Using Mixin to Move Responsibilities Out

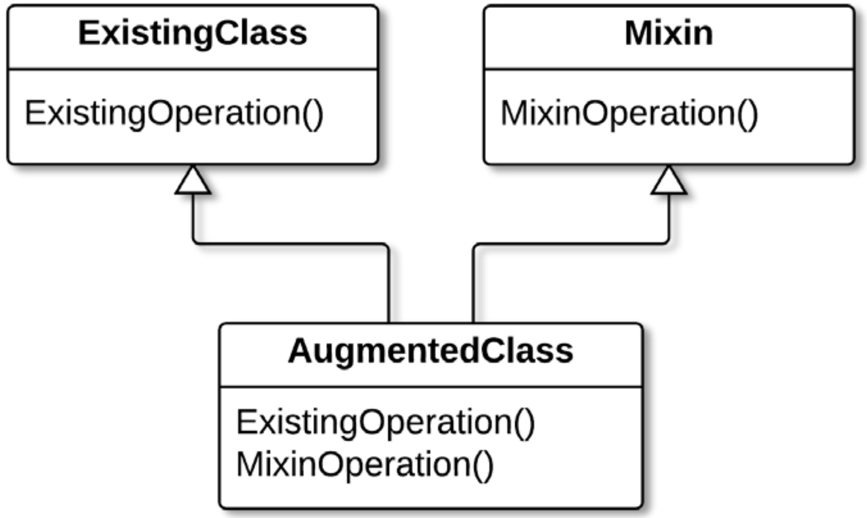
Zoran Horvat

@zoranh75 | www.codinghelmet.com

Mixin Class Defined

"A mixin class is a class that's intended to provide an optional interface or functionality to other classes. It's similar to an abstract class in that it's not intended to be instantiated. Mixin classes require multiple inheritance."

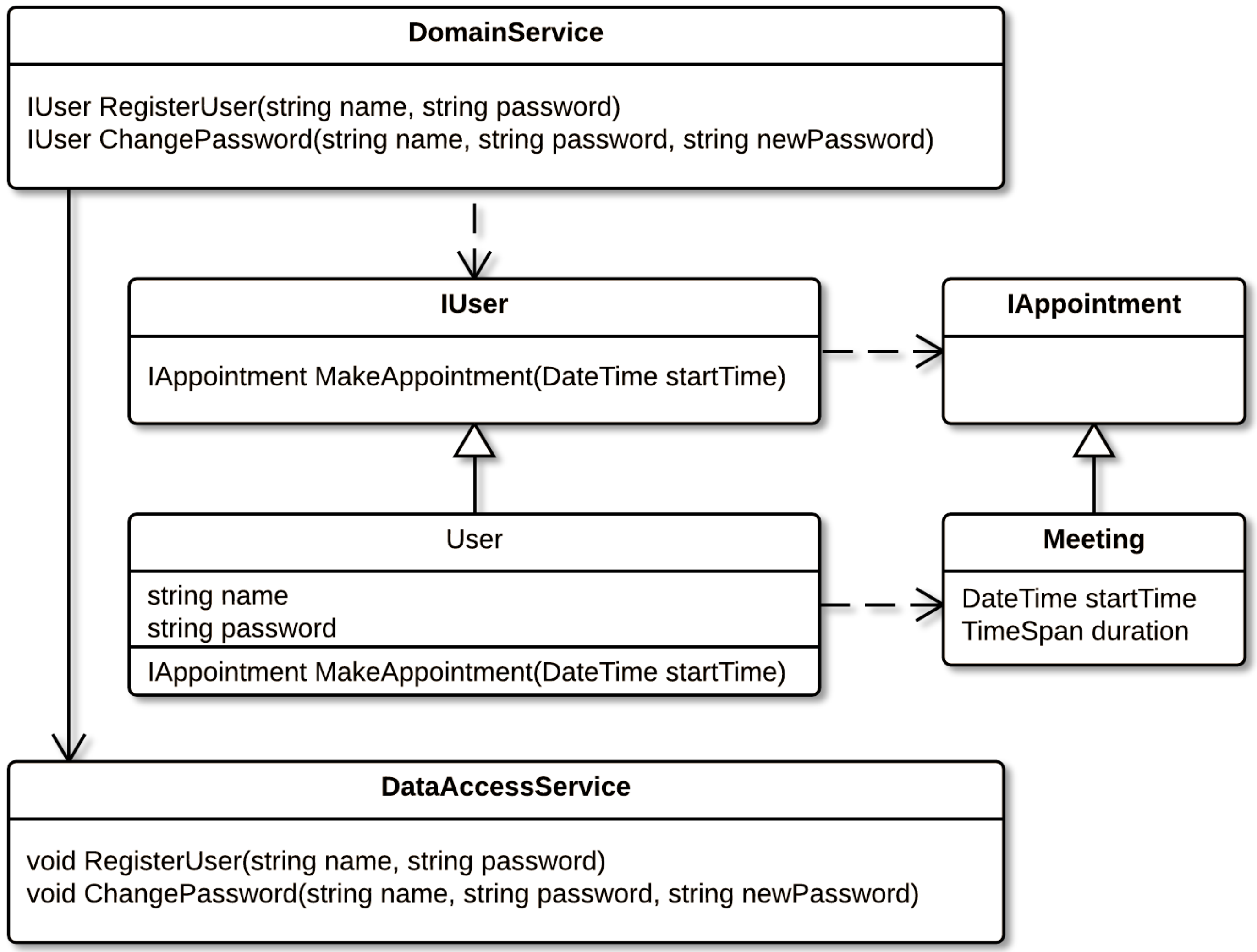
Gamma et al., Design Patterns: Elements of Reusable Object-Oriented Software



