Exercises

This is an exercise on design patterns using Object Oriented stuff.

There is an assumption that you have a notice of the things below:

- *Class (Base and child thru inheritance), Abstract Class, Abstract Method, Method, Property, Indexer, Interface, Public and Private, ….*

The goal of this exercise is to let you think about how software is build up and what the advantages and disadvantages of some solutions and techniques are.

The goal is design patterns. How do you use Object Oriented techniques in your solution.

The goal is to think first and then execute and test.

You can find solutions for this exercise, but you will not learn anything without doing it by yourself.

The code, the solutions and a try out routine can be found in the exercise 00999-a FuzzyDuck and DuckyFuzz Starting Point.zip

Notes

.....

.....

.....

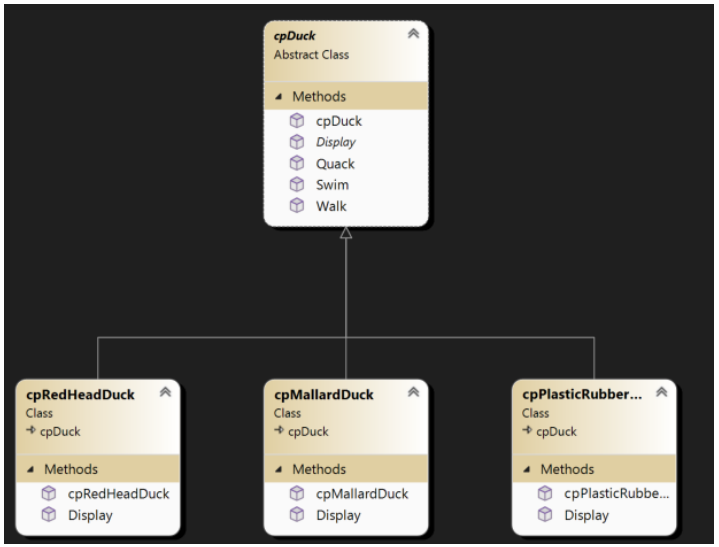
.....

.....

.....

.....

COPY PASTE)



Picture 1: Class Diagram Duck Game.

Notes

Exercise Fuzzy Duck and Ducky Fuzz

Intro

Let's assume you are part of a developer team that has created a game. The name of that game is "Fuzzy Duck".

The game is working, players can play it on PC, PlayStation. You have a lot of clients who pay for it. Everybody (developers, company and clients) are happy.

The game is evolving, so it is subject to change. And we want to be as "lazy" as possible.

There is a DuckGameLibrary. This contains the needed classes. They are used in 2 other projects.

- DuckGameTest. This should contain the test routines.
  There are no tests yet.
- Fuzzy Duck. The actual game that can be played.
  There is no game yet. Imagine there is a functional game.

There is a base abstract class for a Duck (cpDuck) and all the other Ducks inherit from that class.

In the picture beside you see the class diagram (*PICTURE 1*).

Read all the comments in the original code of the DuckGameLibrary. Don't look at the code of the Fuzzy Duck Game. Correct the comments that are wrong. (I know 😊)

The developers has chosen a specific design pattern. It is good enough for the moment. But it will be subject to change (by you).

You are changing an existing program, so be careful in what you do. Make sure you know why you are doing stuff.

Part 01

In the Fuzzy Duck Game there are two startup forms.

- frmDuckGameTryout.
  - Here you will make your own exercise.
- frmDuckGame. This is an implementation of this exercise Part 01.
  - It is just for demoing purposes. Don't look at the code. It is important that you make this exercise by yourself
  - Your solution (battleplan) will be different than mine.

	*Run the program to see what is wanted. See the end goal.* *When you have executed the tasks, you should have something similar.* *Change in Fuzzy Duck Game the cpProgram.Main() routine so that frmDuckGameTryout is shown and not frmDuckGame.* *This screen is empty.*

Your tasks

- Put a combo box on the screen.
- Put a picture box on the screen.
- In the combo box, you show all the class names that inherits from "cpDuck" class.
  - Example 00124-g AssemblyQuery is an example on how to do this.
  - A technique reflection is used.

	*In my solution, I've put the code inside the form.* *This is not the best location of your code. This is on purpose. So don't follow my solution. It is just for demoing purposes.* *Think where you should put your code to find all the needed classes.* *Think on how you get them inside your combo box.*

Notes

- You have your combo box filled with the correct classes.
- When you run your program and you choose an option, show the correct corresponding picture.
- In the picture box, you show the correct picture depending on what you have chosen in the combo box.
- The pictures can be found in the project “DuckGameLibrary” in the folder “DuckDisplays”.
    - Example 00013-g PasswordControl to show an image.

	*For now, we do this hardcoded in the form. We will not use the method “Display()” that is part of cpDuck.* *I say, for now, in the future we will do that. But now we just show the picture using the events of the combo box that shows the correct child classes.*

- Put a label on the screen.
    - Set a good error message in that label when the picture is not found.
- To test your routine, you add an extra child class “cpWoodDuck”. Don’t add a corresponding picture yet.
- Run your routine. All should work, and for “cpWoodDuck” you should see the error message.
- Now add corresponding picture of a Wood Duck in the folder “DuckDisplays”.
- Run your routine again. You should see all the correct pictures.

	*Send your solution to Vincent.* *Your solution will probably be good, but depending on your solution you need to do some correction steps, or not.* *Do not continue before you have confirmation that it is correct.*

Notes

.....

.....

.....

.....

.....

.....

.....

COPY PASTE)

Part 02

- You have a correct solution of part 01.

Don't' continue with this pages, until you have implemented the needed changes in Part 01.

You have received an evaluation of Vincent that your solution is correct.

We will add a test routine in the project "DuckGameTest". This is a console application. We will not actually create a game, we will just create a console application to see if the basic functionality works.

Are the methods starting up when we need it? The methods shows text, not in the console window, but in the immediate window. The reason is that the classes will be used in a game, and there is not a console window present in it.

Your tasks

- Create a list of cpDuck.
- Add an instance of "cpMallardDuck" to it.
- Add an instance of "cpPlasticRubberDuck" to it.
  - Pay attention here, you need to change the access modifier of cpPlasticRubberDuck to be able to use it.
- Add an instance of "cpRedHeadDuck" to it.
- Add an instance of "cpWoodDuck" to it.
- Execute all methods of the Ducks in the list.
  - Display, Quack, Swim and Walk.

Send your solution to Vincent.

Your solution will probably be good, but depending on your solution you need to do some correction steps, or not.

Do not continue before you have confirmation that it is correct.

Notes

Part 03

In the Product Backlog, given by business, it is mentioned that we now want the ducks to fly.

Flying ducks would be perfect for the next release of the game.

- How would you solve it?

	*First think, then execute.* *This is really important, that you first plan what you want to do.*

- Implement the solution.
- Adapt your test routine, so the Fly method is also executed.

	*Send your solution to Vincent.* *Do not continue before you have confirmation that it is correct.*

Notes

.....

.....

.....

.....

.....

.....

.....

COPY PASTE)

Part 04

- This part of the exercise will be sent to you when part 03 is finished.

Notes

.....

.....

.....

.....

.....

.....

.....