Table of Contents

[1 Solution Folders and Structure 2](#_Toc117759328)

[1.1 Startup project 2](#_Toc117759329)

[2 Add a new project 4](#_Toc117759330)

[2.1 Add new project 4](#_Toc117759331)

[2.2 Create structured folders 4](#_Toc117759332)

[2.3 Create report engine Decorator 5](#_Toc117759333)

[2.4 Create report entity 6](#_Toc117759334)

[2.5 Create program entity 7](#_Toc117759335)

[2.6 Update CoreSystemConsole main program 8](#_Toc117759336)

[3 Solution Structure 10](#_Toc117759337)

[3.1 Block Diagram 10](#_Toc117759338)

# Objective

This repository contains several report engines, mainly for explorer and try as a playground to test those report engines.

How is the functionality, what is reuqired on install and setup to work with .Net Core.

# Background

A new project is launch, to develop a ERP(Enterprise Resource Planning)/CMS (Content Management System) like system.

The front end will be Angular web application, the back end system will be separated as a standalone part to provide web service (web API)

We decided to build the back end by .Net Core with the "Entity Framework", "Code First Approach"

Therefore I need to exploe and test how the back end system satisfy the report generation functionality.

# Scope of this repository

Try and error for study how to implmenet report(xlsx, pdf) generation in C# .net Core5

### Back end engine/tools

* Crystal Report (excel, pdf) (Implemented)
* Jasper Report (excel, pdf) (Implemented)
* EPPlus (excel) (Implemented)
* iText (pdf) (Implemented)
* Puppeteer (pdf) (Implemented)
* IronPDF for .net (pdf) (Implemented)
* OpenXmlSDK (excel) (too complicated, I give up)

### Front end engine/tools

The front end engines always are JS library, most likely in pure javascript or officially support NodeJs or having NodeJs version. Usually NodeJS is it not necessary in read/write but may be required if develop on those JS library.

The front end JS libraries feature is limited and can't satify my needs compare to the library which is based on c# in my observation.

If you want a lite and the simplest report read/write function, just check out below listed library, those were not included in this project becuase this project is not focus on the report read/write in front end.

* parallax/jsPDF (write pdf)
* mozilla/pdf.js (read pdf)
* SheetJS (read/write excel)

# Configuration and setup

1. Control which reports you would like to test
2. Control the report generate folder
3. Control which reports you would like to test

comment and uncomment the lines in

|  |
| --- |
| SolutionRoot\CoreSystemConsole\Program.cs |

|  |
| --- |
| // comment-off the Report Entity Program  //InvoiceProgram invoiceProgram = new InvoiceProgram();  //HitRateHTMLProgram hitRateHTMLProgram = new HitRateHTMLProgram();  //HitRateXMLProgram hitRateXMLProgram = new HitRateXMLProgram();  //EPPlus5XlsxTemplateProgram ePPlus5XlsxTemplateProgram = new EPPlus5XlsxTemplateProgram();  ITextGroupIPdfTemplateProgram iTextGroupIText5PdfTemplateProgram = new ITextGroupIPdfTemplateProgram(); |

1. Control the report generate folder

open and edit

|  |
| --- |
| SolutionRoot\CoreReport\VisualizationEntity.cs |

|  |
| --- |
| protected string tempRenderFolder = @"D:\\Temp"; // report will be generated in this directory |

# Installation

If you want to test all report engines, please install the report engines and complete the configuration setup

Some engines installation is not required, because those library develop under pure C# and already installed by NuGet library manager

Some engines installation is required, because those library relay on the external executable program like JAVA…

for examples, JasperReport and Crystal Report need to install

The installation steps details are described below

## Crystal Report

### Pre-installation

Before run, please install Crystal Reports, Developer for Visual Studio Downloads <https://wiki.scn.sap.com/wiki/display/BOBJ/Crystal+Reports%2C+Developer+for+Visual+Studio+Downloads>

### Documentation

Connecting to Object Collections <https://help.sap.com/viewer/0d6684e153174710b8b2eb114bb7f843/SP21/en-US/45afd8f46e041014910aba7db0e91070.html>

### Example and Tutorial

Tutorial: Connecting to Object Collections <https://help.sap.com/viewer/0d6684e153174710b8b2eb114bb7f843/SP21/en-US/45c50fec6e041014910aba7db0e91070.html>

## Jasper Reports

### Pre-installation

Before run, please install .NET jsreport sdk(jsreport binary, jsreport local) by nuget <https://jsreport.net/learn/dotnet>

