Eksamen-1400-kandidatummer: 34

May 2021

Oppgave 1

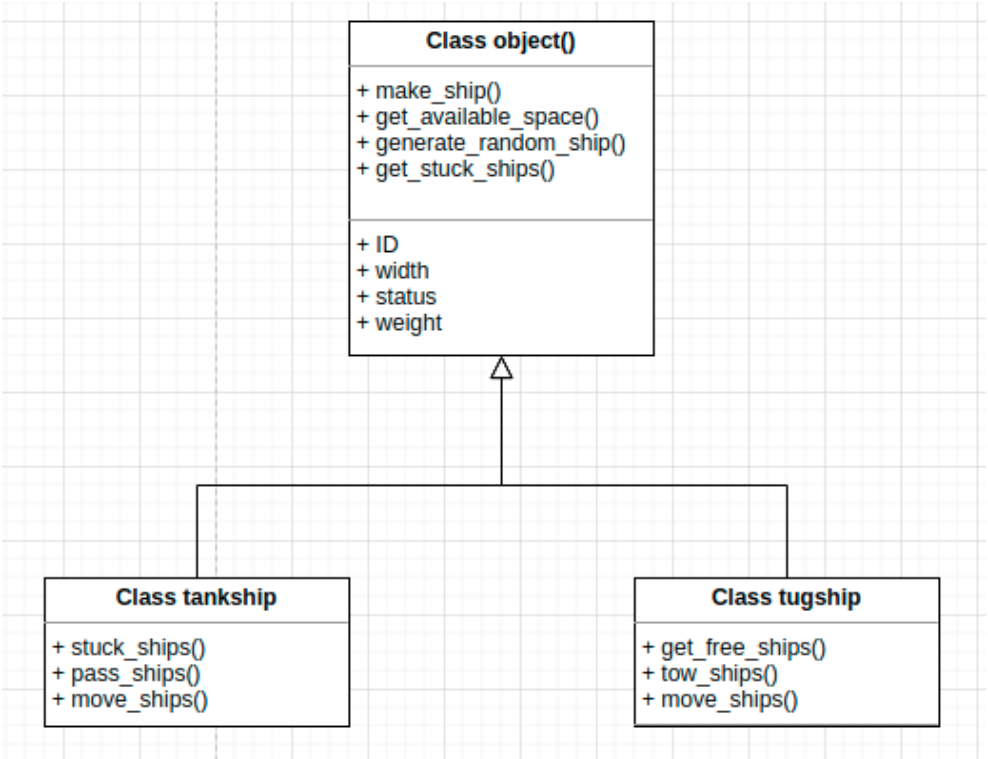

Figure 1: Illustration of an UML.

The UML diagram illustrates the hierarchy of the three created classes. The arrow from the tankship- and tugship -class to the object class shows which direction the inheritance goes. The tankship- and tugship -class inherit the behaviors in the object parent class. Two "move ship" functions in each subclass illustrates polymorphism.

Oppgave 2

The existing code has changed know that we have implemented three new classes. The object class is the parent class and has the behaviors and attributes that are general to each ship. Both sub classes "tankship" and "tugship" inherit these from the object class. In addition, they have specific behaviors that are dedicated to their class and how to behave in the channel. Assumptions we make are as follows: All types of ships are interested in knowing if another ship is stuck, not just tugships. Tankships cannot free other tankships. Tugshisp does not get stuck and all tugships can pass the channel.

Polymorphism is when a class is treated differently when implemented in different subclasses. So the move ships function sets up the possibility for different movement for the two subclasses. Since the tugships cant get stuck, the move ship function will give different movement results for the different classes, even if the function is the same. If different movement trough another function is enough to label this as polymorphism, is uncertain. Polymorphism is here shown in theory in the UML. We have also made an assumption that tugships do not return to the canal after they have run out of it (which would be natural). By putting the move function in different classes we could adjust the movement so that tugships return to the channel and go through the list backwards (we have not done this in the implementation). However, we have implemented a more convenient way to separate the movement for the two classes. We make an assumption that tugboats is faster than tankships in the channel, therefore the tugboats can move two positions at each iteration while tankships only can move one iteration.

