BDSA - Assignment04

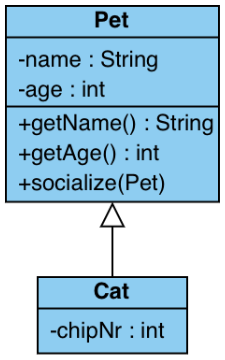
ehel, olfw, lawu, jakst

September 2022

Link to GitHub: https://github.com/Gufhans/assignment-04

Exercise 1

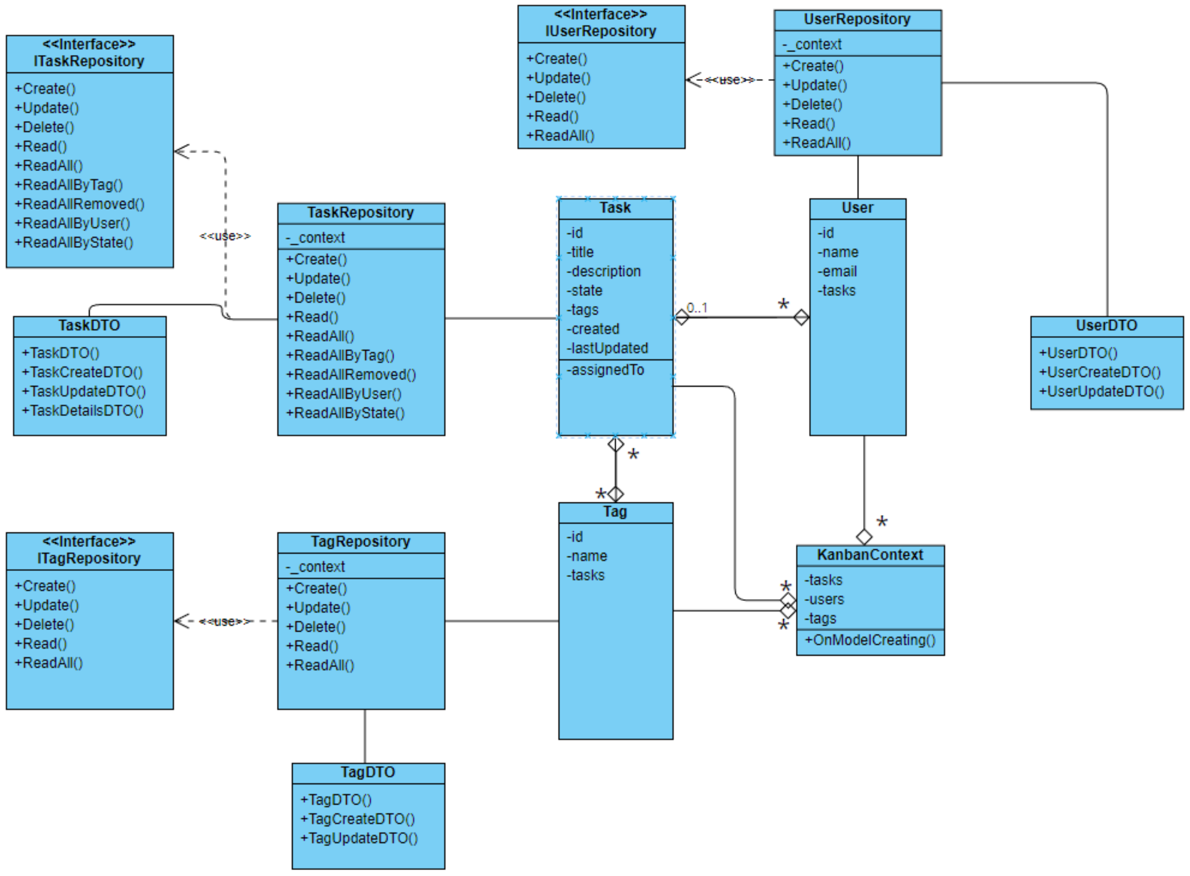
Encapsulation: Bundling data along with methods into a single unit and being able to restrict the direct access to some components of that unit.
Inheritance: A class can inherit fields and methods from a superclass. This helps with avoiding code duplication and allows for polymorphism.
Polymorphism: A subclass can act as its superclass or a class that inherits an interface can act as an instance of that interface.



The diagrams show encapsulation in form of the fields and operations of Pet. Its fields are private and can only be accessed via its getter methods. As for inheritance, Cat inherits all the fields and methods from Pet in addition to its own field chipNr. Since Cat is a subclass of Pet, it can be used anywhere where a Pet is used and therefore a Cat object can be used as a parameter when calling the socialize-method.