### Documentation

jsreport documentation <https://jsreport.net/learn>

Recipes <https://jsreport.net/learn/recipes>

Templating engines <https://jsreport.net/learn/templating-engines>

.Net local reporting <https://jsreport.net/learn/dotnet-local>

.Net Client <https://jsreport.net/learn/dotnet-client>

### Example and Tutorial

GitHub jsreport/jsreport-dotnet <https://github.com/jsreport/jsreport-dotnet>

**Page header, footer, page number**

Merge dynamic header with items

<https://playground.jsreport.net/w/admin/ihh7laK2>

Merge header and footer with page numbers

<https://playground.jsreport.net/w/admin/kMI4FBmw>

Merge with render for every page enabled

<https://playground.jsreport.net/w/admin/1A7l_UG_>

## EPPlus5

EPPlus5 is open source, but you are required to purchase license for commercial use

The library(ies) were installed under the project through Package Manager Console

Install-Package EPPlus -Version 5.8.0

## iText Group

Some products of iText 7 Suite is open source, but you are required to purchase license for commercial use

The library(ies) were installed under the project through Package Manager Console

Install-Package itext7 -Version 7.1.16

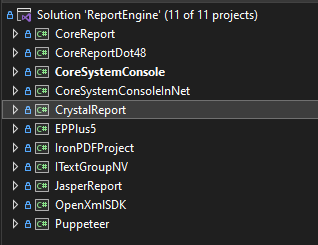
Install-Package itext7.pdfhtml -Version 3.0.5

# Demo

If you want to test the Crystal Report, change the startup project to “CoreSystemConsoleInNet”

If you want to test others report engines, change the startup project to “CoreSystemConsole”

# Solution Folders and Structure



## Startup project

Because Crystal Report only support under .net framework, you must change the start-up project to “CoreSystemConsoleInNet” for debug, and test Crystal Report.

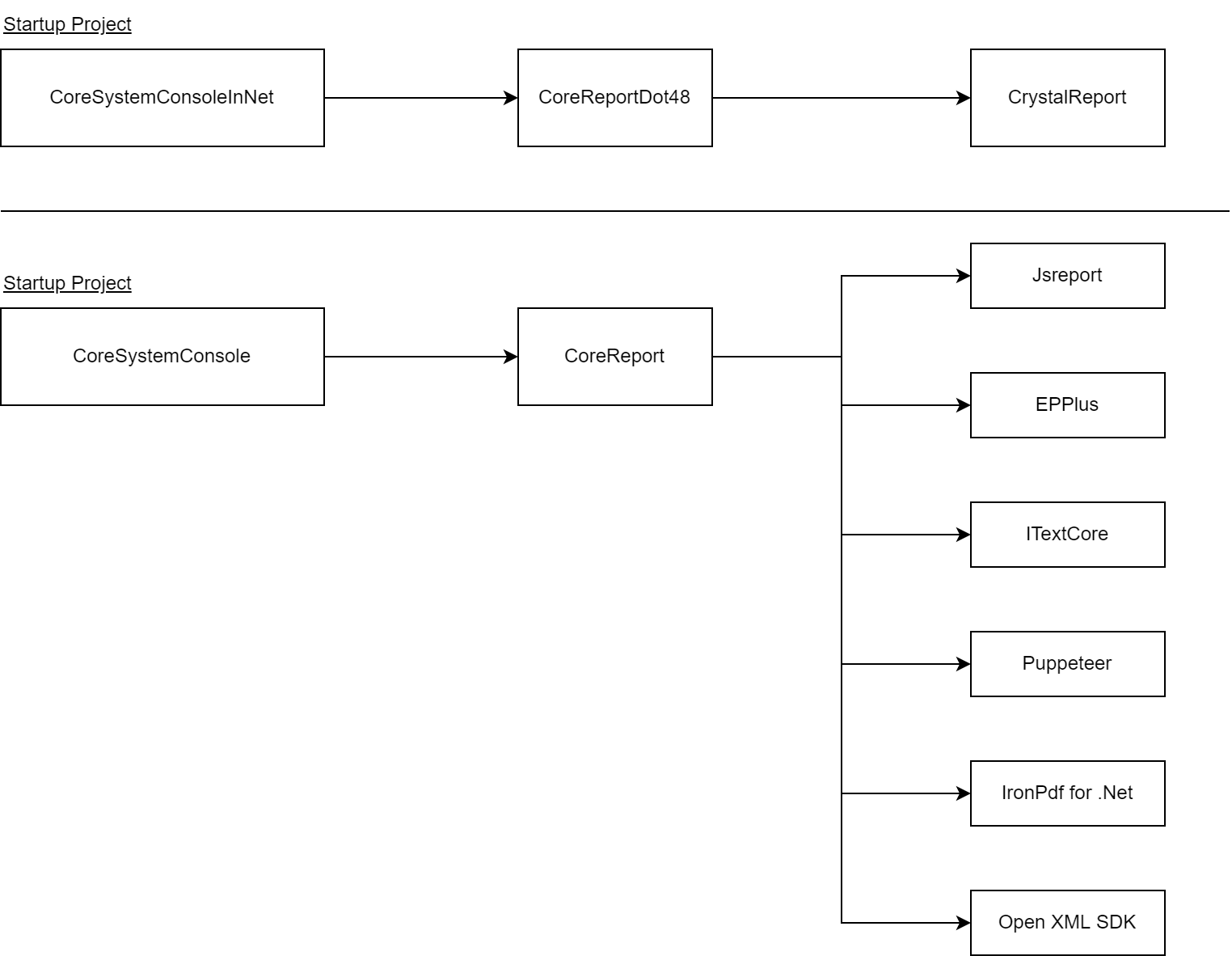
For the future the project would create in .Net Core probably (5 is the latest when I create this github repository, 6 is publish when I create this document), that is why I choose .Net Core in other projects.

