Assignment 4

Group 5
malj
anlf
samd

September 2021

C#

Link to github repository

Software Engineering

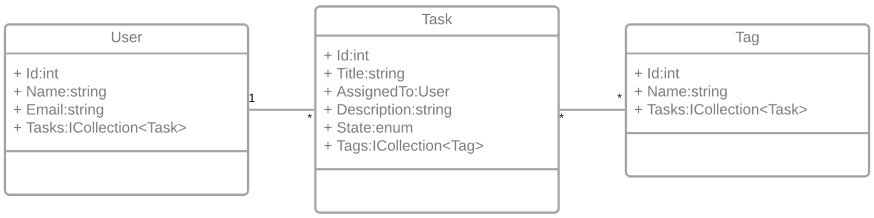
Exercise 1

Study the meaning of encapsulation, inheritance, and polymorphism in OO languages. Provide a description of these concepts including UML diagram showcasing them.

- **Encapsulation:** bundling of data, along with the methods that operate on that data, into a single unit. In Object Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulates them.
- **Inheritance:** The capability of a class to derive properties and characteristics from another class.
- **Polymorphism:** Polymorphism is the ability of any data to be processed in more than one form. The most common use of polymorphism in object-oriented programming occurs when a parent class reference is used to refer to a child class object. Here we will see how to represent any function in many types and many forms.

Exercise 2

As a maintainance activity (i.e., after having written the code): draw a class diagram representing your implementation of the entities for the C sharp part. The purpose of the diagram should be to document the main relationships between the entities and their multiplicity.



Exercise 3

As a maintainance activity (i.e., after having written the code): draw a state machine diagram representing your implementation of the task entity. The purpose of the diagram should be to document the different states the entity can go through and the transitions that trigger the changes.

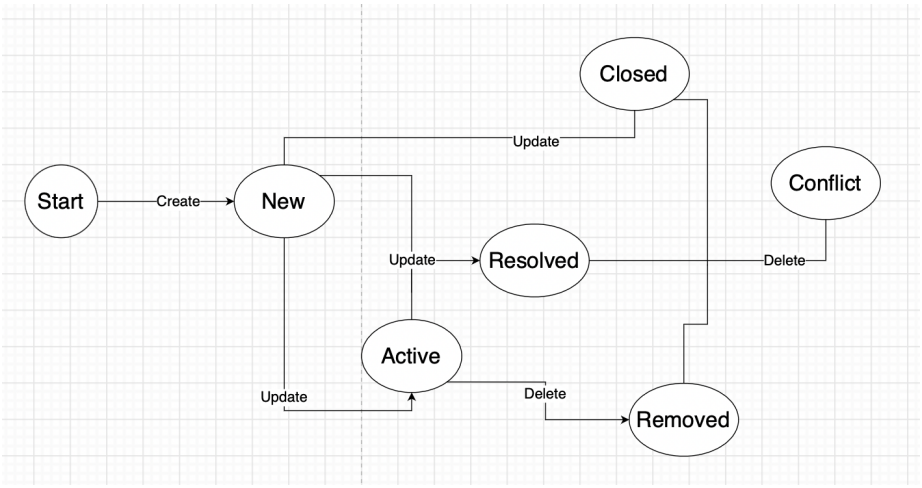


Figure 1: State Machine Diagram

Exercise 4

For each of the SOLID principles, provide an example through either a UML diagram or a code listing that showcases the violation of the specific principle. Note: the examples do not need to be sophisticated.
SOLID principles:

