03:

“Dependency Injection”



Agenda

- ▶ **Classifying Dependencies**
- ▶ Dependency Injection Containers
- ▶ Summary



Volatile Dependencies

- ▶ Out-of-process or unmanaged resources
- ▶ Nondeterministic resources
- ▶ Resources to be
 - Replaced
 - Intercepted
 - Decorated
 - Mocked



Examples of Volatile Dependencies

- ▶ Databases
- ▶ File system
- ▶ Web services
- ▶ Security contexts
- ▶ Message Queues
- ▶ **Hệ thống Ràng buộc** (or similar)



Stable Dependencies

- ▶ A dependency is *stable* if it's not volatile...!

ințelesgăce ÎușesRôlêRăsșes

çööl Răsșe șțsîng sôlê

çlășș Dîșr'lăyșțățê

řučl'ic șțsîng Năne gêtj

řučl'ic șțsîng Dîșr'lăyTêyțj gêtj

řučl'ic Dîșr'lăyșțățê NôwîêDțô nôwîê

Discussion Point:

Which type of dependency is **Computation**?

*Dependency Injection applies exclusively
to Volatile Dependencies.*

Don't inject Stable Dependencies!

Agenda

- ▶ Classifying Dependencies
- ▶ **Dependency Injection Containers**
- ▶ Summary



Pattern: Pure DI (a.k.a. Poor Man's DI)

- ▶ *Pure DI is the practice of applying DI without a DI Container.*
- ▶ Outline
 - This is essentially what we have been doing throughout the workshop
 - Compose object graphs manually at Composition Root
- ▶ See:
"Dependency Injection Principles, Practices, and Patterns"
Steven van Deursen and Mark Seemann (2019)



Dependency Injection Containers

Autofac

Simple Injector

Microsoft.Extensions.DependencyInjection

Unity

Ninject

Castle Windsor

Spring.NET

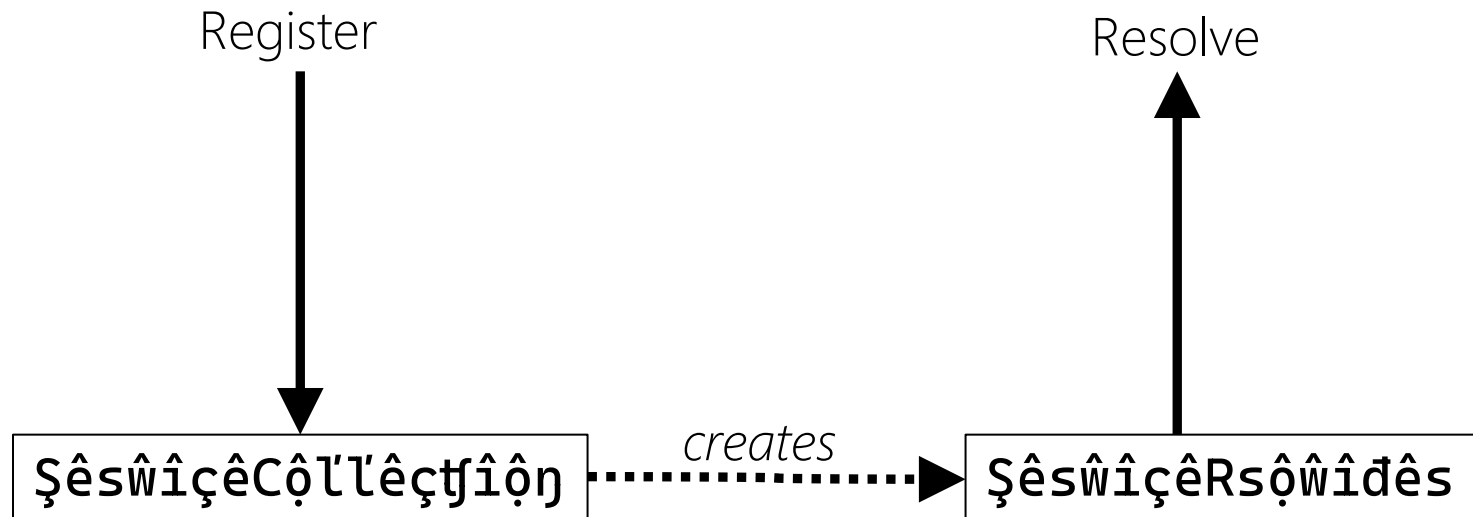
Simpleloc

...