**For Crystal Report, change startup project to “CoreSystemConsoleInNet”**

Program flow: CoreSystemConsoleInNet (.net framework 4.8) 🡺 CoreReportDot48 🡺 CrystalReport

**For other report library, change startup project to “CoreSystemConsole”**

CoreSystemConsole (.Net Core 5) 🡺CoreReport 🡺 JSReport / EPPlus / ITextCore / Puppeteer / IronPdf for .Net / Open Xml SDK (MS office sdk)



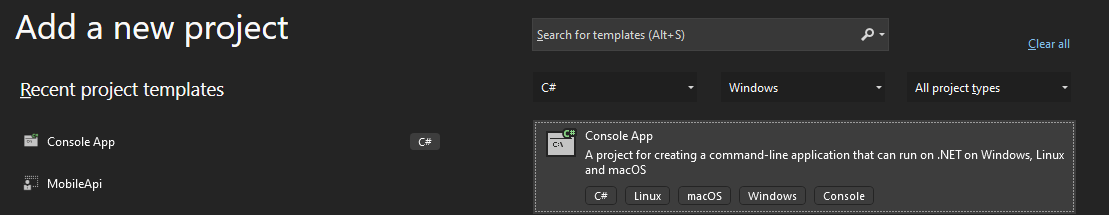
# Add a new project

Steps

1. Add new project
2. Create structured folders
3. Create report engine Decorator
4. Create report entity
5. Create program entity
6. Update CoreSystemConsole main program

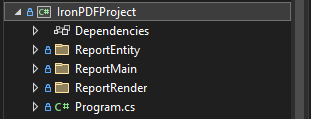
## Add new project

To use a new report library, create a new Console App project.



## Create structured folders

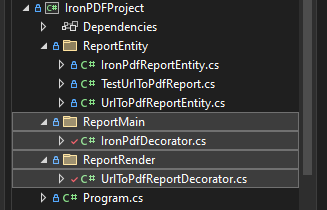
Create folders as below structure



## Create report engine Decorator

In ReportMain folder, create a generic Decorator that inheritance VisualizationDecorator

For IronPDFProject, I created IronPdfDecorator.cs, you should put all common object and/or initialize variable that would be used by the report engine in this class.



For IronPDF, it could generate pdf in several approaches, include

* HTML(source code) to PDF
* HTML Files to PDF
* URL to PDF
* ASPX Pages to PDF
* Images To PDF
* TIFF to PDF with Multi-Page Support
* Javascript in Html To Pdf

To separate the purpose and related resource in terms of different approach, I also created UrlToPdfReportDecorator.cs to control the data flow of “URL to PDF”. UrlToPdfReportDecorator.cs is inheritance IronPdfDecorator.cs

## Create report entity

In ReportEntity folder, create a generic Report Entity that inheritance BaseReportEntity