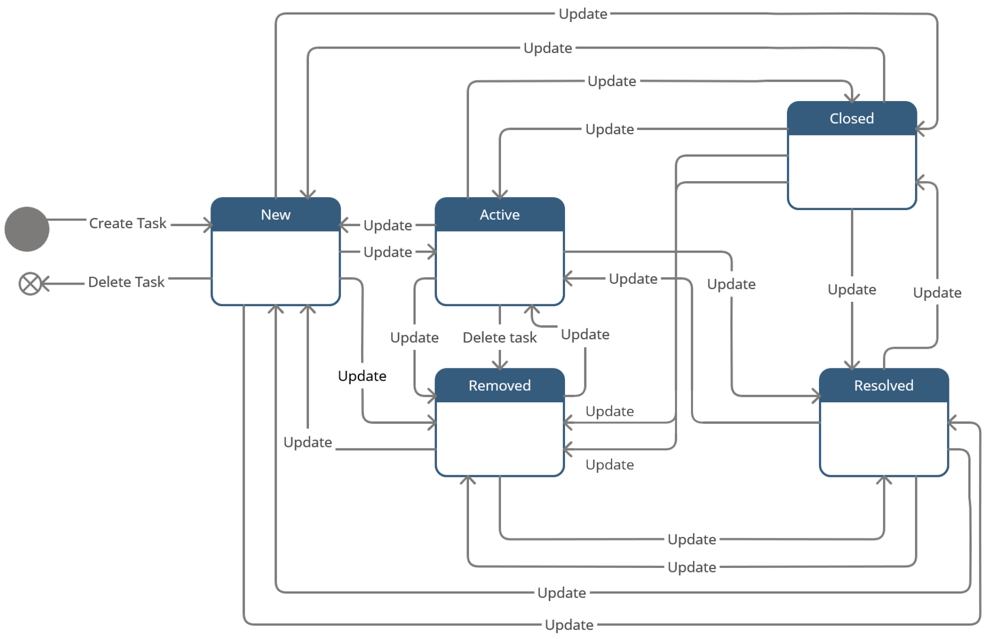
Exercise 2

Draw a UML class diagram that illustrates your implementation of the entities of last week's C assignment



Exercise 3

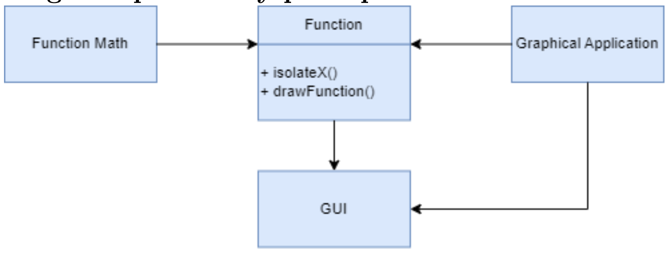
State diagram of the implementation of the WorkItem entity



WorkItem will always start with the state *New*, since that is the default state when created. Only from *New* can it be completely deleted again and thereby exit the state diagram. If attempting to delete while the WorkItem is in the *Active* state, the state will change to *Removed*. Attempting to delete while in any of the other states, only a response will be returned, and therefore it is not relevant in this diagram.
The only other way to change state in our implementation is the *Update* function, which can be called from any one state and change it to any other state.

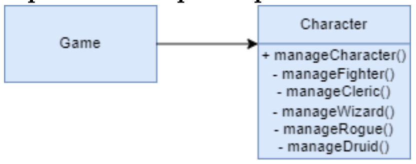
Exercise 4

Single responsibility principle



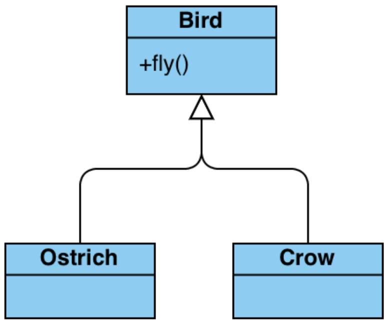
The Function class is responsible for both the underlying math regarding functions, and representing a function as a graph through a GUI.

Open-closed principle



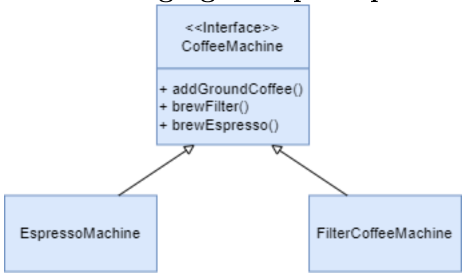
Character has a method for each type of character and therefore the class has to be modified each time a new character is added.

Liskov substitution principle



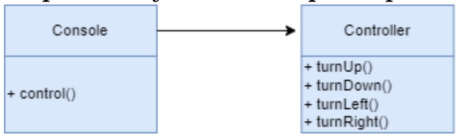
Ostrich is a subclass of Bird but does not have all the abilities of Bird (it cannot fly). Ostrich can therefore not be substituted for a Bird.

Interface segregation principle



Both coffee machines have a method from the coffeeMachine interface that it does not use/implement.