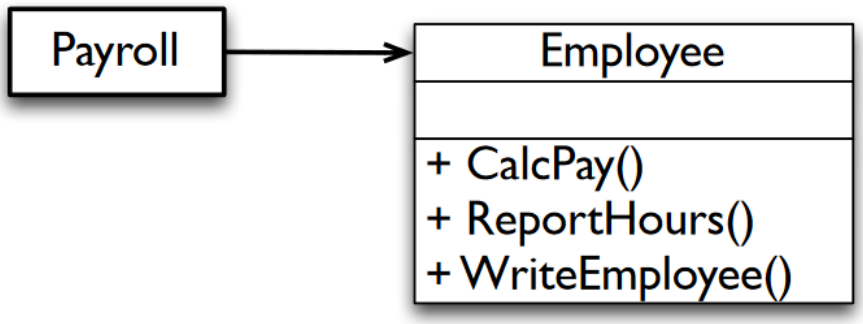


Figure 2: Single responsibility principle

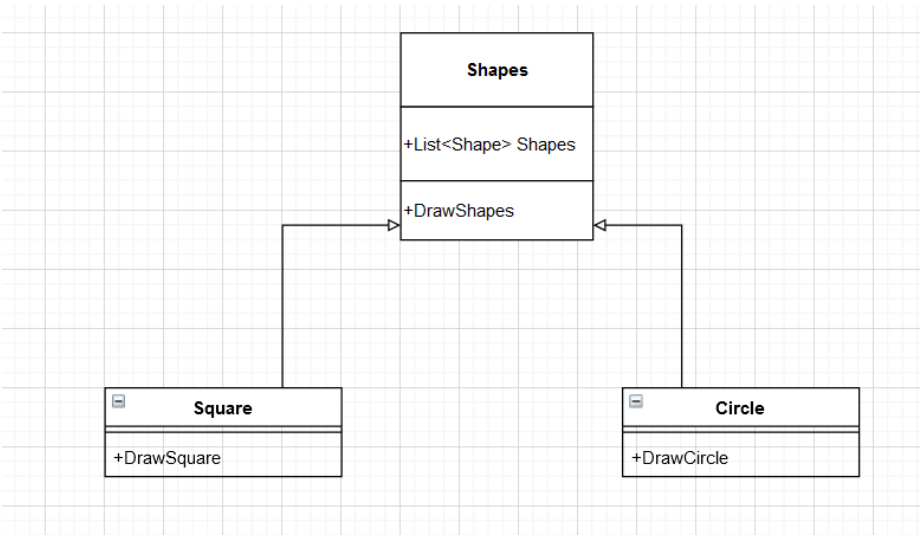


Figure 3: Open/closed principle

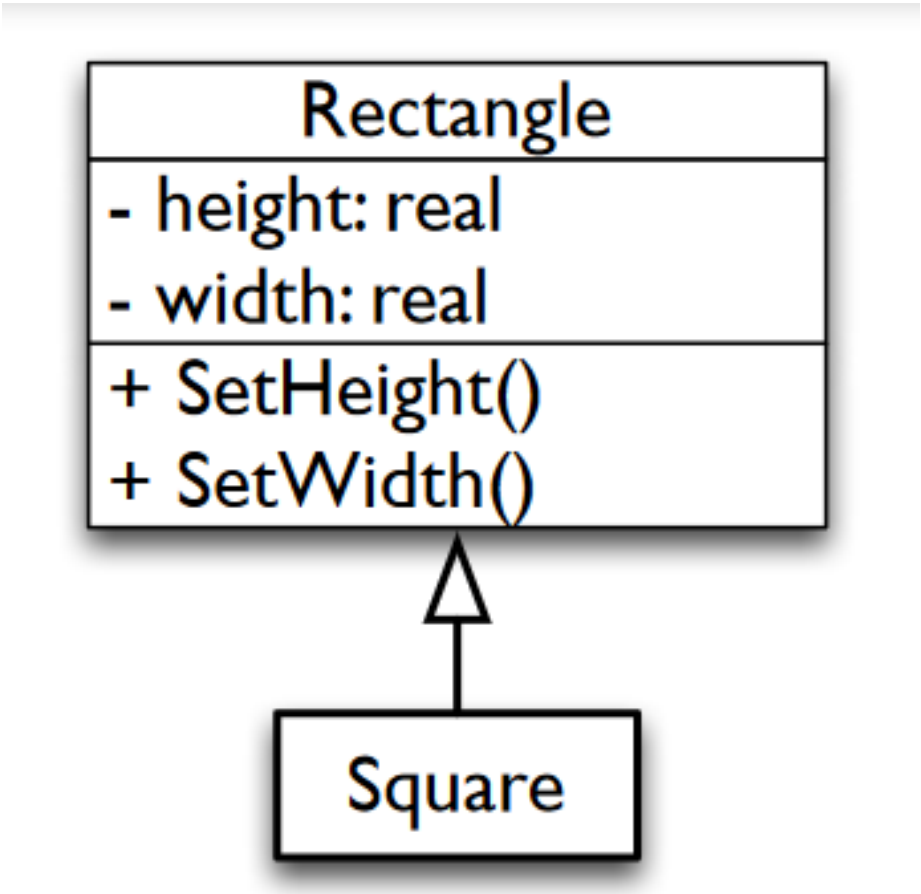


Figure 4: Liskov substitution principle

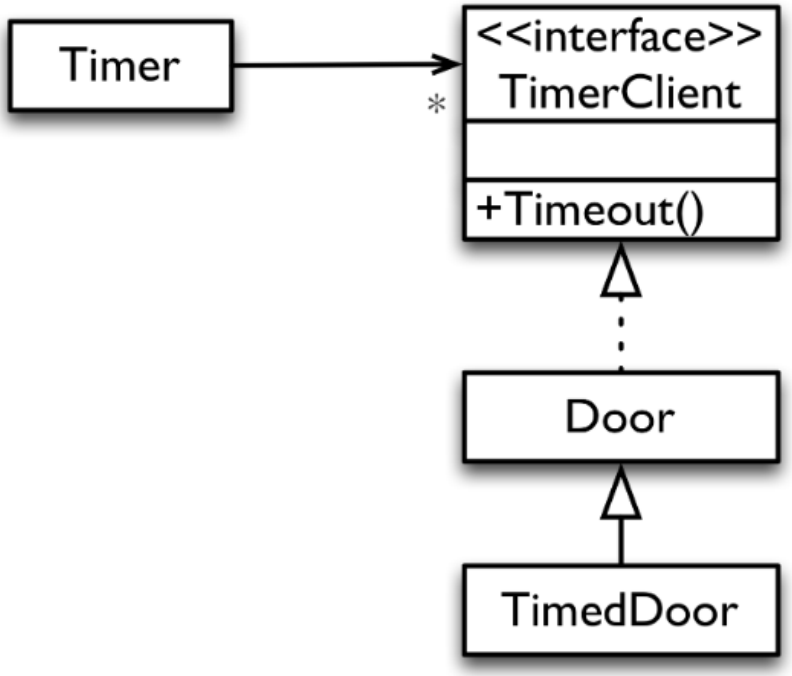


Figure 5: interface segregation principle