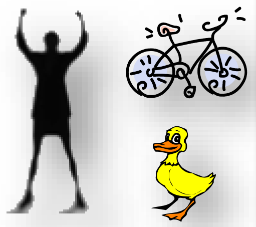
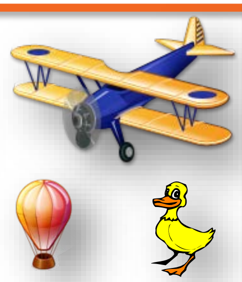
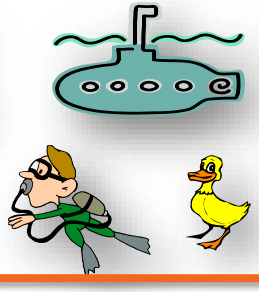
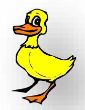
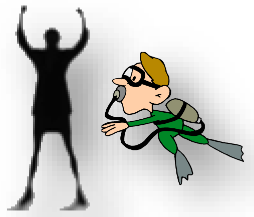
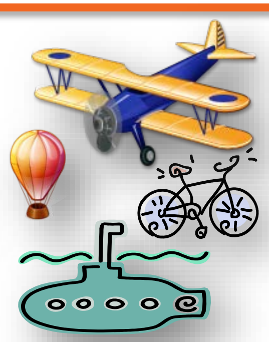
Mixins in .NET

- .NET languages do not support multiple class inheritance
  - Mixin can be an interface
- A class can implement interfaces even when it has a base class
  - Existing class can also be an interface

An Example


DomainService
IUser RegisterUser(string name, string password)
IUser ChangePassword(string name, string password, string newPassword)
IUser
IAppointment MakeAppointment(DateTime startTime)
IAppointment
User
string name
string password
IAppointment MakeAppointment(DateTime startTime)
Meeting
DateTime startTime
TimeSpan duration
DataAccessService
void RegisterUser(string name, string password)
void ChangePassword(string name, string password, string newPassword)

Motivation Behind Mixin Classes

Feature	Ability to Move	Ability to Fly	Ability to Dive
Classes			
Hierarchy	Animals	People	Vehicles
Classes			

Is Mixin a Pattern?

