Assignment 4 - Group 24 - BDSA

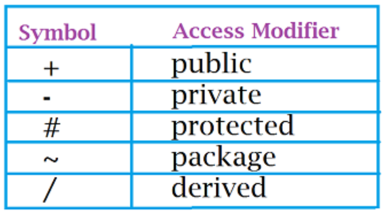
emtj

September 2022

Exercise 1

Recapitulate the meaning of encapsulation, inheritance, and polymorphism in object-oriented programming. Provide a description of these concepts including UML diagrams to illustrate your descriptions.

encapsulation means hiding data and the methods working with that data inside the same component, or class in C#. This is used to protect or hide a class data or state from the outside world and helps to avoid a lot of unfortunate buds. Each class can hide their data and method, with the use of access modifiers. this is shown in a UML Diagram with + for public data or method, and a - for private data or method.



Symbol	Access Modifier
+	public
-	private
#	protected
~	package
/	derived

Figure 1: encapsulation shown with access modifiers

inheritance in object-oriented programming describes the way that classes inherit from each other. inheritance is a mechanism where you can to derive a class from another class for a hierarchy of classes that share a set of attributes and methods. The class that other classes inherit from is called a superclass. In a UML diagram inheritance is shown with arrow like in 2

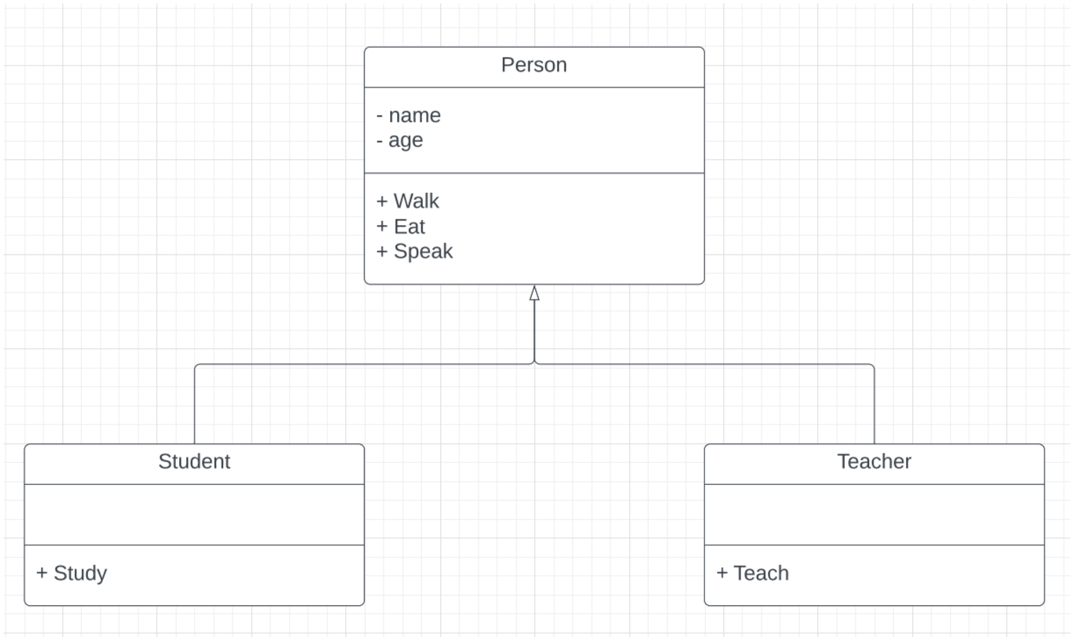


Figure 2: inheritance

polymorphism means many forms. In OOP it means that a class can have a different way to do the same thing as its superclass. polymorphism is shown in a UML diagram with the method in italic like in the figure 3