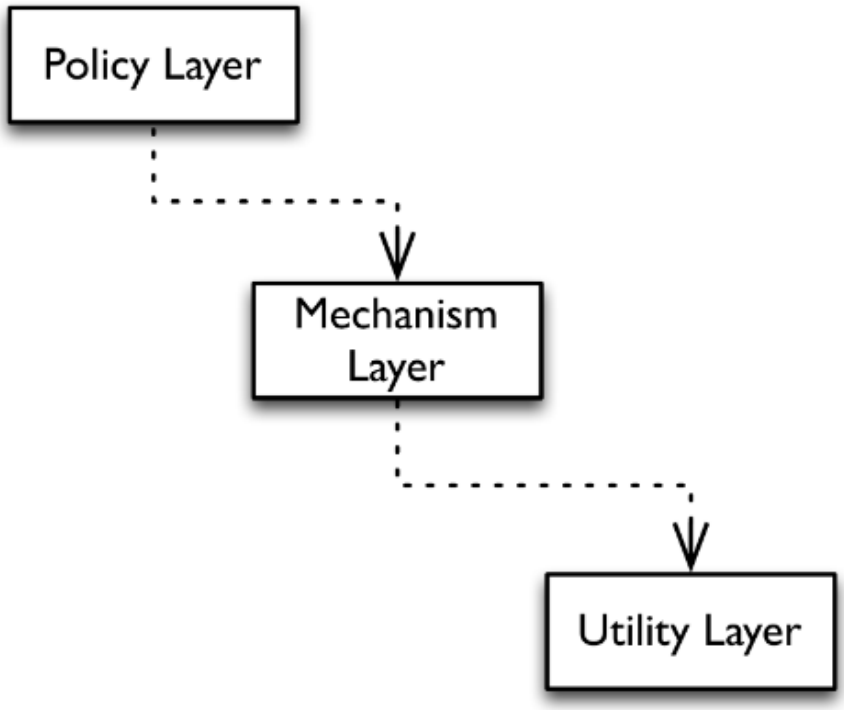


Figure 6: Dependency inversion principle

Exercise 5

For each of the examples provided for SE-4, provide a refactored solution as either a UML diagram or a code listing that showcases a possible solution respecting the principle violeted. Note: the examples do not need to be sophisticated.
SOLID principles:

