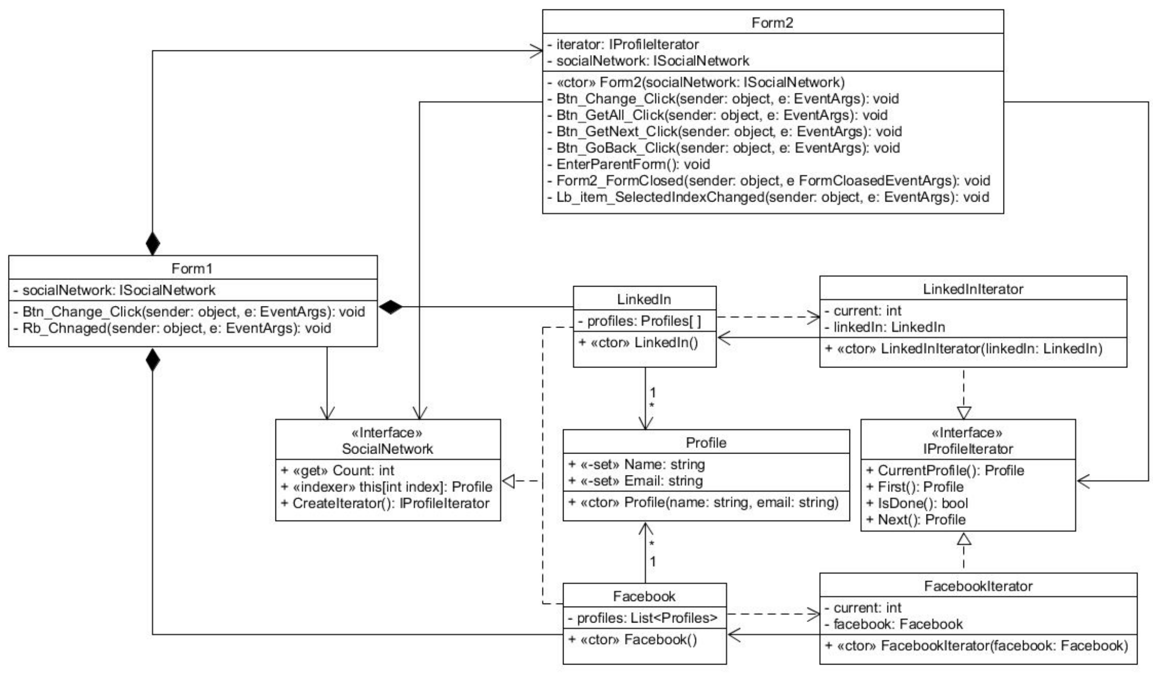
Adapter Pattern

Mrunal Patil - 3101851
Emil Karamihov - 2992884

Application Description:
The application is used to iterate through instances in a sequential order by keeping their underlying representation. This application will give the opportunity to the user to go through all of the profiles currently in the two social network systems. The first social network is Facebook and the second one is LinkedIn. These classes have their own Iterator FacebookIterator for Facebook and LinkedInIterator for LinkedIn. These iterators implement IProfileIterator which contains the signature of methods giving the functionality of the iterator pattern. The FacebookIterator has a instance of Facebook and LinkedInIterator has a LinkedIn instance. The reason for this is to make the connection between the collections and their iterators, since these classes contain different types collections and different profile values. For Instance, Facebook contains a list of profiles and linkedIn an array.
There is only one object of type Profile which is one and the same for Facebook and LinkedIn.
Additionally, the interface ISocailNetwork is an interface which is implemented by Facebook and LinkedIn. It creates the collection iterator depending on the type of the collection. Form2 (The client) is not aware of the different types of iterators and also the social networks. Form1, on the other hand, creates the and knows the collection types (ConcreateAggregates) and passes it to the Form2. Form2 is being created by Form1, it is a child form.
How it works
Form1 gives the user a chance to chose his collection type (Facebook or LinkedIn) and pases it as a ISocialNetwork object to Form2. Form2 is made to use the iterator and its functions. You can get profiles one by one; get more info about a profile by selecting the a name in the listbox. If the user wants to change the collection, he needs to click the go back button which will return Form1. From there the use case goes back to the beginning.


Form2
- iterator: IProfileIterator
- socialNetwork: ISocialNetwork
- «ctor» Form2(socialNetwork: ISocialNetwork)
- Btn_Change_Click(sender: object, e: EventArgs): void
- Btn_GetAll_Click(sender: object, e: EventArgs): void
- Btn_GetNext_Click(sender: object, e: EventArgs): void
- Btn_GoBack_Click(sender: object, e: EventArgs): void
- EnterParentForm(): void
- Form2_FormClosed(sender: object, e FormCloasedEventArgs): void
- Lb_item_SelectedIndexChanged(sender: object, e: EventArgs): void
Form1
- socialNetwork: ISocialNetwork
- Btn_Change_Click(sender: object, e: EventArgs): void
- Rb_Chnaged(sender: object, e: EventArgs): void
LinkedIn
- profiles: Profiles[ ]
+ «ctor» LinkedIn()
LinkedInIterator
- current: int
- linkedIn: LinkedIn
+ «ctor» LinkedInIterator(linkedIn: LinkedIn)
«Interface»
SocialNetwork
+ «get» Count: int
+ «indexer» this[int index]: Profile
+ CreateIterator(): IProfileIterator
1
*
Profile
+ «-set» Name: string
+ «-set» Email: string
+ «ctor» Profile(name: string, email: string)
«Interface»
IProfileIterator
+ CurrentProfile(): Profile
+ First(): Profile
+ IsDone(): bool
+ Next(): Profile
*
1
Facebook
- profiles: List<Profiles>
+ «ctor» Facebook()
FacebookIterator
- current: int
- facebook: Facebook
+ «ctor» FacebookIterator(facebook: Facebook)

Maintainability, Reusability, Extensibility.
This design is rather maintainable - keeps collections secure, keeps the code isolated, keeps concreateAggregates and concreateIterators isolated from the client. Reusability is good since there are different collection types like arrays and list and the methods are modifiable to according to the type of collection. Therefore the pattern can adjust the iterator methods accordingly. Extensibility is hard since we need to add two classes for a single upgrade. Imagen we add Twitter profiles, we have to make a twitter instance and an iterator for it.

Pros

1. Every single part of the code is isolated, it is easy to maintain.
2. Reusable since there are different collections types like arrays and list
3. Suitable for complex collections - maintainability
4. Every AggregateCollections and ConcrateIterators are isolated from the client Form2 making it more secure

Cons

1. When a new instance of an concreateAggegate is added a new class for it must be crated and also a iterator for it. It is hard to Extend since it requires knowledge over the pattern.
2. Since a new iterator must be created every time mostly with the same code with some specifics changed. There is a reusability problem.