For IronPDFProject, I created IronPdfReportEntity.cs,

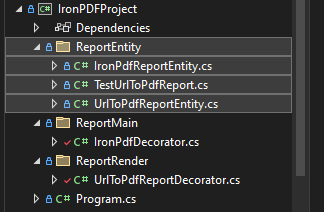
UrlToPdfReportEntity inheritance IronPdfReportEntity.cs

To separate the purpose and related resource in terms of different approach, there should be

* HtmlSourceToPdfReportEntity
* HtmlFilesToPdfReportEntity
* ImageToPdfReportEntity
* …etc

To separate the purpose and related resource in terms of different approach, all report programs that should inheritance UrlToPdfReportEntity if it is going convert a URL to PDF. For me to test the URL to PDF function, I also created TestUrlToPdfReport.cs to control the data of “URL to PDF”.

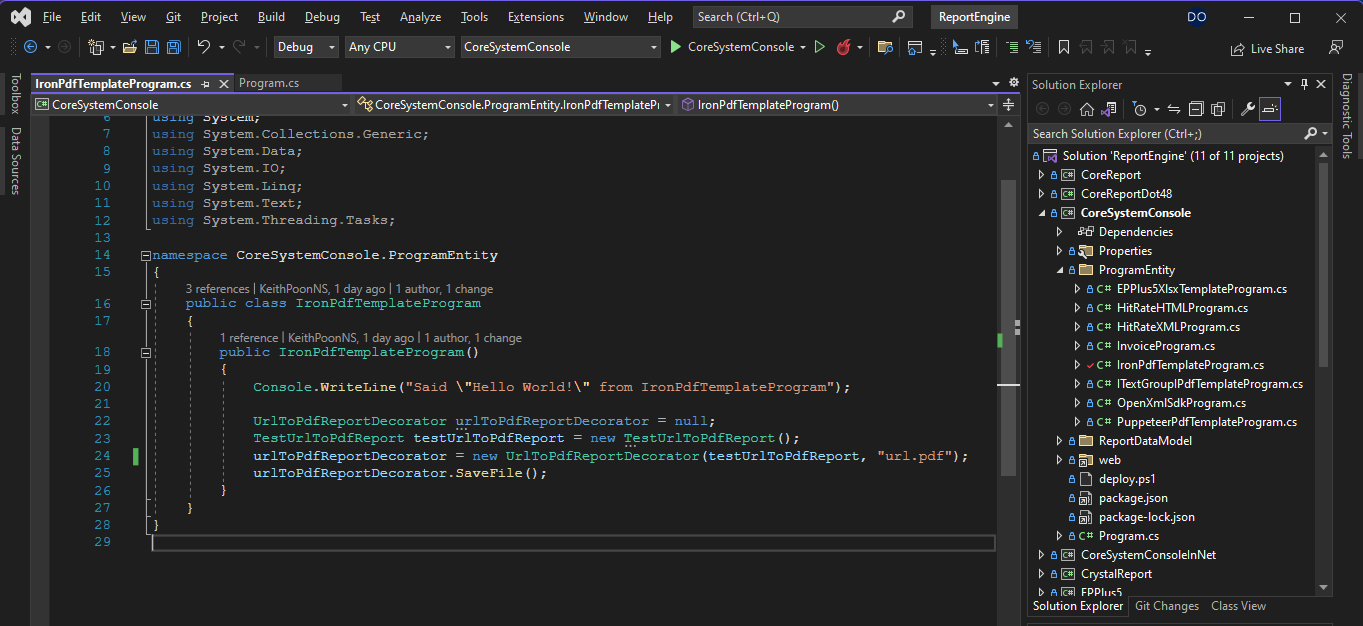
You could define and overwrite the “Template Directory”, “Template File”, Header Setting, Footer Setting, Data set here



## Create program entity

In CoreSystemConsole project, ProgramEntity folder, create a program that will initialize the report instance and report decorator

I created IronPdfTemplateProgram.cs to test the above Iron programs.

