Step 3 (Classes, Namespaces, assemblies)

Wednesday, January 9, 2019 19:02

Objective

- Understand solution structure
- Understand what project is
- Understand how classes are related to projects
- Understand how projects are referencing each other
- Get familiarity with classes definition

Links

https://mva.microsoft.com/en-US/training-courses/c-fundamentals-for-absolute-beginners-16169
17-18

https://metanit.com/sharp/tutorial/
Chapters:

> Классы. Объектно-ориентированное программирование

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/generics/

https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/inheritance

Self-assessment task

Note:
Application created in this test task will be used for next test tasks. Consider robust implementation ready for further extensions.

Solution overview:
Application we are going to work on is Time Tracking system for a company. Aim of the project - is to track time, spent for each project by each employee. Users are allowed to submit time for a project. Admin users are able to manage users, projects and generate reports.

Application should provide functionality for interaction with a user.
There should be a possibility to select User Interactions from console (implementation will be empty yet!). Define user interaction which you consider to be useful. Note: actual implementation for these actions will be implemented in the next test task

Required user interactions:

- Login/logout
- View list of available projects
- Submit time
- View time, submitted for a project
- For project leader:
  - Run predefined report: show time, submitted for the project in current month
- For admin:
  - CRUD operations for users, projects
  - Assign project for user
- (You can extend application with any functionality you like)

At the end (after Step 6) next functionality will be implemented in the application:

- Data Contract assembly and its classes allows specifies data contract for Time tracking system data
- Business assembly and its classes allows to manage users, manage admin users, manage user's projects, manage time submitted to a project, create reports based on certain conditions
- UI module (Console application will be implemented in scope of the course, solution will be robust enough that UI might be extended/switched depending on context: WPF, ASP.NET or console)
- Data repository (XML approach will be implemented in scope of the course, solution will be robust enough that Approach might be selected/switched depending on context: xml, json or db)

First steps:
Create C# solution. It should consist of next projects. Names for projects should be used as described below

- "Application"
  - Console application
  - Startup project
  - Is responsible for the program flow, user interaction. (e.g. all the logic asking user to input any decisions etc. should go here)
- "DataContracts"
  - Assembly
  - Contains data classes definition
- "Infrastructure"
  - Contains any classes that might be used in different solutions
- "Business

    - Contains everything related to data processing and manipulations

Create classes structure required for managing/storing data and put these classes into DataContract assembly. Provide cleverhierarchy tree (base classes)
Note: DataContract should contain Normalized data (link)

Put additional useful fields in. All data entities should have BaseEntity as base class.

- BaseEntity
    - Id
    - Comment
- User
    - Username
    - Password
    - FullName
    - IsActive
    - AccessRole (enum)
- Project
    - Name
    - ExpirationDate
    - MaxHours
    - LeaderUserId (key for a User)
- TimeTrackEntry
    - UserId (who submitted)
    - ProjectId (where submitted)
    - Value (how much was submitted, in hours)
    - Date (when submitted)

Step 4 (Collections, Linq, Exceptions, Events)

Wednesday, January 9, 2019 19:02

Objective

- Understand what is collection
- Understand how to use LINQ
- Understand what is event and delegate
- Understand how to use exceptions

Links

https://mva.microsoft.com/en-US/training-courses/c-fundamentals-for-absolute-beginners-16169
19-23

https://metanit.com/sharp/tutorial/
Главы:

> Коллекции
> LINQ
> Обработка исключений
> Делегаты, события и лямбды

Exceptions

> https://docs.microsoft.com/en-us/dotnet/api/system.exception?view=netframework-4.7.2

Linq

> https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/working-with-linq
> https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/linq-to-objects
> https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/linq-to-xml
> https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/linq-to-adonet-portal-page

Events

> https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/
> https://docs.microsoft.com/en-us/dotnet/api/system.action?view=netframework-4.7.2
> https://docs.microsoft.com/en-us/dotnet/api/system.func-2?view=netframework-4.7.2

Mediator (pattern)

> https://refactoring.guru/ru/design-patterns/mediator
> Книга "Паттерны проектирования" Эрик Фримен, Элизабет Фримен. Глава 14 "Другие паттерны", "Посредник"

Self-assessment task

Extend application from Test Task 3 with actual implementation for user interactions.

> Business assembly should contain DataFacade class. This class should contain several properties with 'Services' classes (e.g. public UserServices UserServices {get;set;})
> Each 'Service' class will contain some specific logic for interaction between Business objects, e.g. GetAllUsers(), GetAllActiveUsers()
> Service classes should provide an access to business objects with populated data.
> BusinessObjects should provide events that some of its data has been changed, some code should be subscribed to this events, e.g. if IsActiveChanged - do some specific logic in the service

Business objects example:

- UserData (class)
  - User (property for reference to User class from DataContracts)
  - SubmittedTime (List<TimeTrackEntry>)
  - IsActiveChanged (event)

Note!

> Difference between business object and DataContract object is that BusinessObject contains some logic, it is not normalized, it contains references to collection of related objects etc.
> For now services should get data from some "stub" classes. Create simple static methods for providing subsets of predefined/hardcoded data (stubs).

Application should provide several reports (at least 3). LINQ should be used in services. E.g. 'Get active users for project X who submitted at least 40 hours this month'

Consider graceful exceptions handling. Some of your code should throw exceptions where it makes sense. Exceptions are to be handled respectively.

Step 5 (Delegates, OOP, GC)

Wednesday, January 9, 2019 19:02

Objective

- Get more practice in classes definition
- Get practice with interfaces
- Understand what is GC and how it works

Links

https://metanit.com/sharp/tutorial/

Главы:

> Делегаты, события и лямбды
> Сборка мусора, управление памятью

https://docs.microsoft.com/en-us/dotnet/standard/garbage-collection/index

Self-assessment task

1. Extend application with 3 new assemblies:

    > Contracts
        This assembly should contain interfaces which might be substituted.
        Create generic class IRepository<T> where T: BaseEntity
        IRepository should have methods needed for data storage or retrieving, e.g.
            IEnumerable<T> GetAll
            Insert(T)

        You might also add specific repositories here, e.g.
        IUserRepository : IRepository<User>
    > Repositories.Xml
        Should contain real implementation for IReposetory<T>.
        For now stub data population should be performed.
        None of projects except Solution should contain references to this assembly.
    > Solution
        Should provide a class responsible for getting your interfaces implementation
        E.g. Mapper class
        IUserRepository GetUserRepository() { return new XmlUserRepository(); }

2. Create 'Mediator' class which provides events for useful application changes. Mediator will also have some methods (which might be called outside) leading to event raise.

    > In the Mediator class add method SubscribeToSomething(Action action)
    > Some of the application BusinessObject should put action delegate to a Mediator. Actions are to be stored in a collection and executed under some condition
    > Consider unsubscribe to avoid memory leaks. (Includes unsubscribe for events from Test Task 4)
    > Method IDisposable should be used here

Step 6 (Serialization, xml)

Wednesday, January 9, 2019 19:03

Objective

- Get familiarity with XML
- Get familiarity with Serialization
- Get familiarity how to work with XML

Links

https://metanit.com/sharp/tutorial/

Главы:

> Работа с XML

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/using-statement
https://docs.microsoft.com/en-us/dotnet/api/system.idisposable?view=netframework-4.5

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/

Self-assessment task

In Repositories.Xml library add actual implementations for storing/retrieving data to XML files.