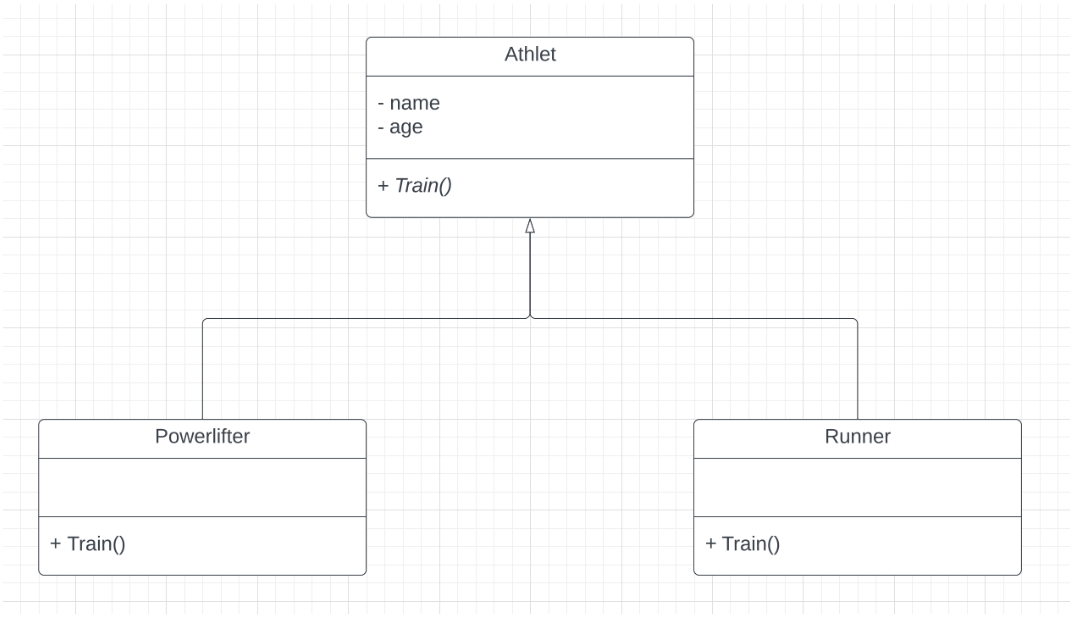


Figure 3: inheritance

Exercise 2

Draw a UML class diagram that illustrates your implementation of the entities of last week's C assignment, see GitHub - Assignment 3 The purpose of the diagram should be to document the main relationships between the entities and their multiplicities.