Oppgave 3

The code is in the attachment: opggave3.py

Oppgave 4

1)
Since the "gravemaskin" behaves like a boat in every way except that it does not follows the queuing system in the canal, it will be natural to place this in a separate class that inherits from the object class. The UML shows where we want to place this in the class hierarchy, where it inherits from (objects) and what behaviors it holds.

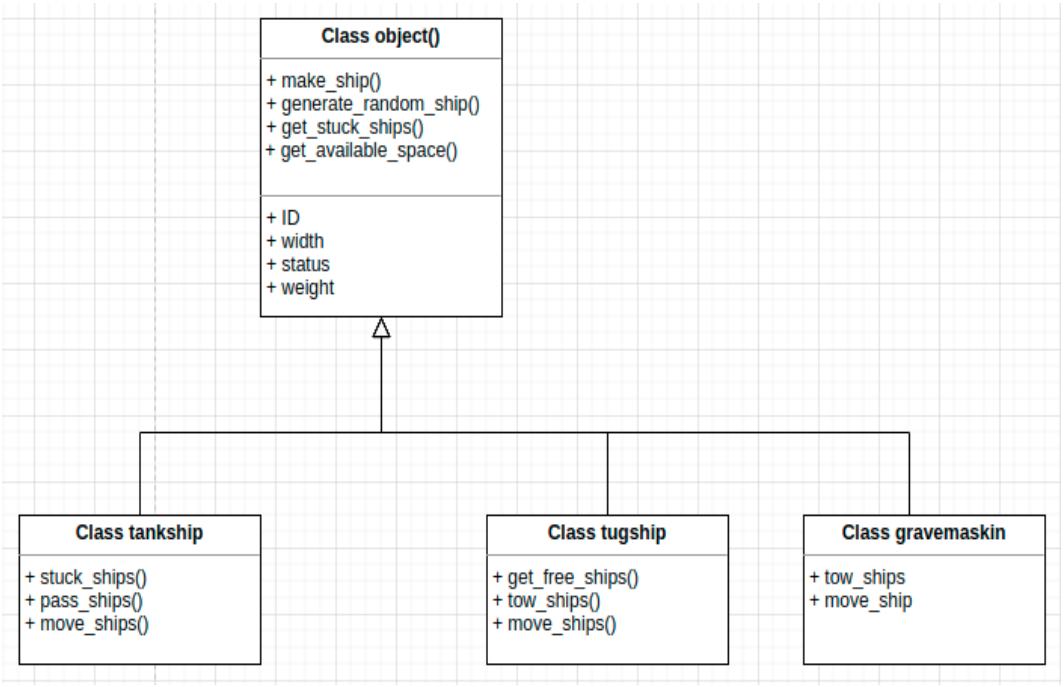

Figure 1: Illustration of an UML task 4.

2)

The "gravemaskin" class inherits the methods and attributes illustrated above from the parent class. In addition it hold two behaviors: "tow ship" and "move ship". Since the class does not depend on the queue and cant get stuck, it does not need to have the behaviors: "get free ships", "stuck ships" and "pass ships" compared to the boat classes that must have these features between them. We have put the width for the excavator to zero. In this way, the excavator does not interfere with the "get available space" function that wants to find out if there is enough space in the channel for ships.

We have set the weight factor to be equal to the number of tugboats. So an excavator and a tugboat for every third tankhip. We have also adjusted the towing capacity and ID for excavators. The excavators will constantly get on no matter how many boats are stuck. To solve this problem we have added movement to the "move ships" function for excavators. Since the job for the excavators is to free the ships, the move function will directly "tow ships" from the list of stuck ships and then update the list. In this way, the excavators don't move freely, but actively engage towing if there is a suck ship in ront of it.

3)

The code is in the attachment: opggave4.py