# COMP3304 Advanced Object Oriented Programming

## Portfolio Design Documentation

Team Name:

'); DROP TABLE "Teams"; --

## Introduction

Currently Object Orientated Programming is used to solve many software developing problems. This project contains analysis and process of creating a software. Throughout the process a software should be created while complying with client’s requirements and OO(Object-orientated) programming standards. This document contains:

Contents

[1. Introduction 2](#_Toc34349183)

[2. Problem Analysis 2](#_Toc34349184)

[3. OO Software Design 6](#_Toc34349185)

[Class diagram: Part 1 6](#_Toc34349186)

[Class diagram: Part 2 6](#_Toc34349187)

[SOLID 7](#_Toc34349188)

[Design patterns: 8](#_Toc34349189)

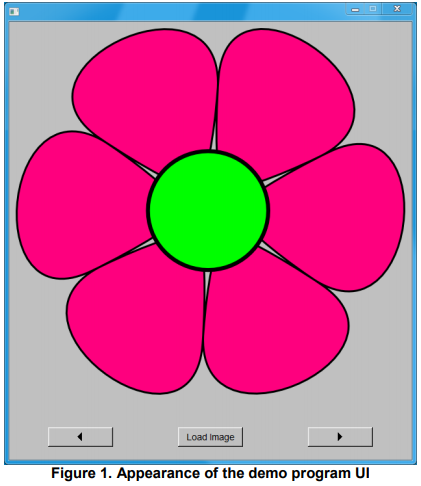
[References 11](#_Toc34349190)

## Problem Analysis

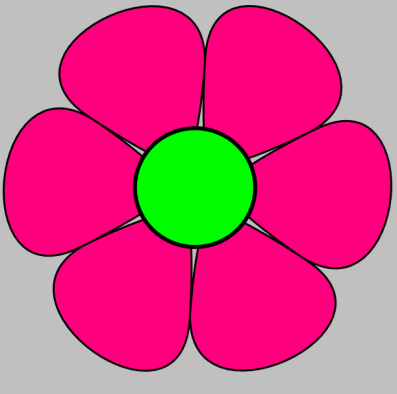
Object Orientated Programming includes problem analysis which must follow client’s brief accordingly. Requirements were proposed by publisher ‘PetSimsRUs’. The project includes creating a software with a major task which is to provide GUI-based image display and manipulation functionality. By breaking down the problem into a smaller task the software must implement:

* the ‘IModel’ C# Interface provided with this brief;
* be able to contain (in memory) a collection of images;
* be able to load one or more images into the collection;
* return a scaled copy of any image in the collection, as a .NET System.Drawing.Image;
* be accompanied with a Windows Forms demo program that makes use of IModel to load and display images, in order to demonstrate compliance of a - d above.

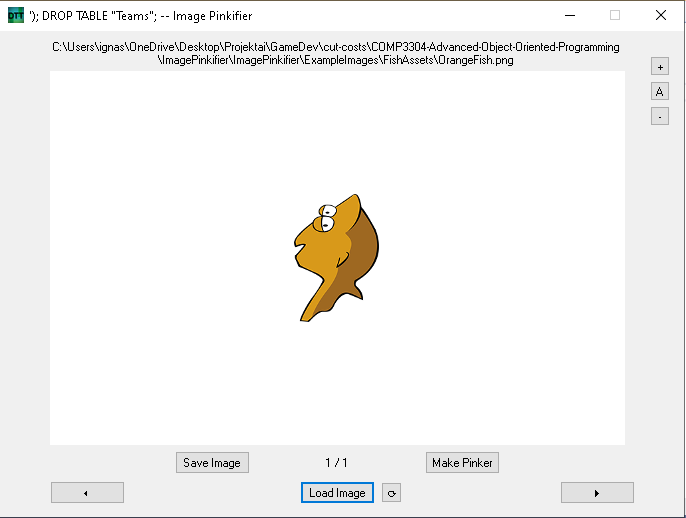
The major goal is to create a fully working software in Visual Studio using C# which complies with S.O.L.I.D. principles to achieve loose coupling and tight cohesion. Also, it must be written according to "COMP3304 Coding Standard" (Dr Price, M. 2020).