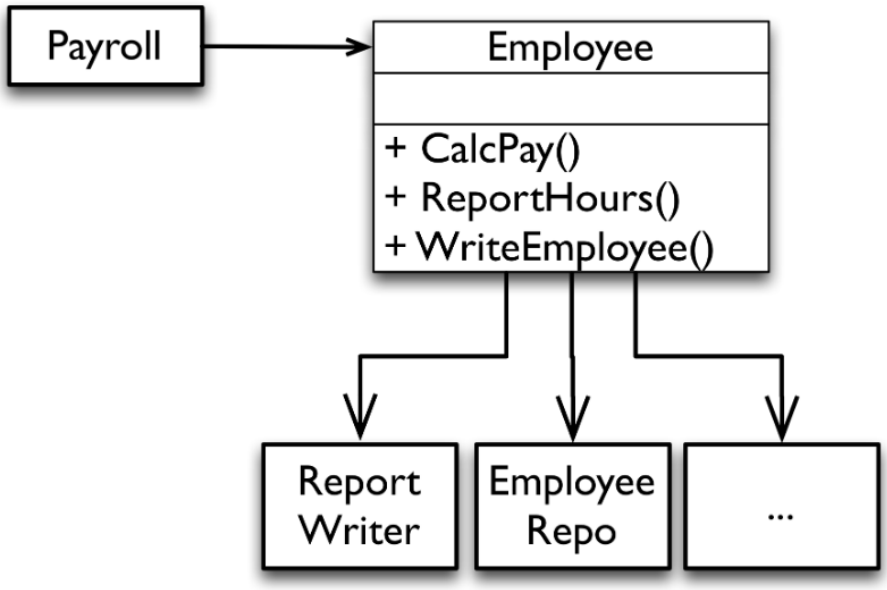


Figure 7: Single responsibility principle

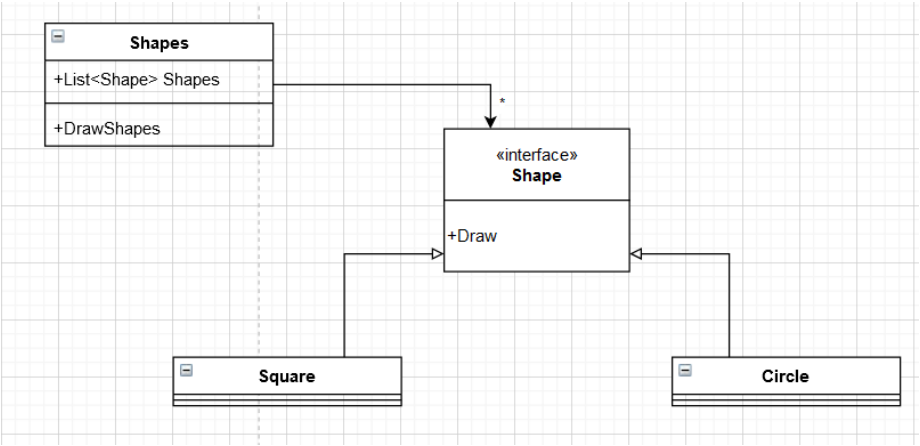