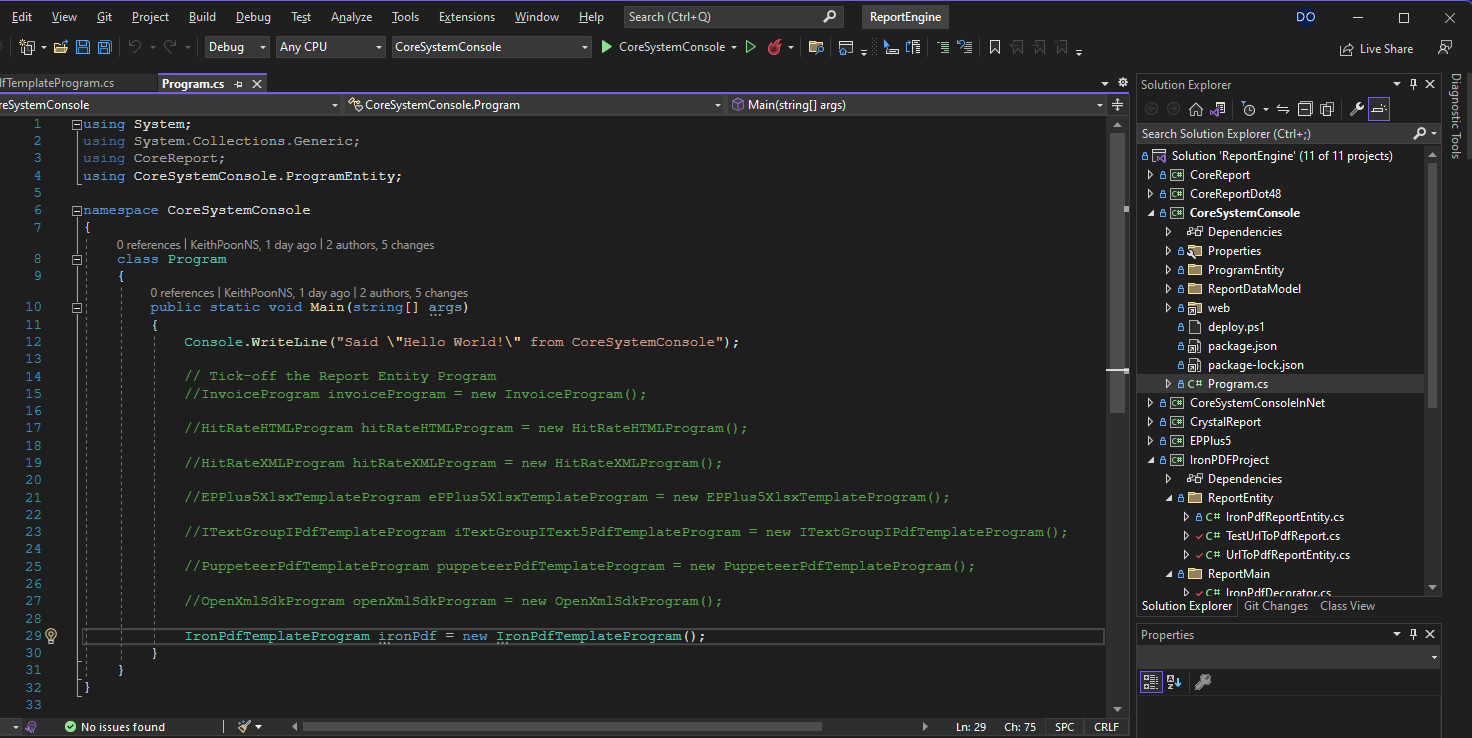


## Update CoreSystemConsole main program

In the root of CoreSystemConsole project, open and update program.cs

Add and call IronPdfTemplateProgram

IronPdfTemplateProgram ironPdf = new IronPdfTemplateProgram();



# Solution Structure

The solution split into two parts, a part is .Net framework 4.8, another part is .Net Core 5

|  |  |  |
| --- | --- | --- |
| C# version | .Net Framework 4.8 | .Net Core 5 |
| Main Project | CoreSystemConsoleInNet  CoreReportDot48 | CoreSystemConsole  CoreReport |
| Report engine project | CrystalReport | JasperReport  EPPlus5  ITextGroupNV  Puppeteer  IronPDFProject  OpenXmlSDK |

There is a story because Crystal Report does not support .Net Core. For the details please read

According to this article: <https://answers.sap.com/questions/13029137/crystal-reports-for-visual-studio-and-net-core-5-a.html>

Because of some low level reasons, Crystal Report must rely on .Net Framework, not planning to move go to support .Net core in future and now

