Révisions

POLYTECH NICE SOPHIA | UNIVERSITÉ CÔTE D'AZUR

- U Les cas d'utilisation : guides et erreurs à ne pas commettre
- C Le diagramme de classes : pourquoi/comment faire des associations
- S Le diagramme de séquences : comment le rendre bien cohérent avec les 2 autres diagrammes
- G GRASP-SOLID : retour sur la distribution des responsabilités
- T User stories : qu'est-ce qu'une bonne US
- P Patrons de conception : comment décider d'en utiliser un ou pas
- Patrons de conception : retour sur un ou des patrons spécifiques (donnez leur nom en répondant à ce message)

U 22 C 34 S 30 G 24 P 46 T 8

Patrons de conception : comment décider d'en utiliser un ou pas

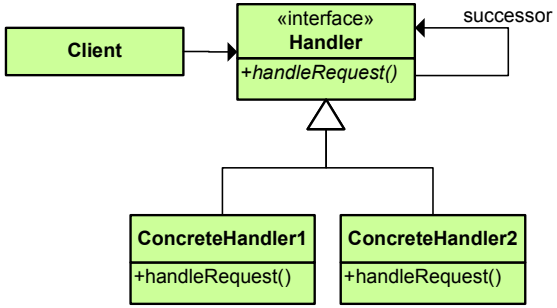
1. Ai-je bien compris le problème à résoudre ?

2. Les motivations du patrons correspondent-elles à mon problème ?

3. Les inconvénients ne sont-ils pas plus importantes que les avantages ?

1. Ai-je bien compris le problème à résoudre ?

- Utiliser le type des patrons
  - Création
  - Structure
  - Comportement
- Puis la motivation décrite dans le patron, donc le point 2…
- Puis la balance entre avantages et inconvénients (over-engineering ou pas)



Chain of Responsibility

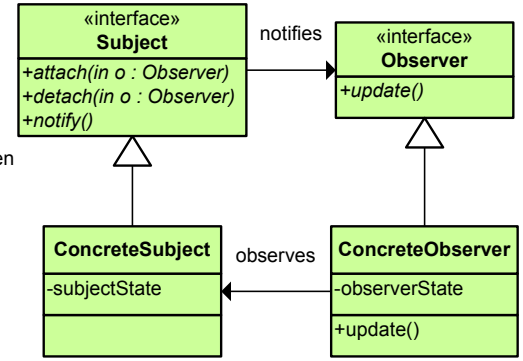
Type: Behavioral

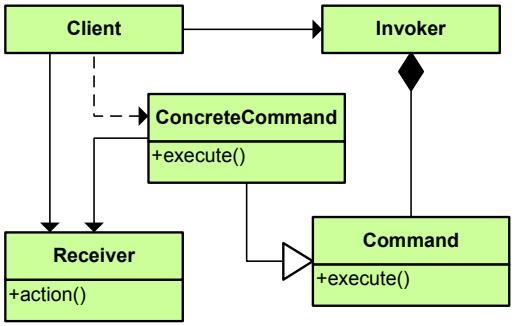
What it is:
Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

Observer

Type: Behavioral

What it is:
Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.





Command

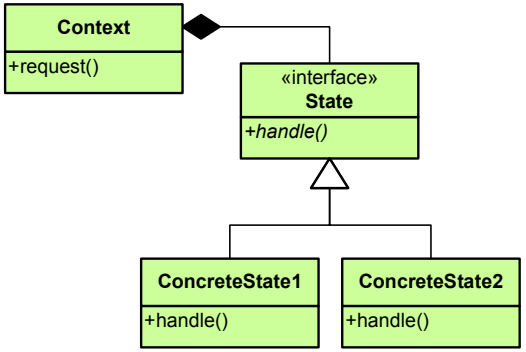
Type: Behavioral

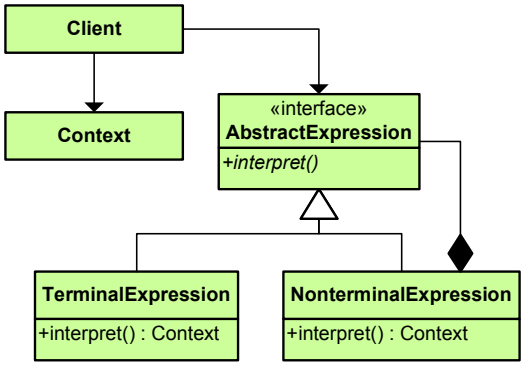
What it is:
Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.

State

Type: Behavioral

What it is:
Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.





