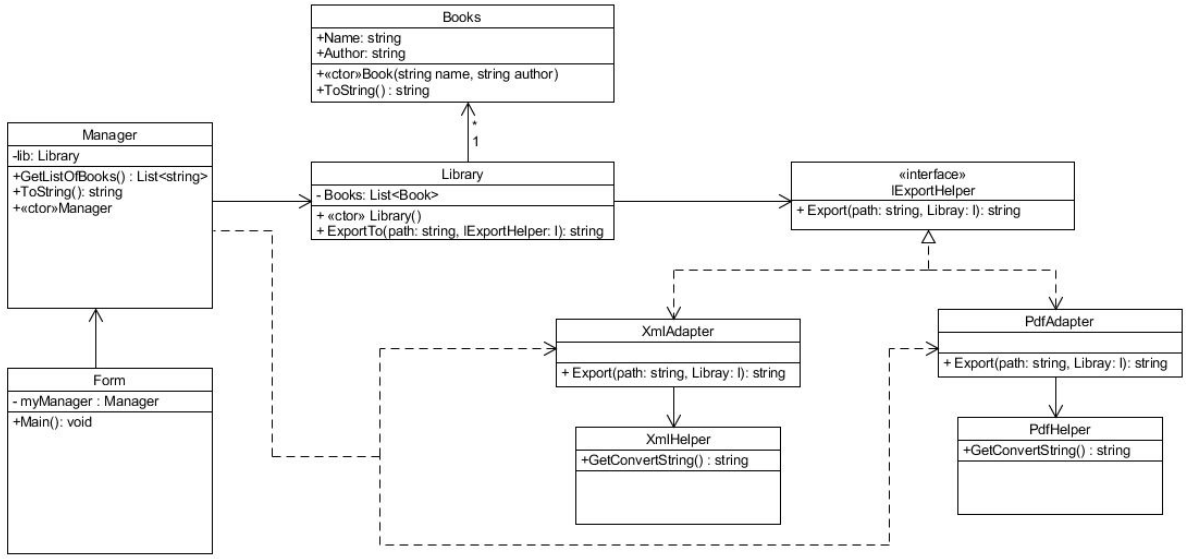
Adapter Pattern

Mrunal Patil - 3101851
Emil Karamihov - 2992884

Application Description: The application is to convert Books to different text formats. Eg:- Books are converted to .pdf, .docx, .txt, .xml, etc. The Form has an object of the Manager, Manager has information about all the Adapters and the Library. Adapters are the links to combine file helpers with the library. The adapter implements the interface IExportHelper. The interface has a method export with pathname and library as parameters where path is the path of the file and library is the library used to convert the file type.



Maintainability, Reusability, Extensibility.

When you want to reuse existing class, but its interface is not compatible with the rest of application's code.

The Adapter pattern creates a middle layer class that translates application calls to the form that the existing class understands.

You need to reuse several existing classes, but they lack some common functionality. And you can not add it to the superclass because it is either closed or used in other code.

You could put the missing functionality into a new adapter class. It will connect your app's code and the classes you are interested in. This solution looks very similar to the Visitor pattern.

Pros

1. Hides from the client code unnecessary implementation details of interface & data conversion.
2. Only the manager has an instance of the adapters, so if a new adaptee is added, there must be an instance in the manager
3. Client does not have to be aware of all the different interfaces in use
4. Adapter can use multiple adaptees, without exposing the complexity to the client
5. When adaptee changes only the adapter needs to be updated
6. You can easily change the library used for conversion of file in the adapter without affecting any other method or class

Cons

1. Increases overall code complexity by creating additional classes. Communication overhead