GEBZE TECHNICAL UNIVERSITY

Computer Engineer Depertmant

OBJECT ORIENTED ANALYSIS AND DESIGN PATTERN

Ibrahim Cetinkaya

Lesson Instructor
Doc.Dr. Erchan APTOULA

October, 2016
Gebze, KOCAELİ

QUESTION 1:

I think the main concern is that people often have a tendency to abuse design patterns. They learn a few, see the usefulness of them, and without realizing it turn those few patterns into a kind of golden hammer which they then apply to everything.

The key isn't necessarily to learn the patterns themselves. The key is to learn to identify the scenarios and problems which the patterns are meant to address. Then applying the pattern is simply a matter of using the right tool for the job. It's the job that must be identified and understood before the tool can be chosen.

And sometimes it's an odd setup and there is no cut-and-dry pattern to solve it right out of the box. At that point more attention needs to be given to the problem rather than the solution. Break it up into component problems, identify areas of the problem which share common traits with problems addressed by known patterns, adjust the patterns to fit the problem, etc.

Design principles are core abstract principles which we are supposed to follow while designing software architect. Remember they aren't concrete; rather abstract. They can be applied on any language, on any platform regardless of the state as long as we are within the permissible conditions.

QUESTION 2:

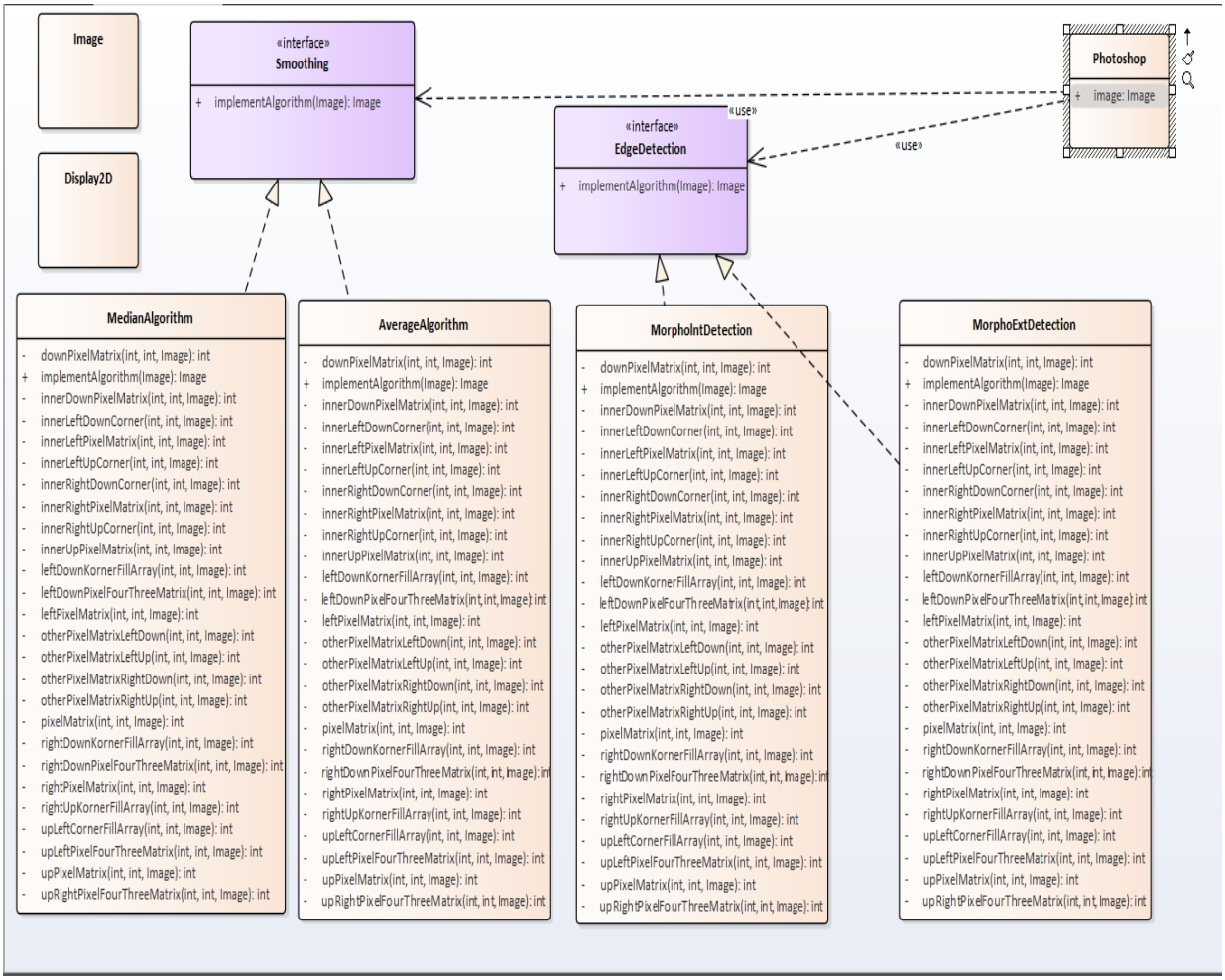
Image
Display2D
«interface»
Smoothing
+ implementAlgorithm(Image): Image
«interface»
EdgeDetection
+ implementAlgorithm(Image): Image
«use»
«use»
Photoshop
+ image: Image
MedianAlgorithm
- downPixelMatrix(int, int, Image): int
+ implementAlgorithm(Image): Image
- innerDownPixelMatrix(int, int, Image): int
- innerLeftDownCorner(int, int, Image): int
- innerLeftPixelMatrix(int, int, Image): int
- innerLeftUpCorner(int, int, Image): int
- innerRightDownCorner(int, int, Image): int
- innerRightPixelMatrix(int, int, Image): int
- innerRightUpCorner(int, int, Image): int
- innerUpPixelMatrix(int, int, Image): int
- leftDownKornerFillArray(int, int, Image): int
- leftDownPixelFourThreeMatrix(int, int, Image): int
- leftPixelMatrix(int, int, Image): int
- otherPixelMatrixLeftDown(int, int, Image): int
- otherPixelMatrixLeftUp(int, int, Image): int
- otherPixelMatrixRightDown(int, int, Image): int
