C# - Week 2

Exercise *Library*

You have the following relations between entities:

- There is a Library that has a collection of Books and a list of authorized Users.
- Each Book has an Author, title and a unique id (isbn in real life).
- The Library is operated by a Librarian
- The User can make request regarding Authors and Book availability only by asking the Librarian. No direct contact to the other entities should be attempted!
- Create all the necessary classes as much independent from each other as possible (loose coupling).
- The rest of the logic should be in class Program. In main() we want to call a static function `Program.AsksForBook(user, librarian, book)` which prints whether the user can access the book if the book exists in the library.

Exercise *Subsequence*

Write a method that finds the longest subsequence of equal numbers in a given List and returns the result as new List. Write a program to test whether the method works correctly. Implement the method

1. as a static method of a static class Utilities
2. as an extension method of the List

Exercise *Fraction*

1. Define a class Fraction, which contains information about the rational fraction (e.g. 1/4).
2. Define the appropriate fields, properties and constructors.
3. Override ToString() to print the Fraction (e.g. "1/4").
4. Override operator * to multiply two Fractions.
5. Define a static method Parse(string str) to create a Fraction from a string.
6. Define a property of type *decimal* to return the decimal value of the fraction (e.g. 0.25).
7. Implement IComparable interface to enable sorting of Fractions.
8. Write a function Cancel() to cancel the Fraction. (e.g. 10/15 is cancelled to 2/3).

Exercise *Mobile*

1. We want to model the functionality of a mobile device. In the namespace *Mobile* implement a class which holds information about a mobile device: model, manufacturer, base price, features of battery (battery type and capacity) and features of the screen (resolution and pixel density). Implement each class and the data structures associated with it in a separate file. After instantiation, the information cannot change.
2. For each class, add some static methods which return an instance of known models.
3. Add a class Usage which holds information about the usage of the mobile device. This includes current percentage of battery, OS information and call history. Each call record saves information about the date, time of start and time of end of the call and if it was incoming or outgoing. Provide useful properties like duration of the call.
4. Add methods to add/remove calls and also remove all.