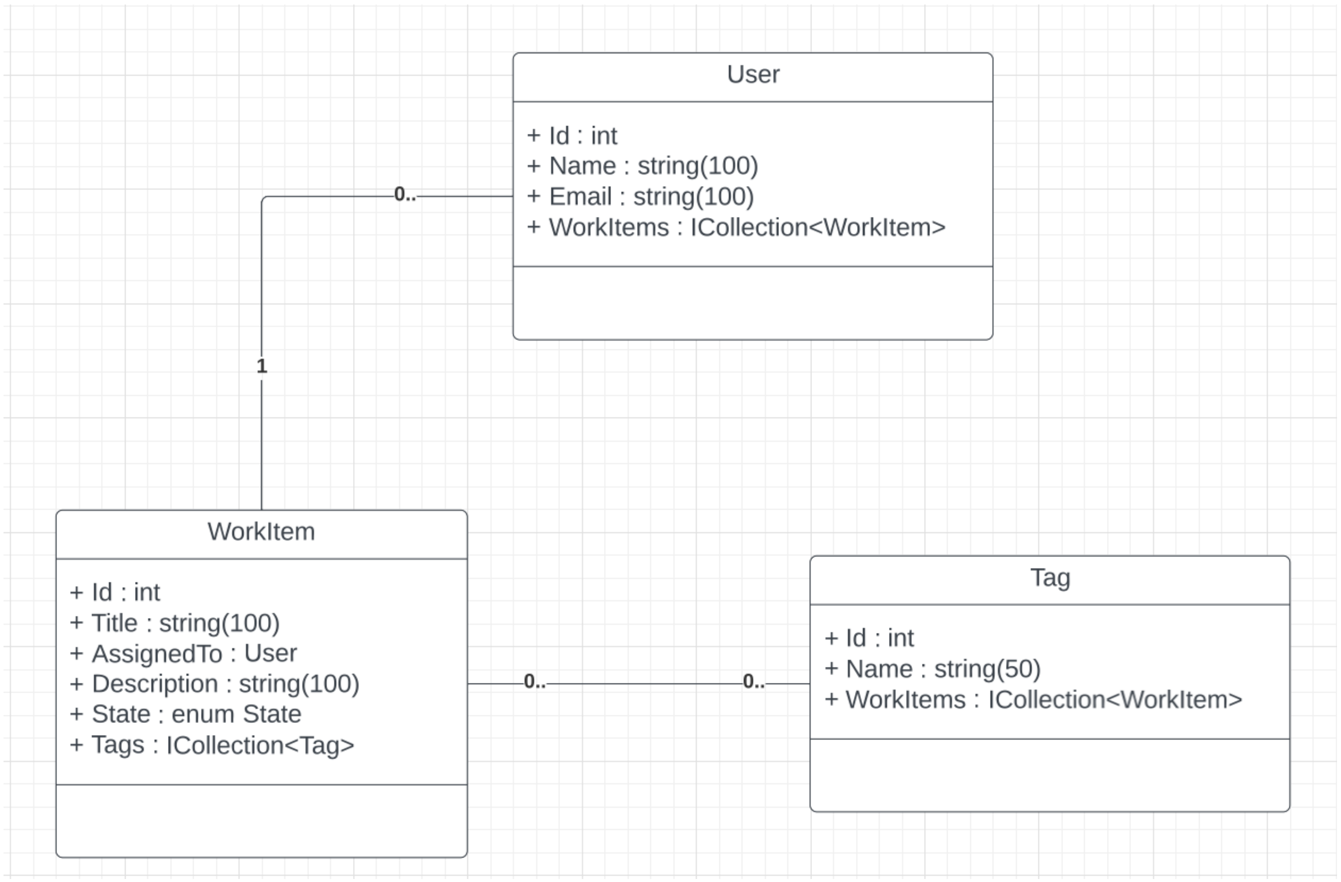


Figure 4: UML Class Diagram

Exercise 3

Draw a UML state diagram that illustrates your implementation of the WorkItem entity from last week's C assignment, see GitHub - Assignment 3. The purpose of the diagram should be to document the different states of the entity and the events that trigger the state changes.

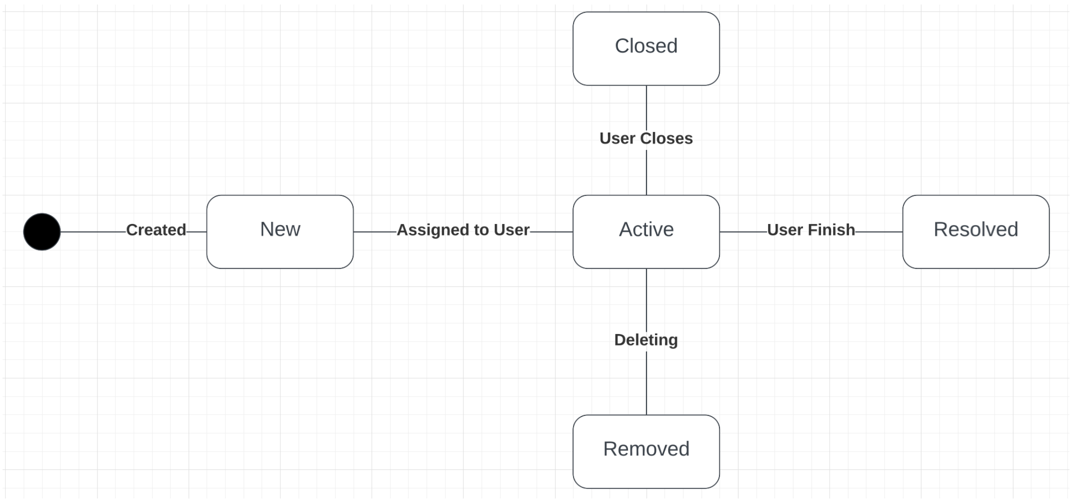


Figure 5: UML State Diagram

Exercise 4 & 5

Exercise 4: For each of the five SOLID design principles, provide an example that illustrates the violation of the specific principle. Your examples can be given either in code or as UML diagrams. Briefly explain under which conditions the respective principle is violated. Note, the examples do not need to be sophisticated.

Exercise 5: For each of the examples of violations of SOLID design principles in Exercise 4, provide a refactored design that respects the respective design principle. Again, the refactored designs can be given either in code or as UML diagrams, briefly explain under which conditions the respective principle is not violated any longer, and remember that the examples do not need to be sophisticated.