- otherPixelMatrixRightUp(int, int, Image): int
- pixelMatrix(int, int, Image): int
- rightDownKornerFillArray(int, int, Image): int
- rightDownPixelFourThreeMatrix(int, int, Image): int
- rightPixelMatrix(int, int, Image): int
- rightUpKornerFillArray(int, int, Image): int
- upLeftCornerFillArray(int, int, Image): int
- upLeftPixelFourThreeMatrix(int, int, Image): int
- upPixelMatrix(int, int, Image): int
- upRightPixelFourThreeMatrix(int, int, Image): int
AverageAlgorithm
- downPixelMatrix(int, int, Image): int
+ implementAlgorithm(Image): Image
- innerDownPixelMatrix(int, int, Image): int
- innerLeftDownCorner(int, int, Image): int
- innerLeftPixelMatrix(int, int, Image): int
- innerLeftUpCorner(int, int, Image): int
- innerRightDownCorner(int, int, Image): int
- innerRightPixelMatrix(int, int, Image): int
- innerRightUpCorner(int, int, Image): int
- innerUpPixelMatrix(int, int, Image): int
- leftDownKornerFillArray(int, int, Image): int
- leftDownPixelFourThreeMatrix(int, int, Image): int
- leftPixelMatrix(int, int, Image): int
- otherPixelMatrixLeftDown(int, int, Image): int
- otherPixelMatrixLeftUp(int, int, Image): int
- otherPixelMatrixRightDown(int, int, Image): int
- otherPixelMatrixRightUp(int, int, Image): int
- pixelMatrix(int, int, Image): int
- rightDownKornerFillArray(int, int, Image): int
- rightDownPixelFourThreeMatrix(int, int, Image): int
- rightPixelMatrix(int, int, Image): int
- rightUpKornerFillArray(int, int, Image): int
- upLeftCornerFillArray(int, int, Image): int
- upLeftPixelFourThreeMatrix(int, int, Image): int
- upPixelMatrix(int, int, Image): int
- upRightPixelFourThreeMatrix(int, int, Image): int
MorphoIntDetection
- downPixelMatrix(int, int, Image): int
+ implementAlgorithm(Image): Image
- innerDownPixelMatrix(int, int, Image): int
- innerLeftDownCorner(int, int, Image): int
- innerLeftPixelMatrix(int, int, Image): int
- innerLeftUpCorner(int, int, Image): int
- innerRightDownCorner(int, int, Image): int
- innerRightPixelMatrix(int, int, Image): int
- innerRightUpCorner(int, int, Image): int
- innerUpPixelMatrix(int, int, Image): int
- leftDownKornerFillArray(int, int, Image): int
- leftDownPixelFourThreeMatrix(int, int, Image): int
- leftPixelMatrix(int, int, Image): int
- otherPixelMatrixLeftDown(int, int, Image): int
- otherPixelMatrixLeftUp(int, int, Image): int
- otherPixelMatrixRightDown(int, int, Image): int
