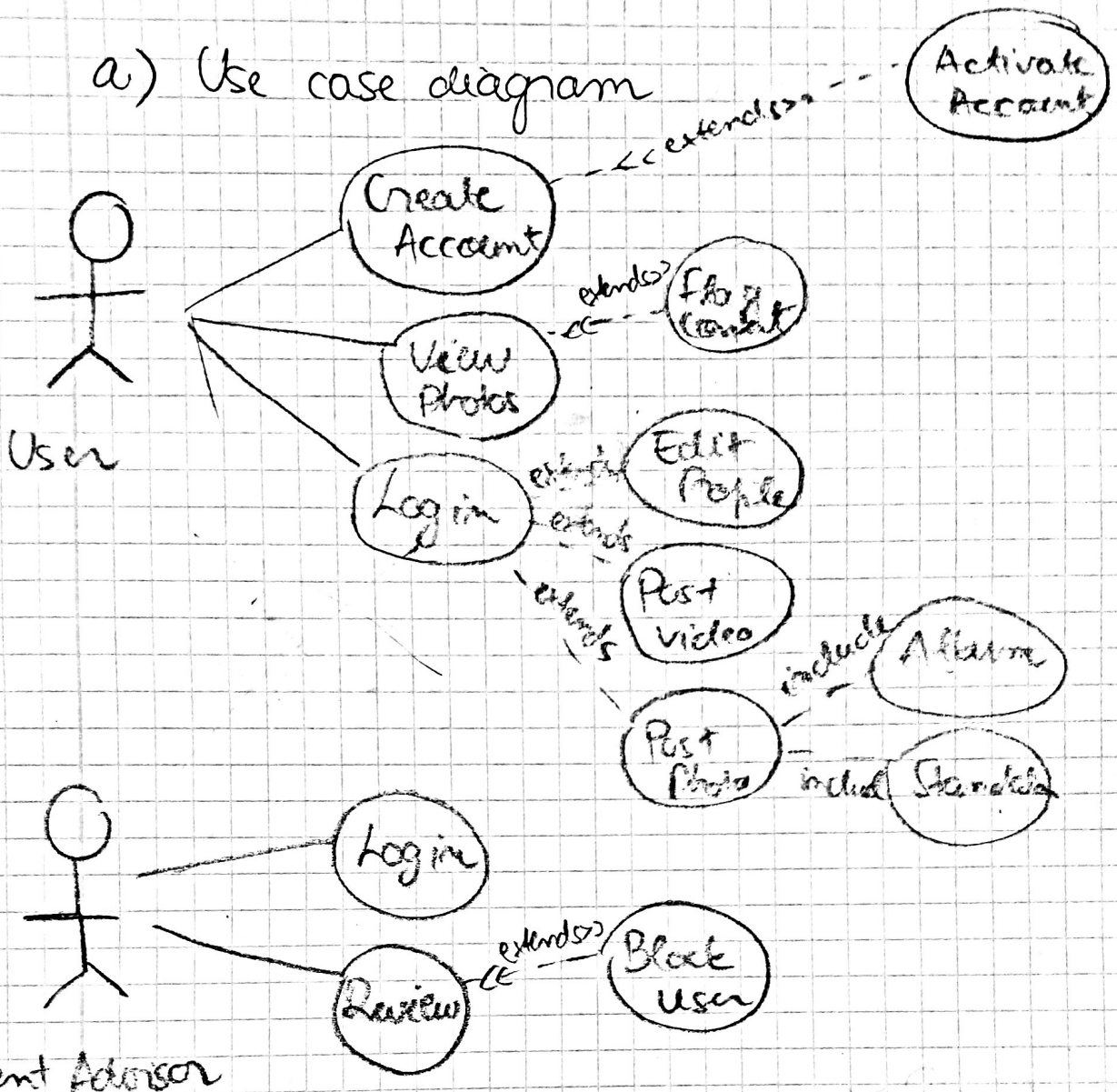
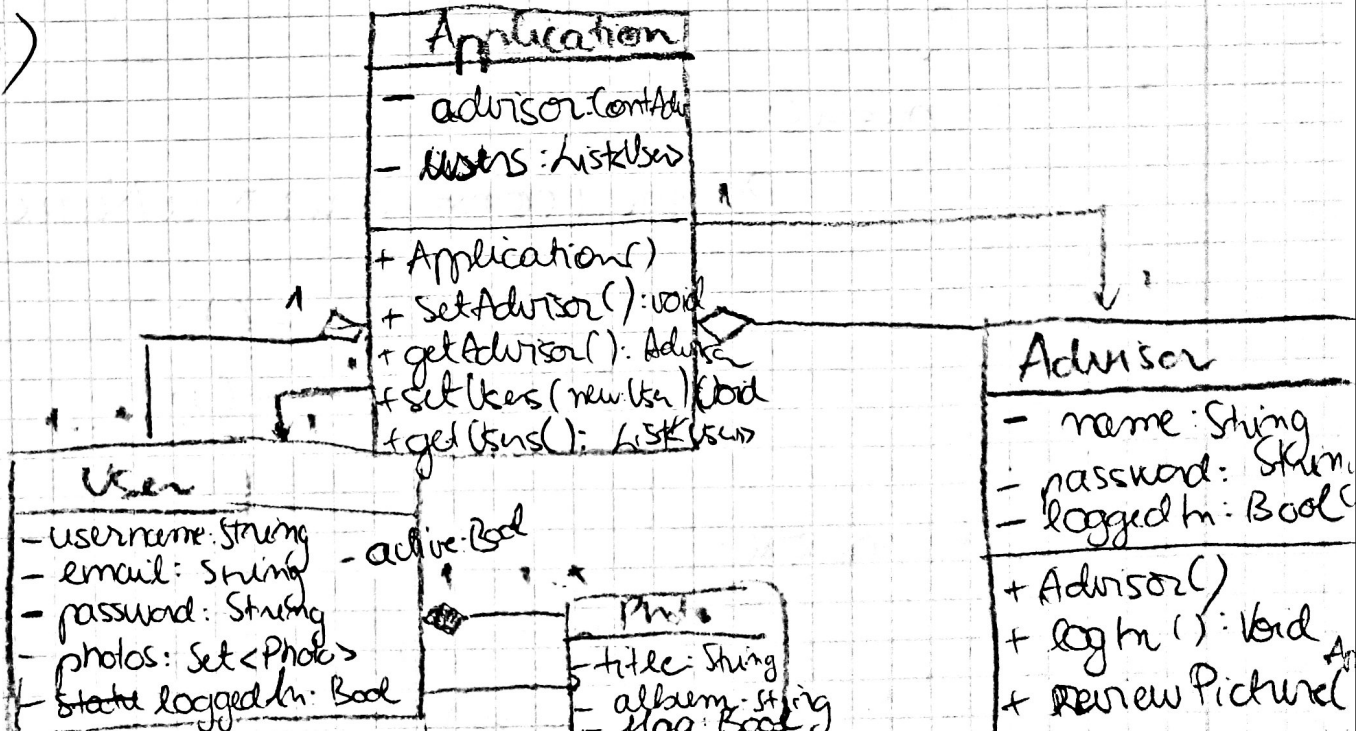