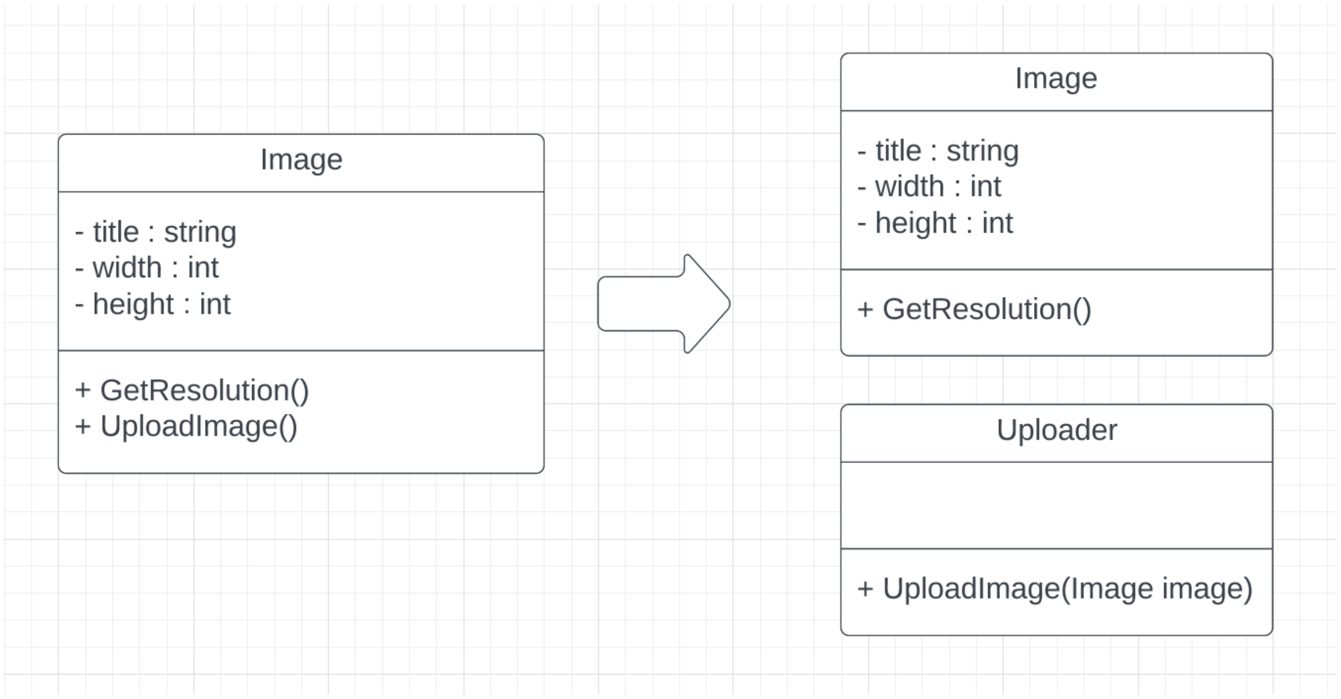


Figure 6: The Single Responsibility Principle, which state the a class should have one and not more responsibility, like shown here the image class should only be responsible for representing a image, and not also uploading this image. This responsibility should be given to another class