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice."

Alexander et al., A Pattern Language

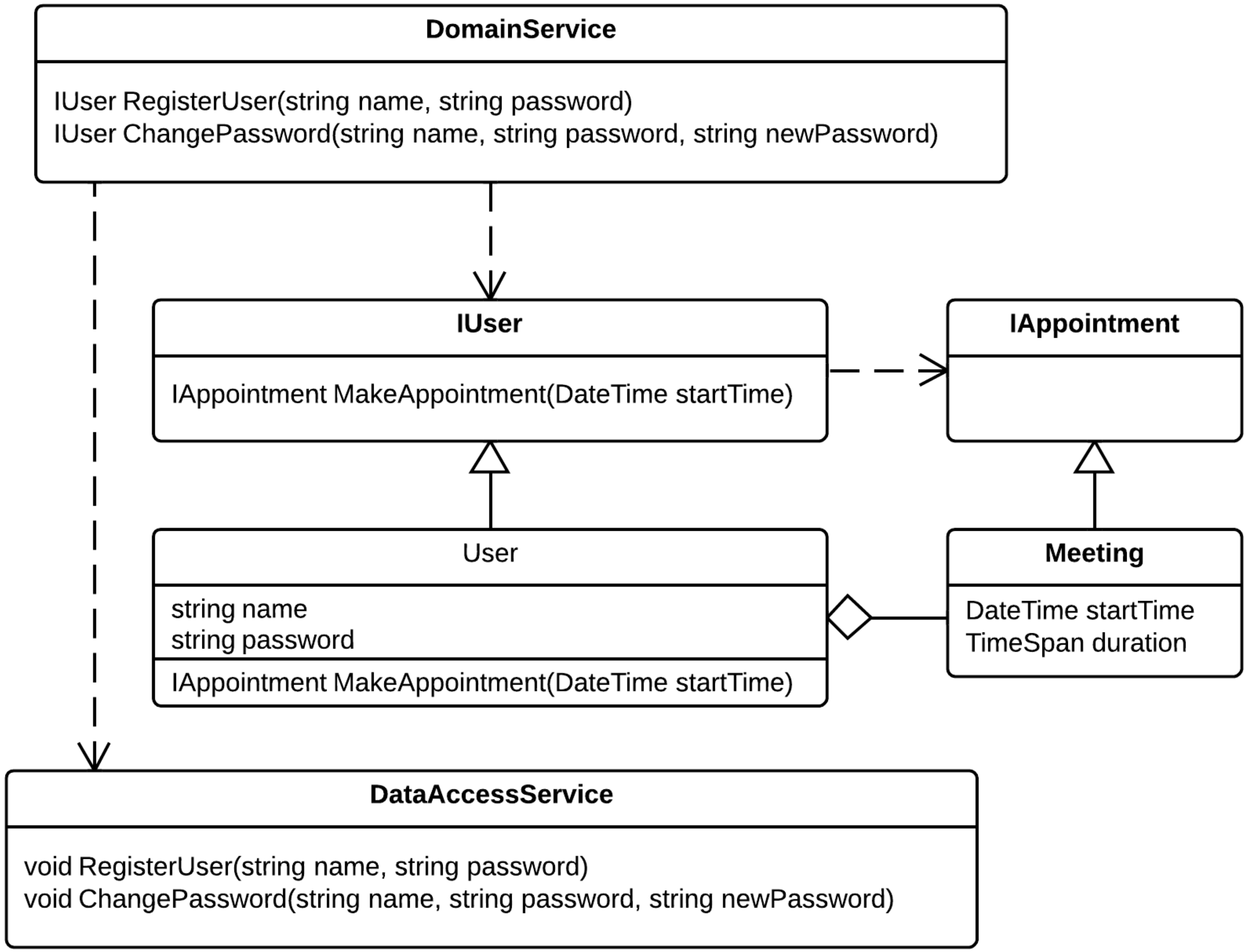
Is Mixin a Pattern?