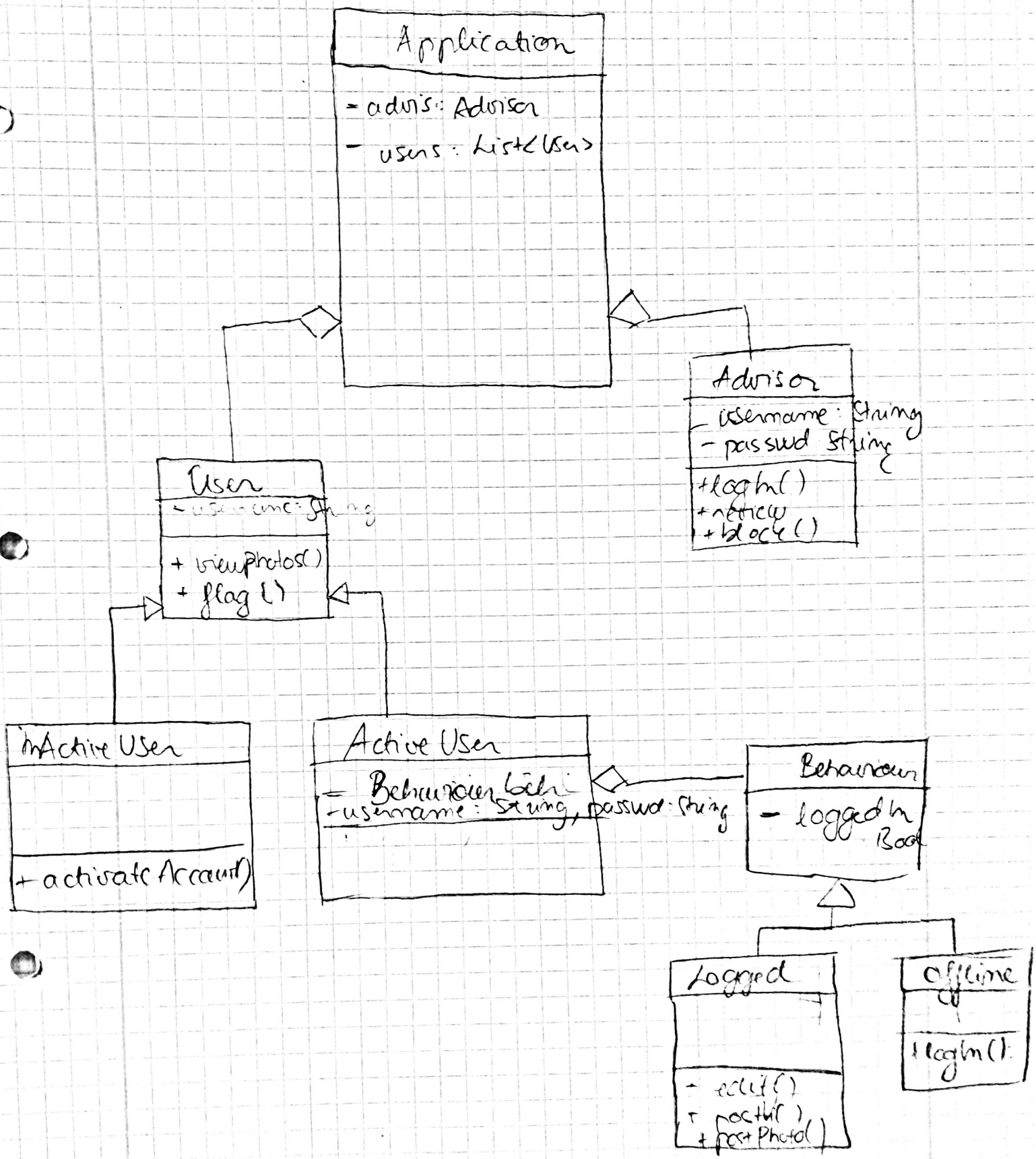
a) Use case diagram