If you must use Crystal Report for your .Net Core application

The offical solution was make your Crystal Report in a .net framework solution as a proxy server (API service)

for the walkthrough: <https://medium.com/scrum-and-coke/view-crystal-report-in-pdf-with-angular-and-asp-net-rest-api-1d6c72168e7c>

solution example: <https://github.com/workcontrolgit/CrystalReportWebAPI>

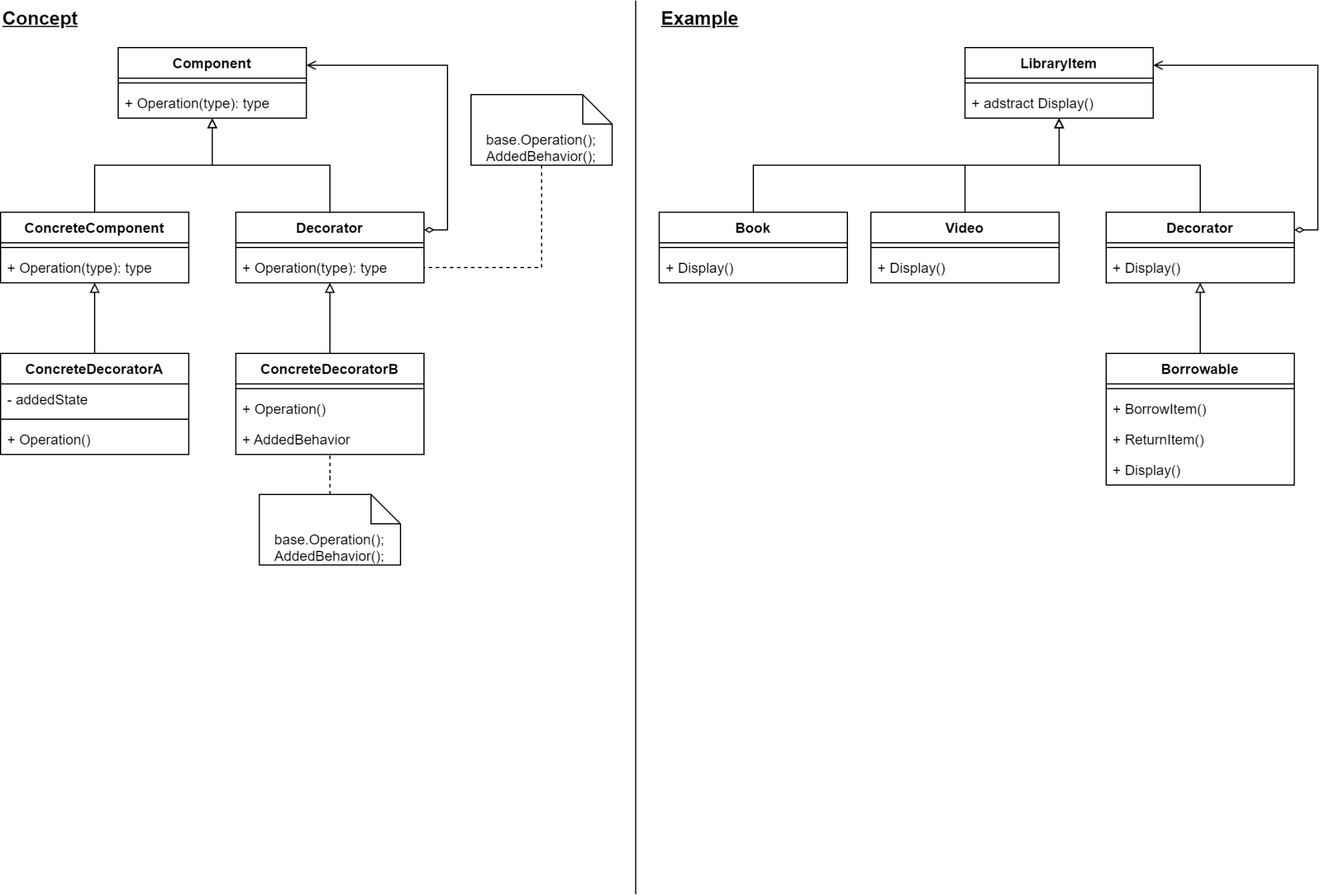
## Design Pattern

I use "Design Pattern - Decorator" to separate the coding files that serve different reporting engines.

Let's said a system contains many functions, a report function represented by a menu item in navigation menu.  
In general, a report function provides the selection criteria, user select the criteria  
Then, click "Export Xlsx" or "Export Pdf" button to generate report file in xlsx, pdf as they want.

"Decorator" Design Pattern gives a report program easy to switch the report enginer, also allows different reports use various engines in a single system

<https://www.dofactory.com/net/decorator-design-pattern#realworld>



## Block Diagram