- Does the language support Mixins natively?
  - Common Lisp has an idiomatic implementation of Mixins
  - C++ supports multiple class inheritance
- Mixins in C#
  - Not supported out of the box
  - Used in a way characteristic to patterns

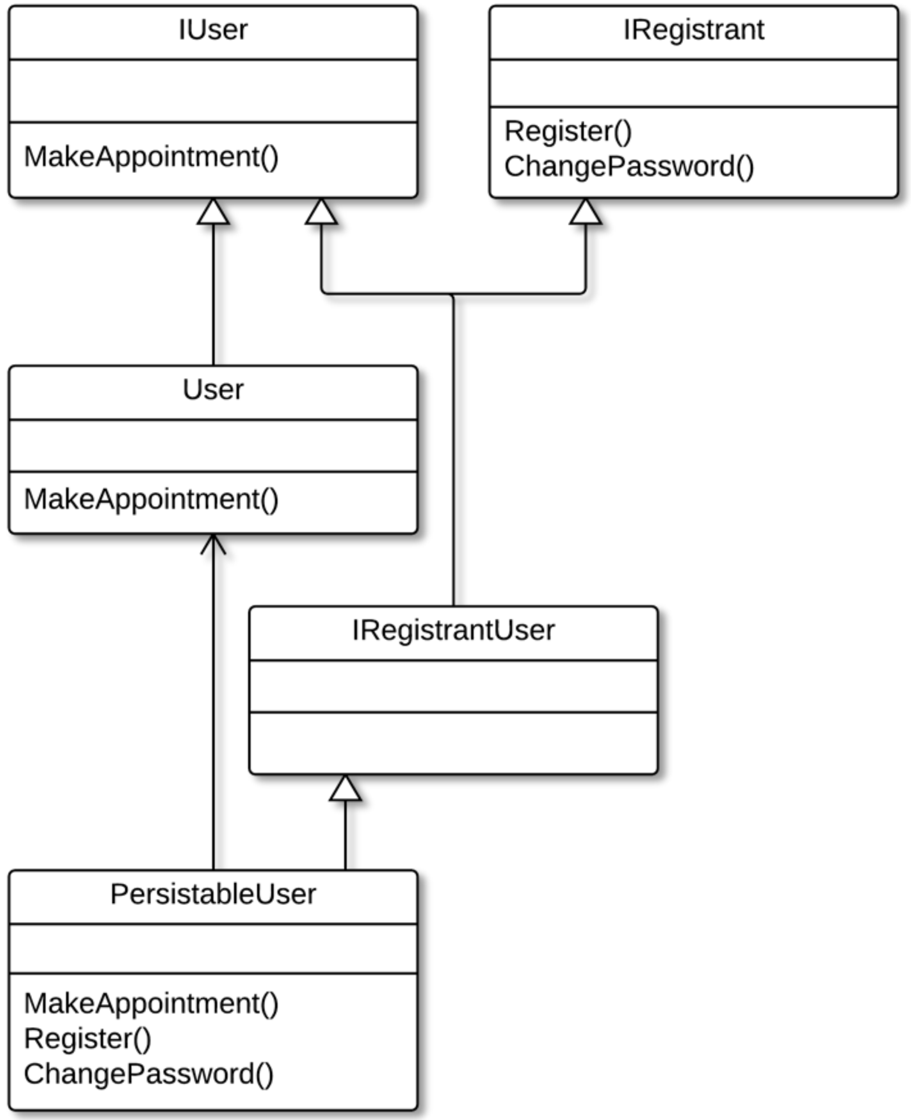
Implementing Mixins in C#

- Aspect-oriented frameworks
  - Intention to mix interfaces is expressed using attributes
    - Beware of strange solutions in these frameworks
  - Frameworks add complexity to the project
    - Make sure that added complexity pays off
- Using extension methods
  - New method is added to the original class
  - Extension method acts as any other method defined on the class
    - Extension method can only see public members of the class
- The goal of this module
  - Implement a simple solution
  - Intentions must be clearly indicated in code