Interpreter

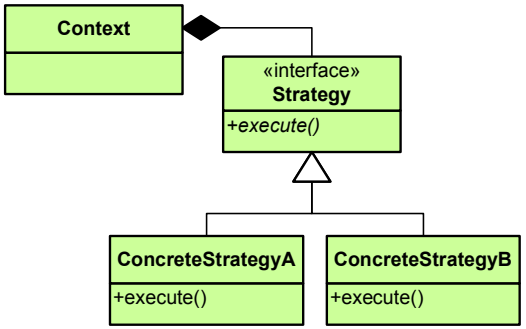
Type: Behavioral

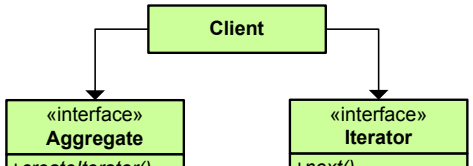
What it is:
Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

Strategy

Type: Behavioral

What it is:
Define a family of algorithms, encapsulate each one, and make them interchangeable. Lets the algorithm vary independently from clients that use it.





Iterator

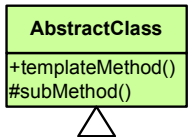
Type: Behavioral

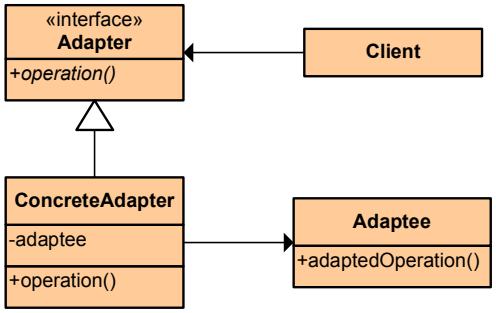
What it is:

Template Method

Type: Behavioral

What it is:





Adapter

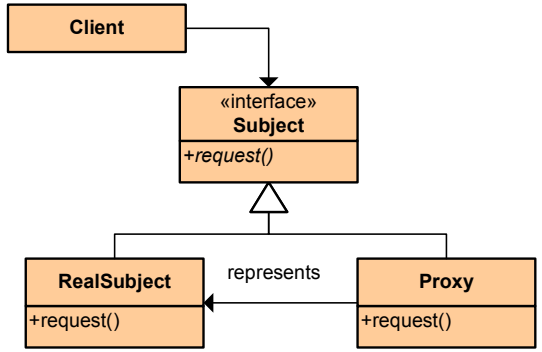
Type: Structural

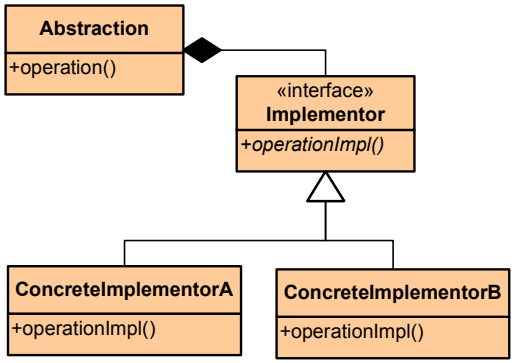
What it is:
Convert the interface of a class into another interface clients expect. Lets classes work together that couldn't otherwise because of incompatible interfaces.

Proxy

Type: Structural

What it is:
Provide a surrogate or placeholder for another object to control access to it.





Bridge

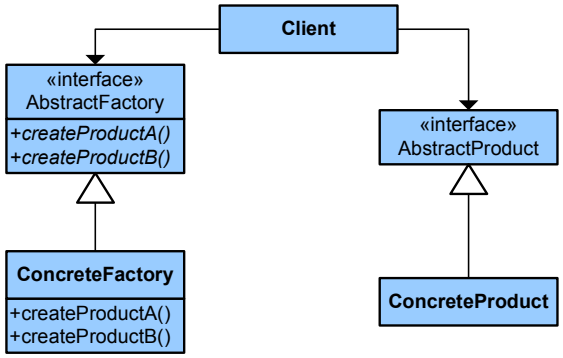
Type: Structural

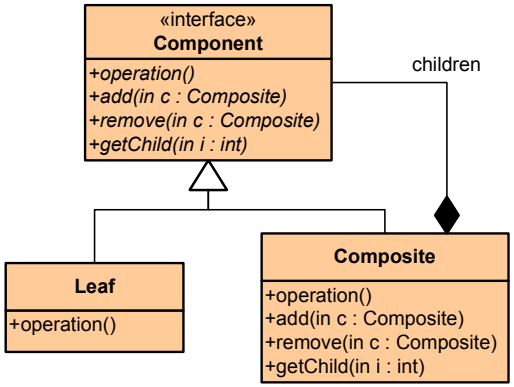
What it is:
Decouple an abstraction from its implementation so that the two can vary independently.

Abstract Factory

Type: Creational

What it is:
Provides an interface for creating families of related or dependent objects without specifying their concrete class.