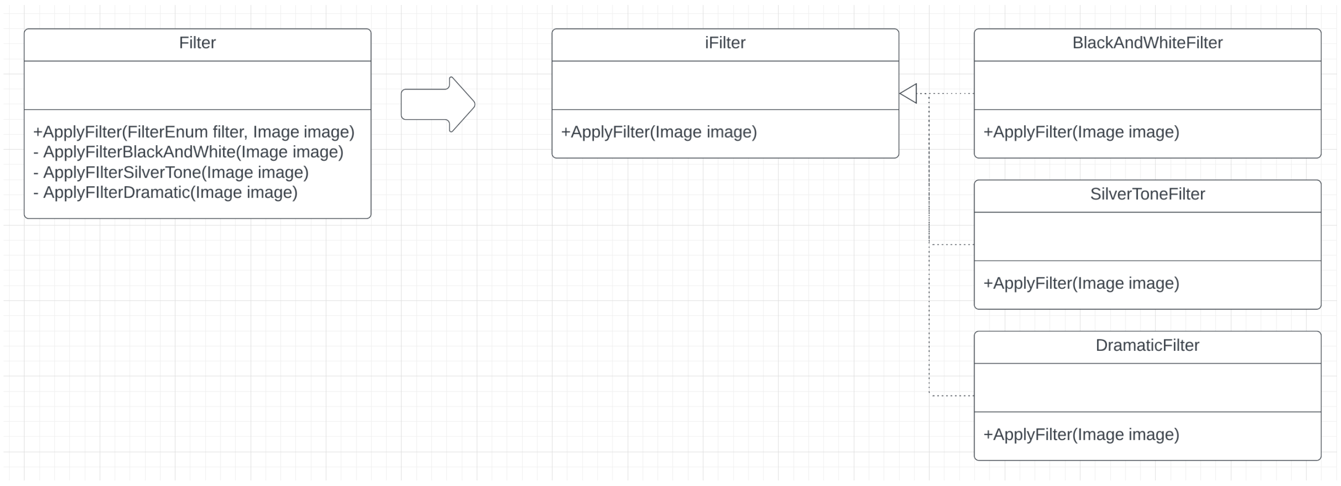

Filter
+ApplyFilter(FilterEnum filter, Image image)
- ApplyFilterBlackAndWhite(Image image)
- ApplyFIlterSilverTone(Image image)
- ApplyFIlterDramatic(Image image)
iFilter
+ApplyFilter(Image image)
BlackAndWhiteFilter
+ApplyFilter(Image image)
SilverToneFilter
+ApplyFilter(Image image)
DramaticFilter
+ApplyFilter(Image image)

Figure 7: The Open Closed Principle, this state that a class should be open for extension, but not for modification, like in this example the filter class is open for modification, if there should ever be the need for adding another filter. This can be resolved by converting the filter into an interface and it is now open for extension but closed for modification

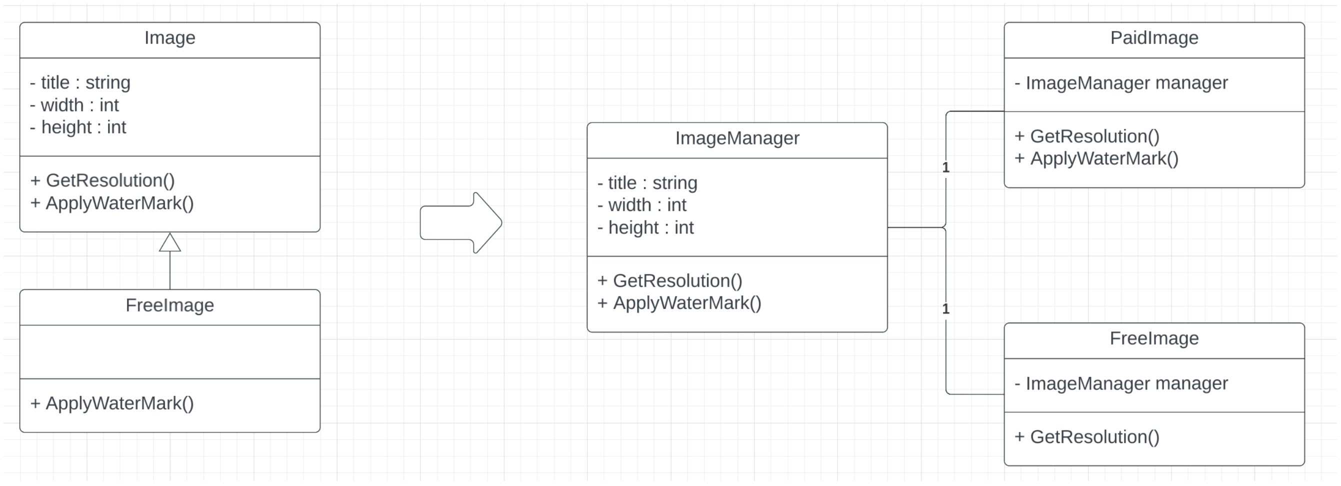


Figure 8: The Liskov Substitution Principle, Subtypes should be replacable for its base type. This is not fulfilled in this example, where an freeImage extends an image. this can be resolved by making an class called imageManager.