- otherPixelMatrixRightUp(int, int, Image): int
- pixelMatrix(int, int, Image): int
- rightDownKornerFillArray(int, int, Image): int
- rightDownPixelFourThreeMatrix(int, int, Image): int
- rightPixelMatrix(int, int, Image): int
- rightUpKornerFillArray(int, int, Image): int
- upLeftCornerFillArray(int, int, Image): int
- upLeftPixelFourThreeMatrix(int, int, Image): int
- upPixelMatrix(int, int, Image): int
- upRightPixelFourThreeMatrix(int, int, Image): int
MorphoExtDetection
- downPixelMatrix(int, int, Image): int
+ implementAlgorithm(Image): Image
- innerDownPixelMatrix(int, int, Image): int
- innerLeftDownCorner(int, int, Image): int
- innerLeftPixelMatrix(int, int, Image): int
- innerLeftUpCorner(int, int, Image): int
- innerRightDownCorner(int, int, Image): int
- innerRightPixelMatrix(int, int, Image): int
- innerRightUpCorner(int, int, Image): int
- innerUpPixelMatrix(int, int, Image): int
- leftDownKornerFillArray(int, int, Image): int
- leftDownPixelFourThreeMatrix(int, int, Image): int
- leftPixelMatrix(int, int, Image): int
- otherPixelMatrixLeftDown(int, int, Image): int
- otherPixelMatrixLeftUp(int, int, Image): int
- otherPixelMatrixRightDown(int, int, Image): int
- otherPixelMatrixRightUp(int, int, Image): int
- pixelMatrix(int, int, Image): int
- rightDownKornerFillArray(int, int, Image): int
- rightDownPixelFourThreeMatrix(int, int, Image): int
- rightPixelMatrix(int, int, Image): int
- rightUpKornerFillArray(int, int, Image): int
- upLeftCornerFillArray(int, int, Image): int
- upLeftPixelFourThreeMatrix(int, int, Image): int
- upPixelMatrix(int, int, Image): int
- upRightPixelFourThreeMatrix(int, int, Image): int

QUESTION 3:

I applied the behavior by defining the interface design strategy. The strategy pattern is the classic case of "prefer composition over inheritence". Rather than modifying the code at compile time, reference the behavior from a variable Strategy pattern allows you to move behavior into separate class which is good by principle 'Strategy' - single responsibility. Image that we need to smoothing and edge detection. you can inject desired image strategy (smoothing, edge detection) without having each image even know about it. Interface segregation is very easy.It just make Smoothing interface make many implementations of smoothing for image and assign EdgeDetection property to desired image class