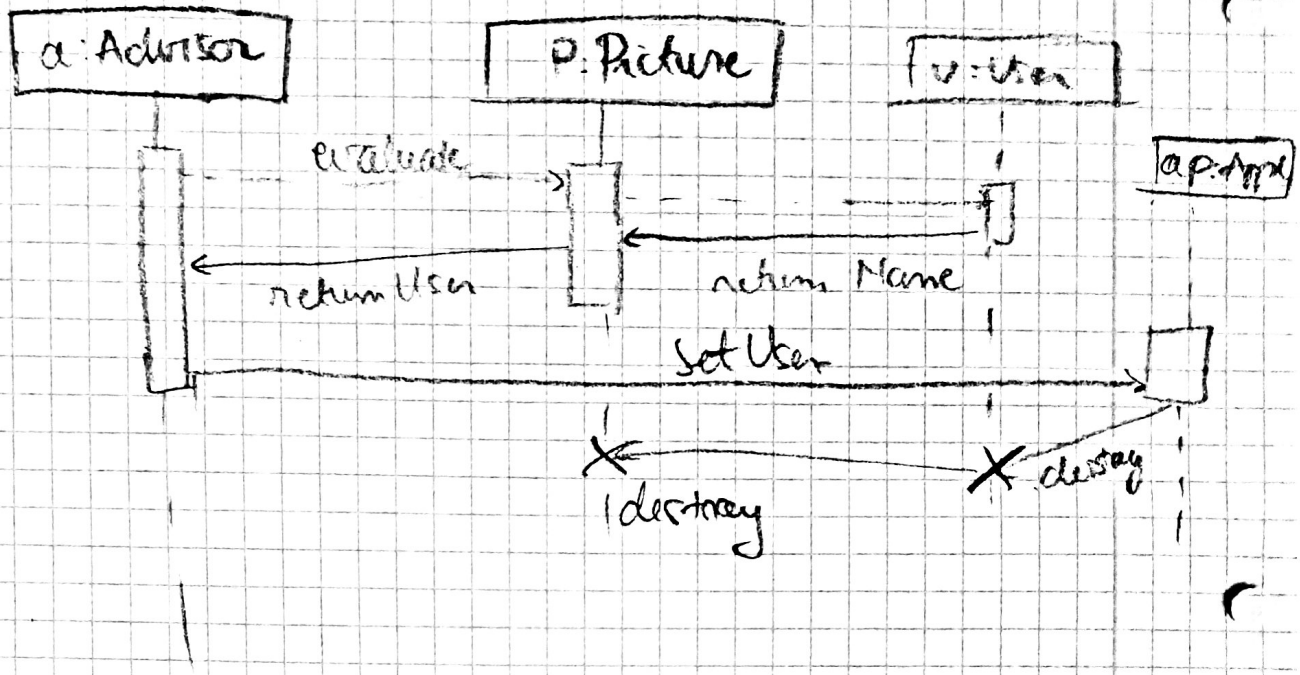


b)




Application
- advis: Advisor
- users: List<User>
Advisor
- username: String
- passwd: String
+ loginA()
+ retrieve()
+ block()
User
- username: String
+ viewPhotos()
+ flag()
InActive User
+ activateAccount()
Active User
- BehaviourLogIn
- username: String, passwd: String
Behaviour
- loggedIn
Logged
+ edit()
+ postPhoto()
Offline
+ loginA()