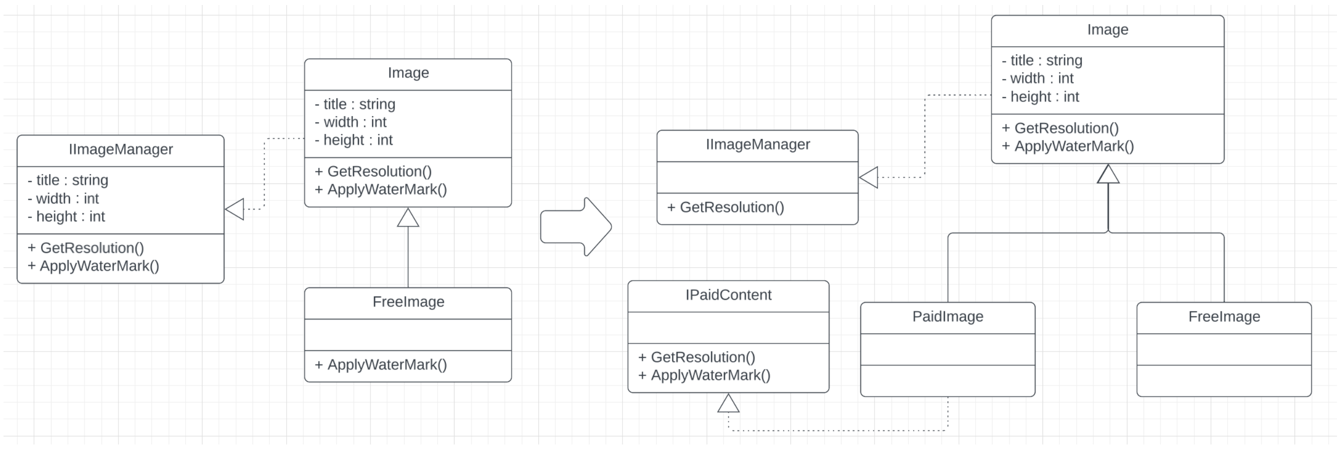


Figure 9: The Interface Segregation Principle

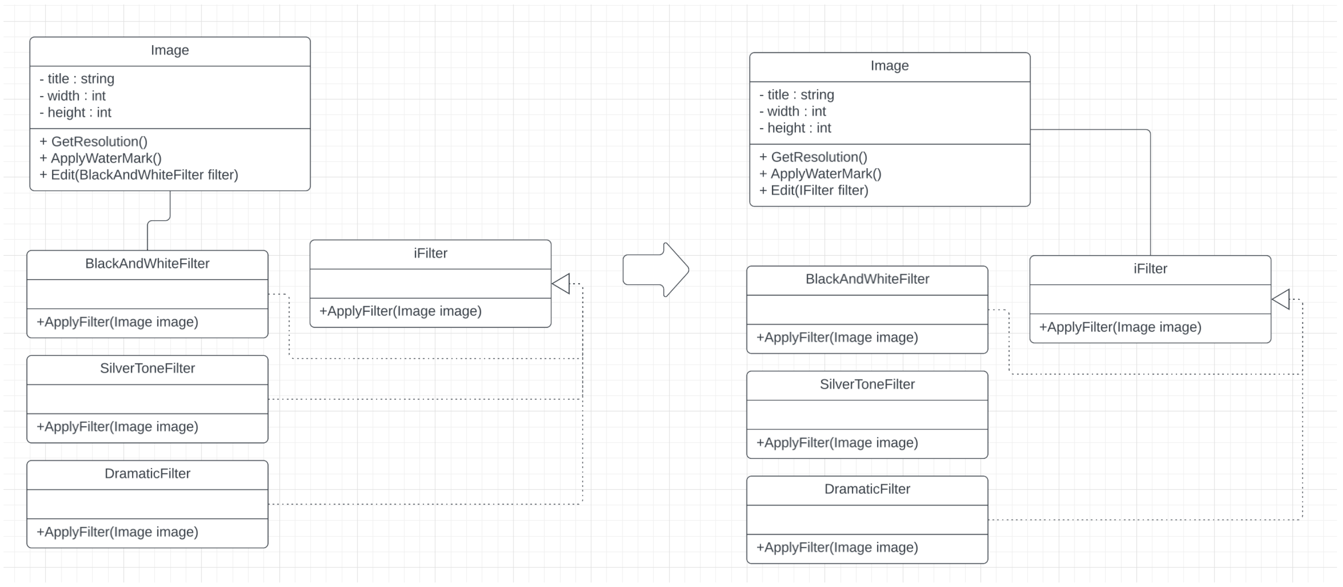


Figure 10: The Dependency Inversion Principle