According to brief requirements the software might look like the following image:

This demo includes:

* Left button 
* Right button 
* Load Image button 
* Image Placeholder 

How implemented software looks like:

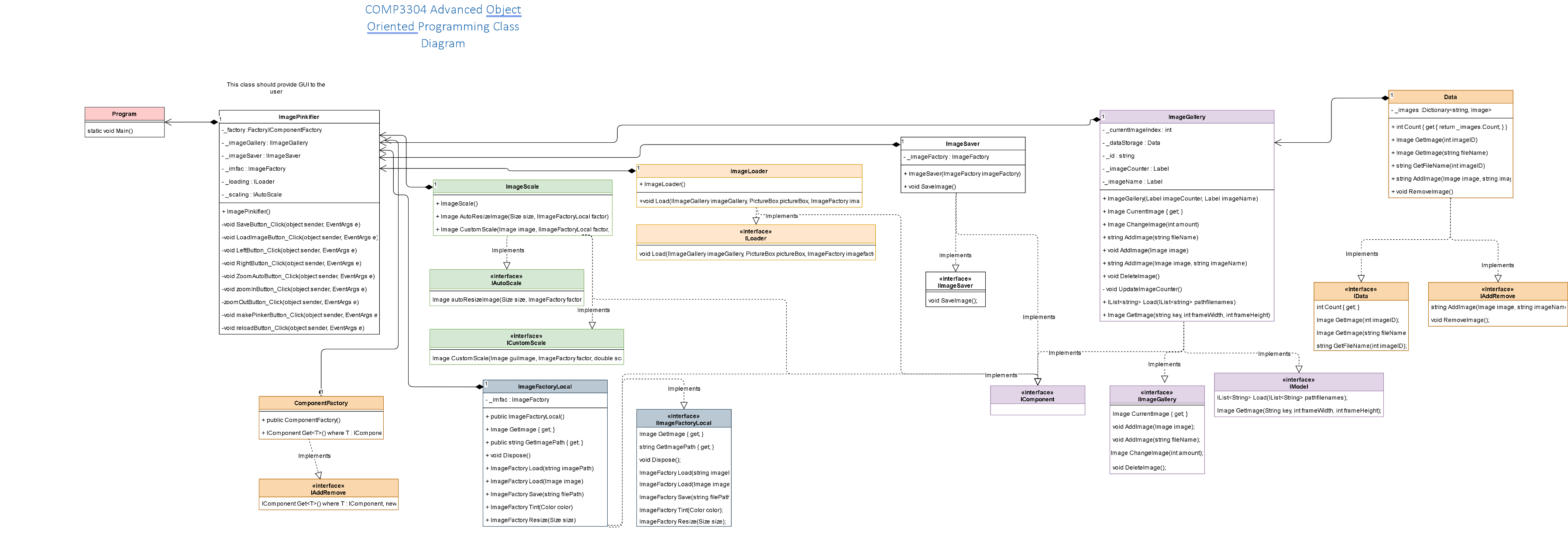


1. Team’s name and software name. 
2. Path file name 
3. Left button 
4. Right button 
5. Save Image button 
6. Load Image button 
7. Reload Image button 
8. Zoom-In image button 
9. Auto scale image button 
10. Zoom-out image button 
11. Make Pinker button 
12. Indicator of which image is selected out of total image count 

After pressing “Load Image” or “Save Image” ‘file chooser’ pop-up should appear which would help to direct either which files to load or where to save the image.

## OO Software Design

## Class diagram: Part 1



## Class diagram: Part 2

### SOLID

#### Single responsibility principle

According to a book “SOLID Principles Succinctly” “A class should have only one responsibility. Let’s say our class is responsible for saving data. That means it should not also be responsible for retrieving data or any other tasks.” (Kumar Arora, 2016).

Currently, classes in the software are being dedicated one thing at the time:

|  |  |
| --- | --- |
| Class Name | Purpose |
| ComponentFactory | Creates components |
| Data | Stores Image related data |
| ImageFactoryLocal | Stores ImageFactory so it could be implemented |
| ImageGallery | Stores a gallery of images with related information |
| ImageLoader | Loads images from pathfile |
| ImagePinkifier | Provides UI (user-interface) |
| ImageSaver | Saves an image to a file using a file dialog |
| ImageScale | Returns a scale version of image |
| Program | Run the software |

#### Open/Closed principle

“Once a class has been written, it should not allow anyone to make changes. No one should be able to go back and amend the class code in order to implement new functionalities.” (Kumar Arora, 2016). Classes only do one thing so to implements new functionalities new classes should be created if any requirements are needed.