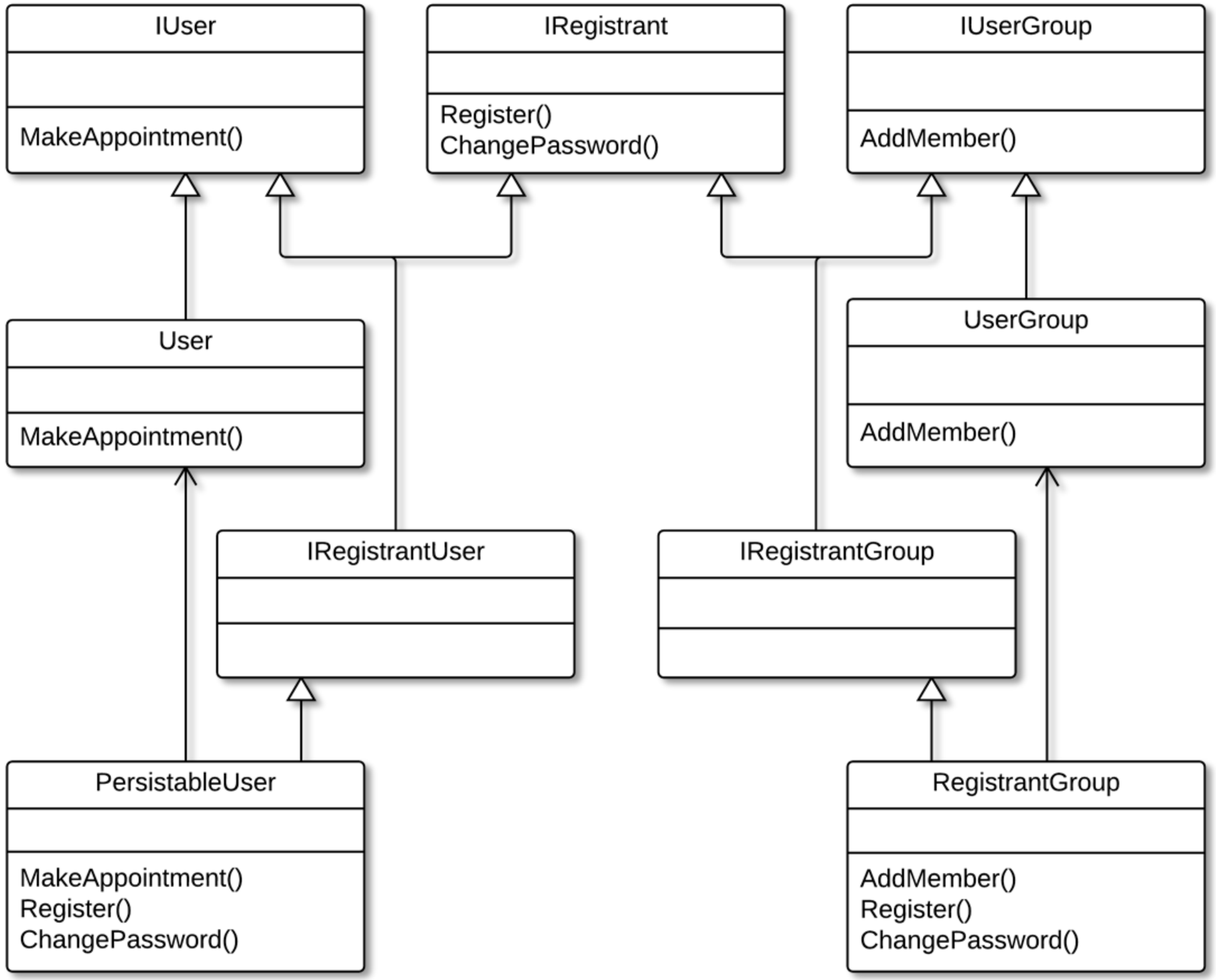
Example Domain Model


DomainService
IUser RegisterUser(string name, string password)
IUser ChangePassword(string name, string password, string newPassword)
IUser
IAppointment MakeAppointment(DateTime startTime)
IAppointment
User
string name
string password
IAppointment MakeAppointment(DateTime startTime)
Meeting
DateTime startTime
TimeSpan duration
DataAccessService
void RegisterUser(string name, string password)
void ChangePassword(string name, string password, string newPassword)