Figure 8: Open/closed principle

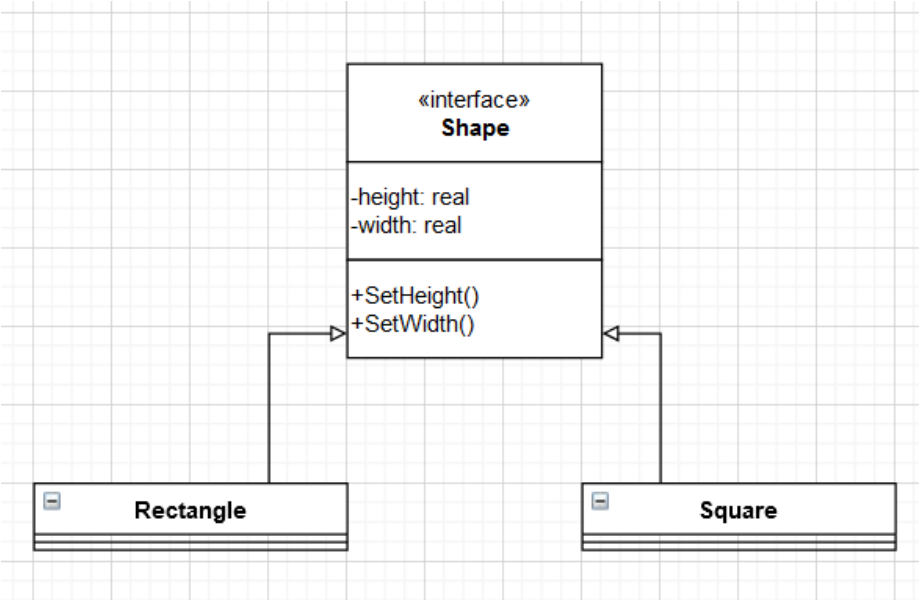


Figure 9: Liskov substitution principle

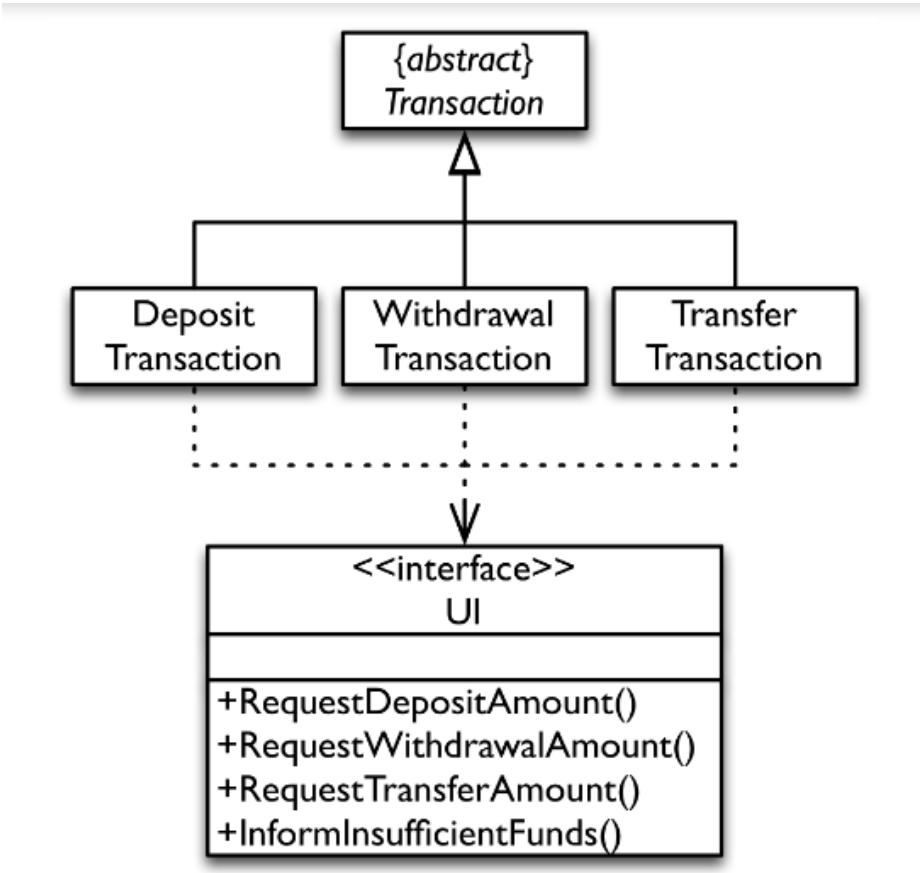