#### Liskov substitution principle

“A dependent object should be able to use any object of type parent object.” (Kumar Arora, 2016). The software’s features do not require to use inheritance.

#### Interface segregation principle

“A smaller interface is recommended. If an interface has one method, there will be only one place to change if we need to change the code.” (Kumar Arora, 2016).

Methods in the interfaces were using as minimal parameters as possible to keep tight coupling.

#### Dependency inversion principle

Put simply, this addresses loose coupling. With the help of DIP, we can write code that does not depend upon concrete classes. (Kumar Arora, 2016). Some interfaces depend on interfaces like IImageFactoryLocal because a library did not implement an interface with would violate the S.O.L.I.D. principles and keep tight coupling. Other interfaces mostly include one method.

### Design patterns:

#### Factory pattern

One of the used patterns was factory pattern. This is crucial for projects because it allows flexibility if any of the objects would have to change. By using factory pattern, the instantiation process is encapsulated. ”The factory pattern creates object without exposing the creation logic to the user and it will also refer to the new created object using interface” (Maleki, S., Fu, C., Banotra, A. and Zong, Z., 2017).

Example structure:

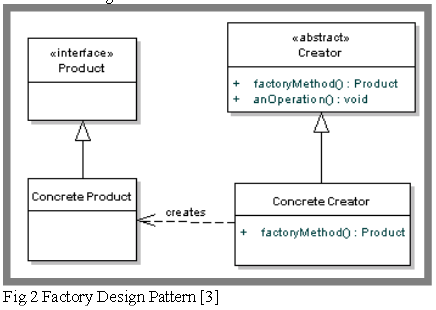
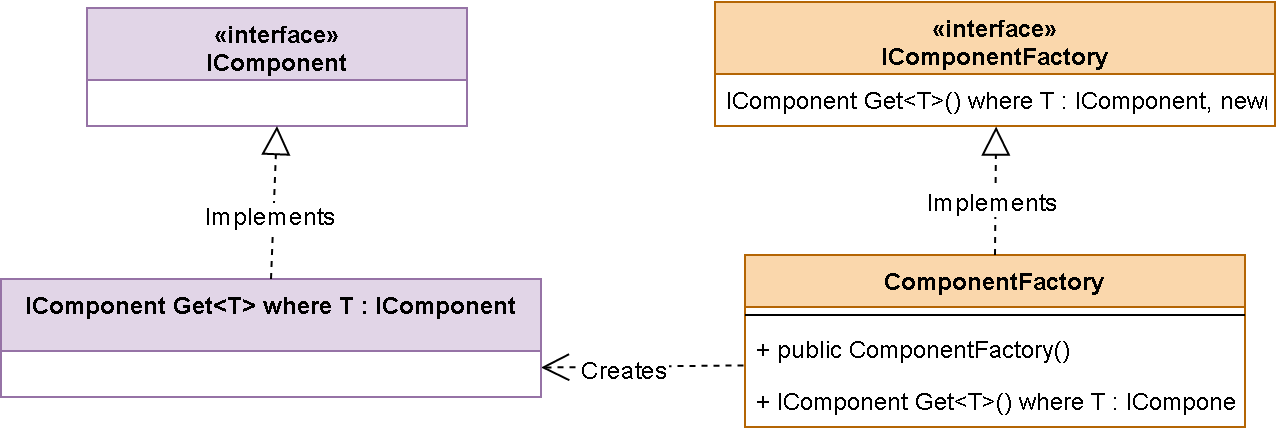


Image 1(Maleki, S., Fu, C., Banotra, A. and Zong, Z., 2017)

Implemented structure:



#### Façade pattern

Façade is another pattern which had to be implemented to create a simplified interface to the user. “Typically this pattern is used to either hide the implementation of the underlying classes it is presenting an interface for, or to simplify the underlying implementation of something that may be complex. A facade may present a simple interface to the outside world, but under the hood do things like create instances of books, manage transactions” (Maleki, S., Fu, C., Banotra, A. and Zong, Z., 2017).