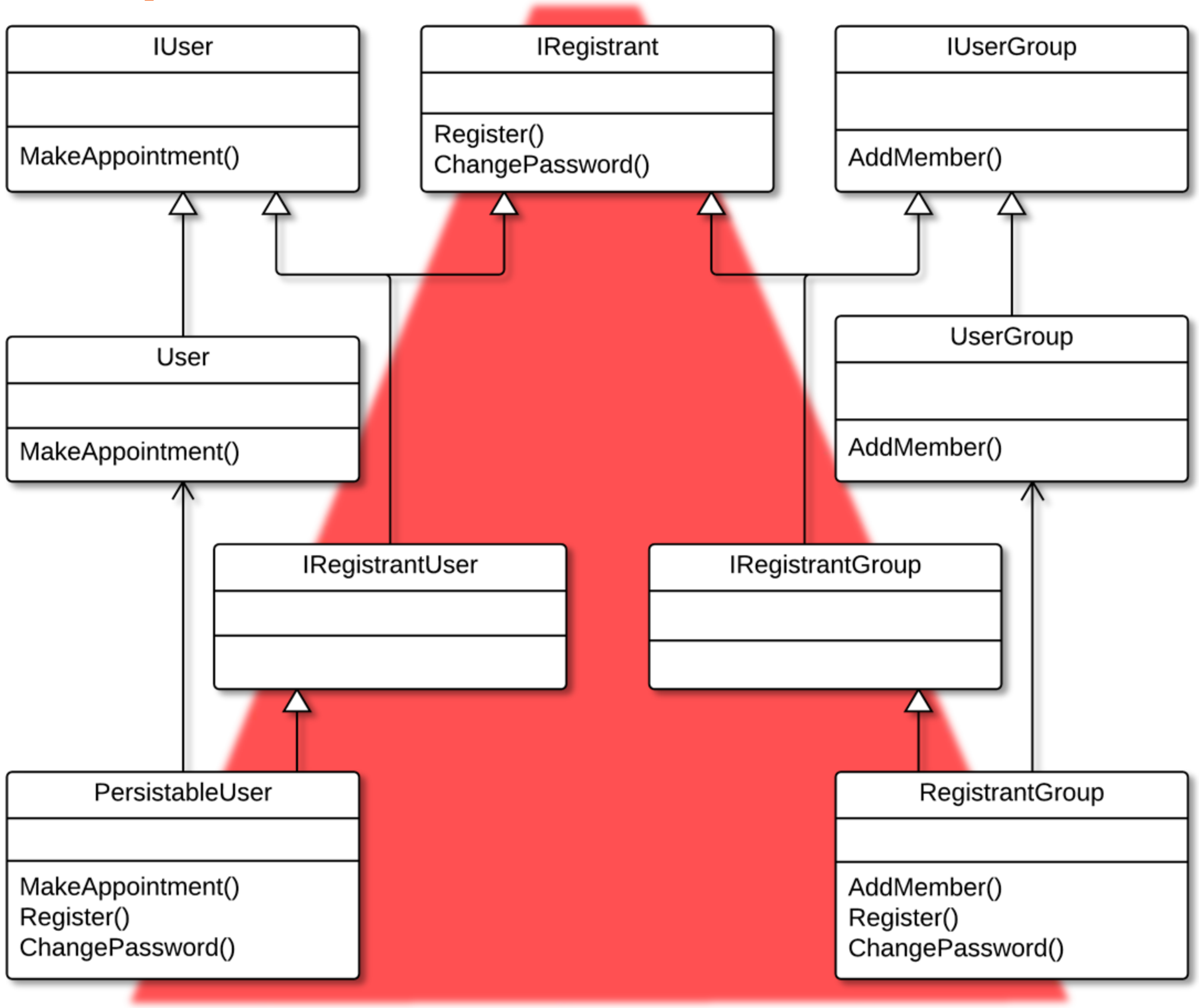
Evolution of the Mixin Class



Mixing the Interface to Other Classes



Polymorphic Behavior of Mixin Classes



Summary

- The question of additional responsibilities
  - Enclose them in a separate class with the same interface
  - Augmented class can fit in the layer which hosted the operation
    - Saving and loading mixins implemented in the Data Access or Infrastructure Layer
    - Service-like mixins implemented in Service Layer

Summary

- New feature is defined by the mixin interface
  - Augmented class has all features of the original class
  - Plus the mixin interface implementation
- Client interested in added feature uses augmented class
  - Core class adheres to Single Responsibility Principle (SRP)
  - Additional responsibilities wrapped in augmented classes
    - One responsibility per augmented class
- Mixins can be used polymorphically
  - Classes from different hierarchies can be augmented with the same mixin interface

Course Summary

- Design patterns are not applied to the solution
  - Solution is transformed to accommodate a pattern
  - Pattern implementation is then tailored to the project
- When is the best time to apply a design pattern?
  - When there is a problem in the existing design
  - This leads to a better design

Course Summary

- Issues connected with class responsibilities
  - Classes should not implement many responsibilities
- Otherwise, issues start to pile up
  - Low cohesion complicates use of a class
  - Rigid design prevents reuse
  - Large interface imposes unwanted dependencies on the client

Course Summary

- Two ways to move responsibilities out of a class