Figure 10: Interface segregation principle

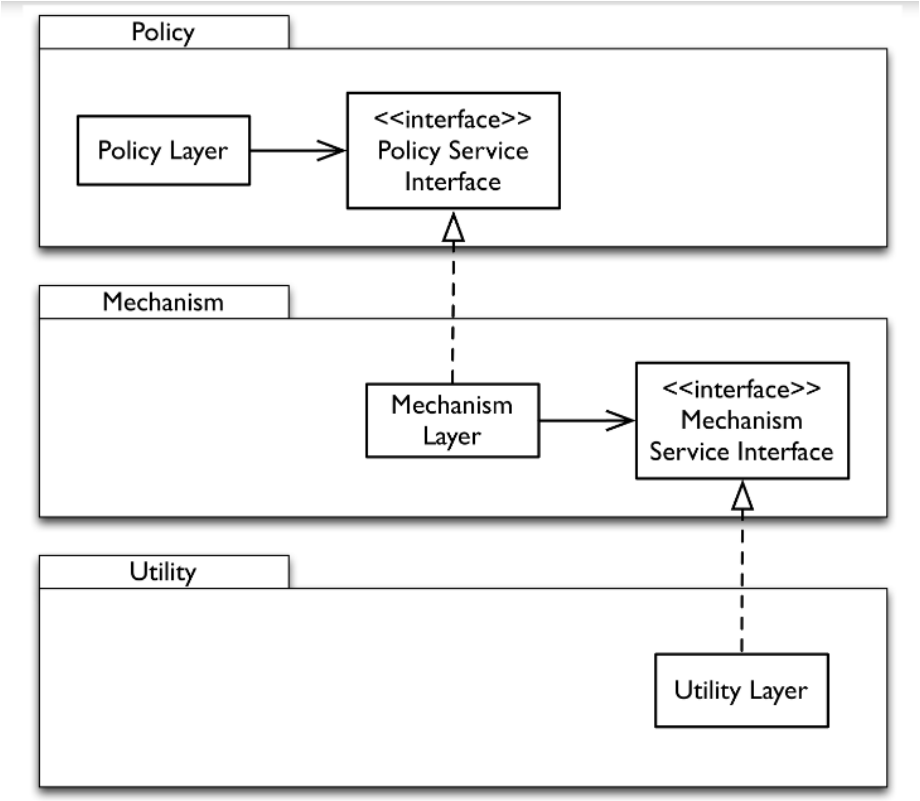


Figure 11: Dependency inversion principle