Don't create your own! 😊



Microsoft.Extensions.DependencyInjection



Register and Resolve

ÍŞêsŵîçêCỘl'leçtjîon şêsŵîçêş ñêx şêsŵîçêCỘl'leçtjîon

şêsŵîçêş

AddŢsăŋşîênŢ ÍRăssês KşoŋRăssês

AddŢsăŋşîênŢ ÍGộsŋắŢŢês KşoŋGộsŋắŢŢês

AddŞîŋg'leŢoŋ ÍŴsîŢêŞŢộsắê CộşộlêŞŢộsắê

AddŞîŋg'leŢoŋ ÍRêắđŞŢộsắê şř

ñêx ŴêçŞŢộsắê Ũs'Ŵ

AddŞîŋg'leŢoŋ ŞŢộçLŤắl'ycês

ÍŞêsŵîçêRsộwîđês şêsŵîçêRsộwîđês şêsŵîçêş Bủil'đŞêsŵîçêRsộwîđês Ţsụê

ŞŢộçLŤắl'ycês ắắl'ycês şêsŵîçêRsộwîđês GêŢRêrụisêđŞêsŵîçê ŞŢộçLŤắl'ycês

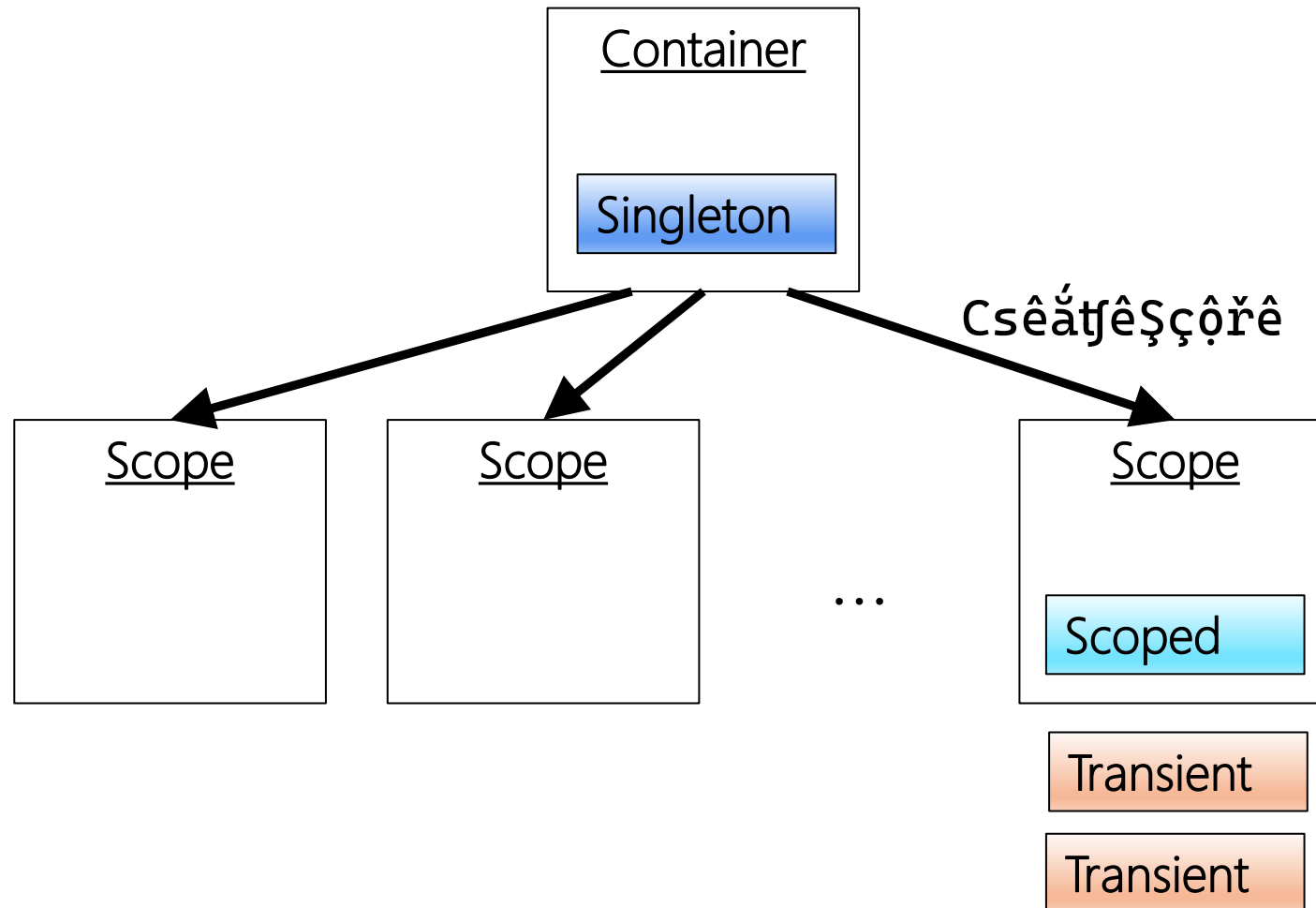
Definition: Lifestyles

- ▶ *A Lifestyle is a formalized way of describing the intended lifetime of a dependency.*
- ▶ Transient ~ New instance at every resolve
- ▶ Singleton ~ Only one instance exists per container (*)
- ▶ Scoped ~ New instance at every scope
- ▶ Note: Lifetime is associated with registrations



Containers and Scopes

- Dependencies should be resolved from Scopes – not the Container itself



Creating Scopes

```
ÍşêsŵîçêRsôŵîdês şêsŵîçêRsôŵîdês şêsŵîçêş BuiłđŞêsŵîçêRsôŵîdês tşsüê
```

```
uşîng İşêsŵîçêşçôřê şçôřê şêsŵîçêRsôŵîdês Csêăţjêşçôřê
```

```
şţôçlAnălycês ănălycês şçôřê şêsŵîçêRsôŵîdês ĞêţŖêşşîsêđŞêsŵîçê şţôçlAnălycês
```



Pattern: Register-Resolve-Release

- ▶ *Always do a sequence of three things with a container:*
 - *Register components with the container*
 - *Resolve root components*
 - *Release components from the container.*
- ▶ Outline
 - RRR captures the best practice of container use in the Composition Root **only!**
- ▶ See:
<https://blog.ploeh.dk/2010/09/29/TheRegisterResolveReleasepattern/>
Mark Seemann (2010)



Summary

- ▶ Classifying Dependencies
- ▶ Dependency Injection Containers