  - Compositional/collaborative approach
    - Instantiate small objects and compose them into a graph
    - Abstract Factory, Composite, Chain of Responsibility
  - Friend approach
    - C++ defines *friend* keyword which lets other class see internals of a class
    - C# defines internal access modifier and InternalsVisibleToAttribute
    - Visitor design pattern lets others see internals of a class
    - Mixin lets another class augment the original class with new feature

Further Learning

- ***Design Patterns Library*** by Steve Smith and other authors
- ***SOLID Principles of Object Oriented Design*** by Steve Smith
- ***Encapsulation and SOLID*** by Mark Seemann
- ***Domain-Driven Design Fundamentals*** by Steve Smith and Julie Lerman
- ***C# Programming Paradigms*** by Scott Allen

Further Reading

- ***Design Patterns – Elements of Reusable Object-Oriented Software***, Gamma et al.
- ***Object Design: Roles, Responsibilities, and Collaborations***, Rebecca Wirfs-Brock and Alan McKean
- ***Agile Principles, Patterns, and Practices in C#***, Robert C. Martin
- ***Refactoring: Improving the Design of Existing Code***, Martin Fowler
- ***Domain-Driven Design: Tackling Complexity in the Heart of Software***, Eric Evans
- ***Implementing Domain-Driven Design***, Vaughn Vernan
- ***Real-World Functional Programming: With Examples in F# and C#***, Tomas Petricek

Comments Are Welcome

- Please post your thoughts and ideas in the discussion section
- Visit http://www.codinghelmet.com for latest articles and examples on design patterns and software design
- Follow @zoranh75 on Twitter for updates