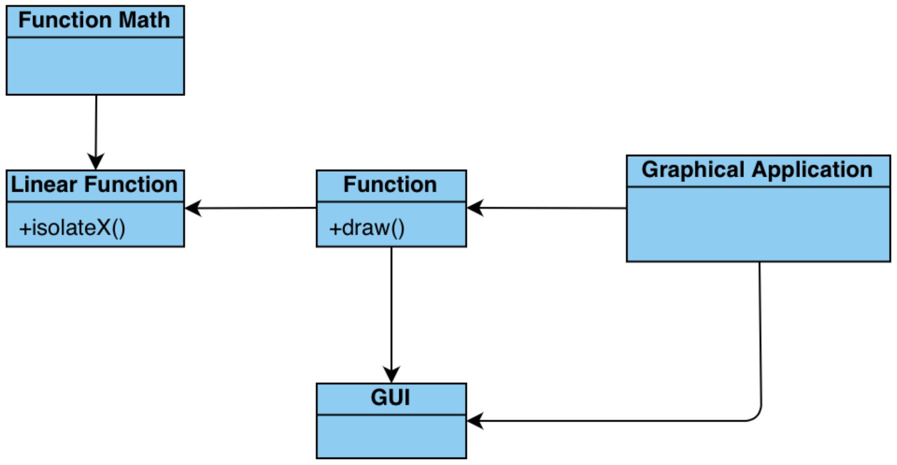
Dependency inversion principle



The Console class depends on that specific Controller class. It is unable to accept a different kind of controller other than what the Controller class defines.

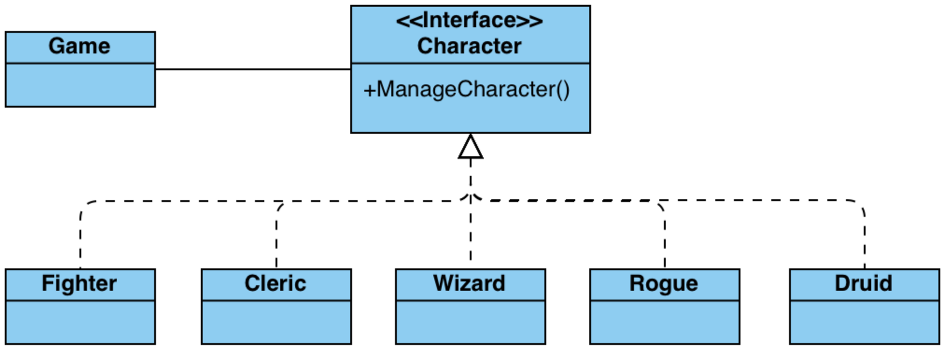
Exercise 5

Single responsibility principle



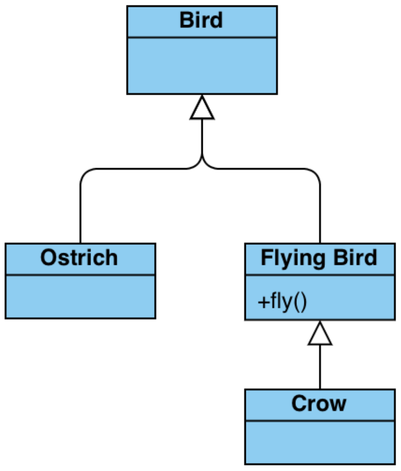
The linear function implementation is now independent from the Function class.

Open-closed principle



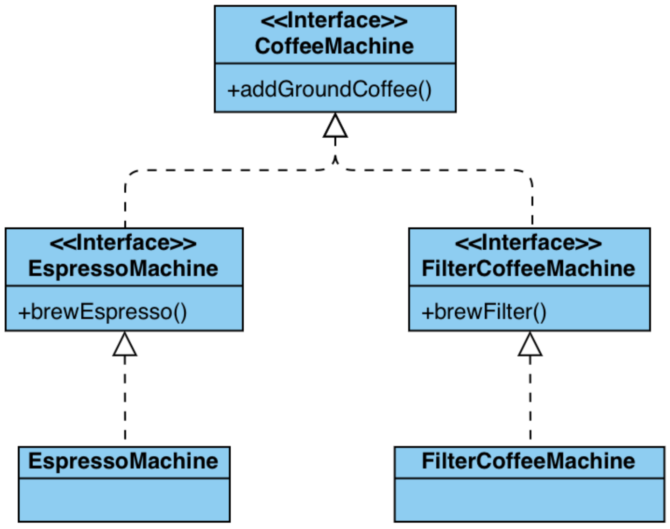
Character no longer needs to be modified when adding a new type of character.

Liskov substitution principle



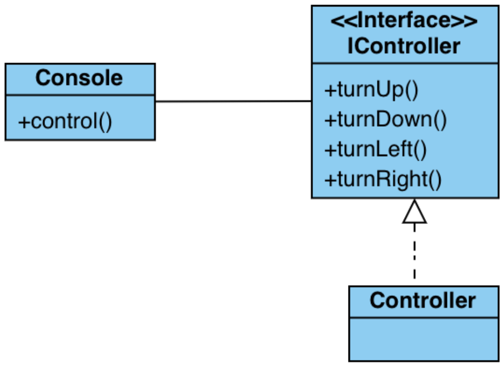
Ostrich can now stand-in for a Bird-object.

Interface segregation principle



The two coffee machines no longer have to implement methods that it cannot use.

Dependency inversion principle



Different kinds of controllers can now be defined and used by Console as long as it implements the IController interface.