Example structure:

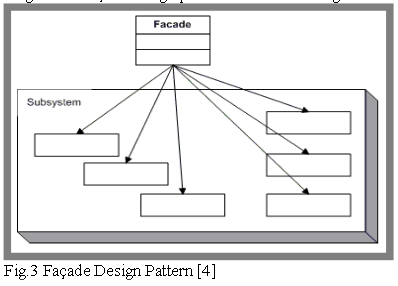
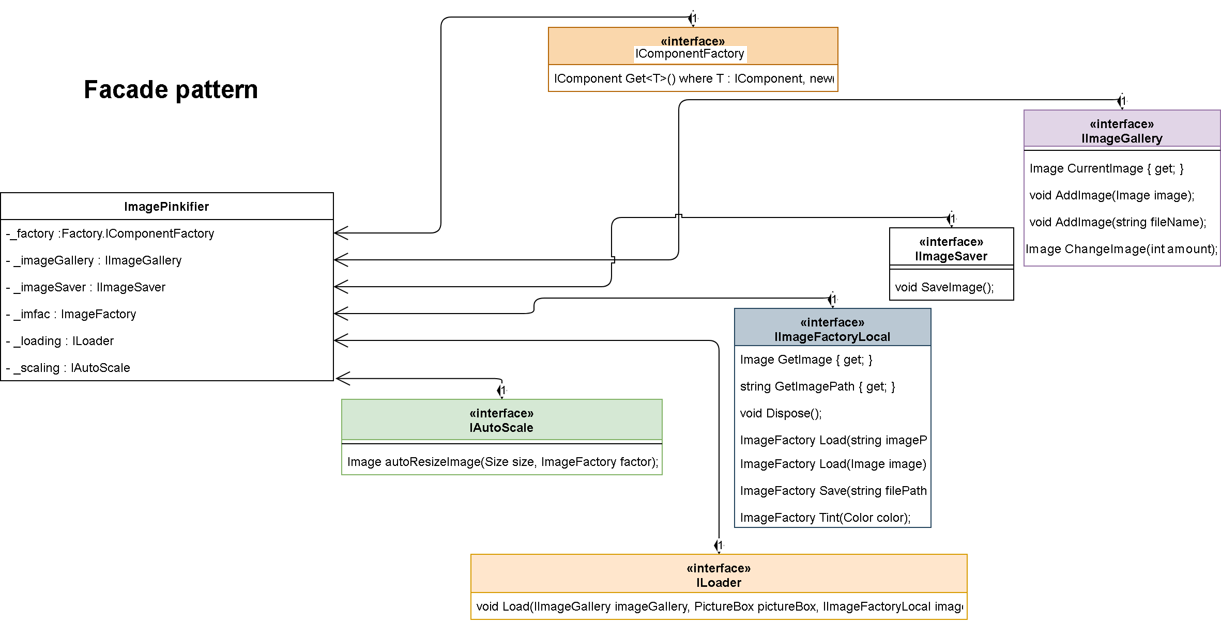


Image 2(Maleki, S., Fu, C., Banotra, A. and Zong, Z., 2017)

Implemented structure:

#### COMP3304 Coding Standard (Dr Price, M. 2020)

The coding standard rules were used to make everything as clear as possible. The following is just a brief description of what should be done.

Fields were commented and described with ‘//’.

Methods, properties, classes were marked with summary ‘///’

Name everything accordingly to code standard (for example \_fields, variable, anotherVariable, Method, Property).

## References

*List your references here using the Harvard referencing style.*

Dr Price, M. (2020). *COMP3304 Advanced Object Oriented ProgrammingCoding Standard*. [online] Worcesterbb.blackboard.com. Available at: https://worcesterbb.blackboard.com/webapps/blackboard/execute/content/file?cmd=view&content\_id=\_1381841\_1&course\_id=\_34561\_1&mode=view#\_1381841\_1 [Accessed 24 Feb. 2020].

Kumar Arora, G. (2016). SOLID Principles Succinctly. Morrisville: Syncfusion Inc., p.26

Maleki, S., Fu, C., Banotra, A. and Zong, Z., (2017). Understanding the impact of object oriented programming and design patterns on energy efficiency. In 2017 Eighth International Green and Sustainable Computing Conference (IGSC) (pp. 1-6). IEEE.