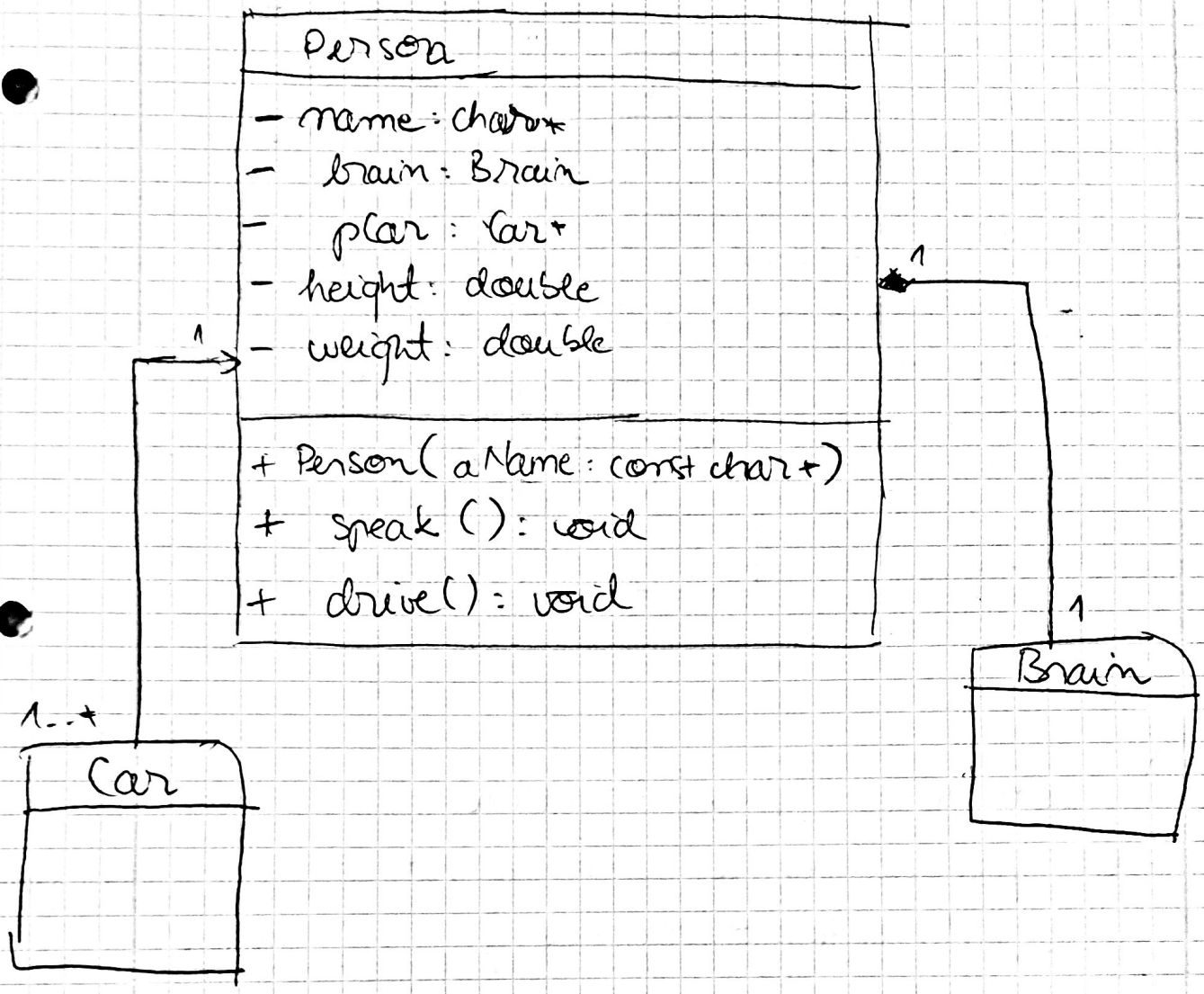
c.) Sequence diagram



P6.

```
class Person
{
public:
        Person (const char * aName);
        <-- void speak();
        <-- void drive();

private:
        char* name;
        Brain brain;
        Car * pCar;
        double height;
        double weight;
};
```

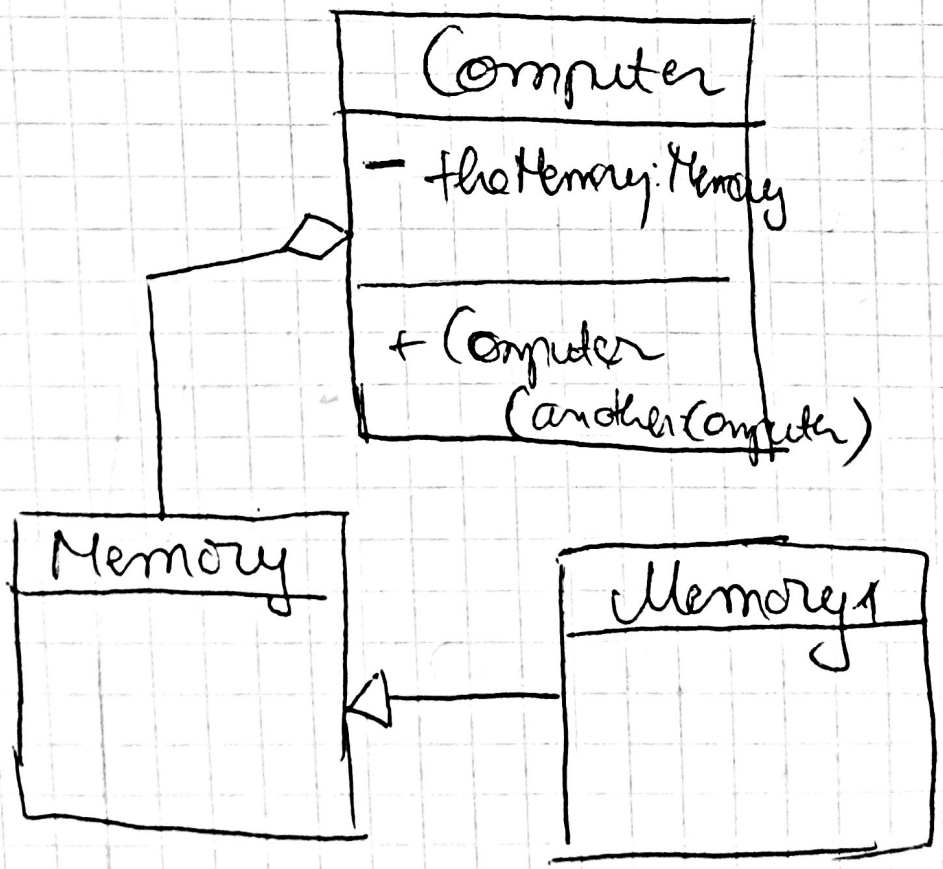


(P7.)

```
class Memory {...}

class Memory1 extends Memory {...}

class Computer
{
    private Memory theMemory;
    public Computer (Computer another)
        {  if (another.theMemory instanceof Memory1)
               theMemory = new Memory1 (another.Memory
           else
               the_Memory = new Memory ( another.Memo
}   }
```


Computer
- theMemory: Memory
+ Computer
(anotherComputer)
Memory
Memory1

(P8)

Factory Method Design Pattern

- we can create object without exposing the creation logic to the client ; refer to object using a common interface
- avoid duplication of code
- define a separate method fa creating the objects -> subclasses overwrite it - they decide which object to instantiate

Usage : Imagine a company that distributes many different products (Vehicle salon - cars, motorcycles, boats ; toy store ; supermarket; thrift shop etc.)

- if with the introduction of every new product we need to modify the class that instatiates the product classe, we break the open/close principle