Composite

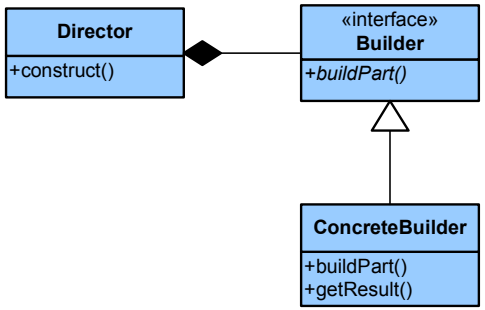
Type: Structural

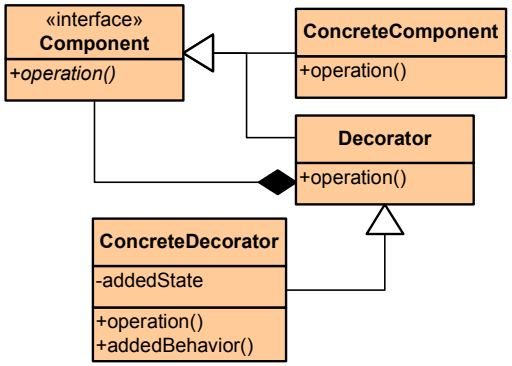
What it is:
Compose objects into tree structures to represent part-whole hierarchies. Lets clients treat individual objects and compositions of objects uniformly.

Builder

Type: Creational

What it is:
Separate the construction of a complex object from its representing so that the same construction process can create different representations.





Decorator

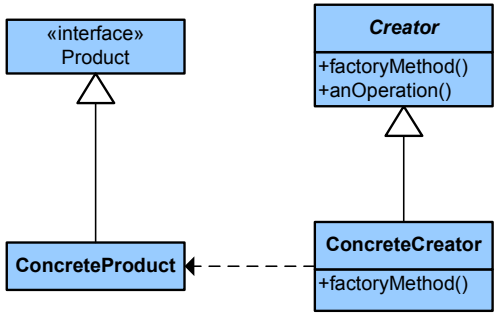
Type: Structural

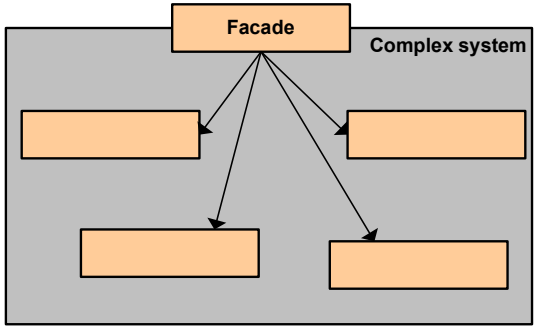
What it is:
Attach additional responsibilities to an object dynamically. Provide a flexible alternative to sub-classing for extending functionality.

Factory Method

Type: Creational

What it is:
Define an interface for creating an object, but let subclasses decide which class to instantiate. Lets a class defer instantiation to subclasses.





Facade

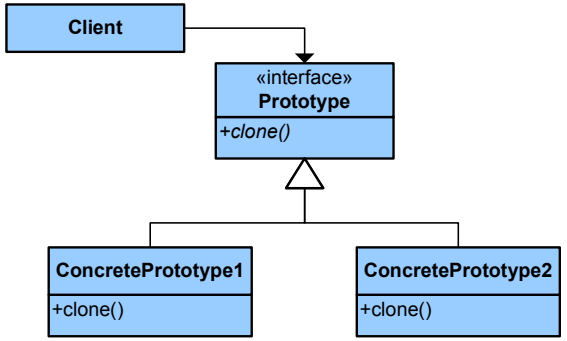
Type: Structural

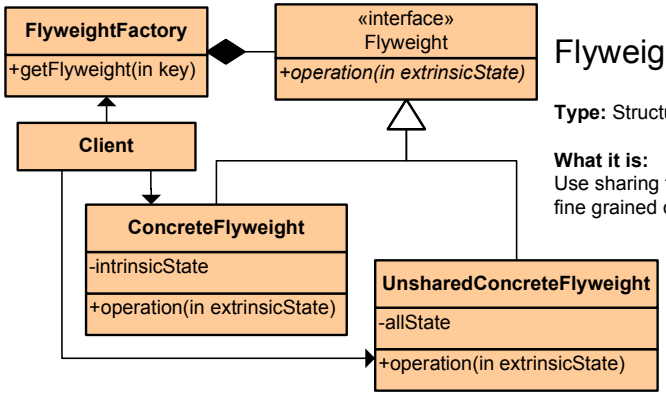
What it is:
Provide a unified interface to a set of interfaces in a subsystem. Defines a high-level interface that makes the subsystem easier to use.

Prototype

Type: Creational

What it is:
Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.





Flyweight

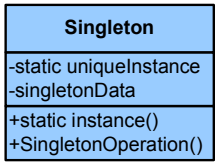
Type: Structural

What it is:
Use sharing to support large numbers of fine grained objects efficiently.

Singleton

Type: Creational

What it is:
Ensure a class only has one instance and provide a global point of access to it.

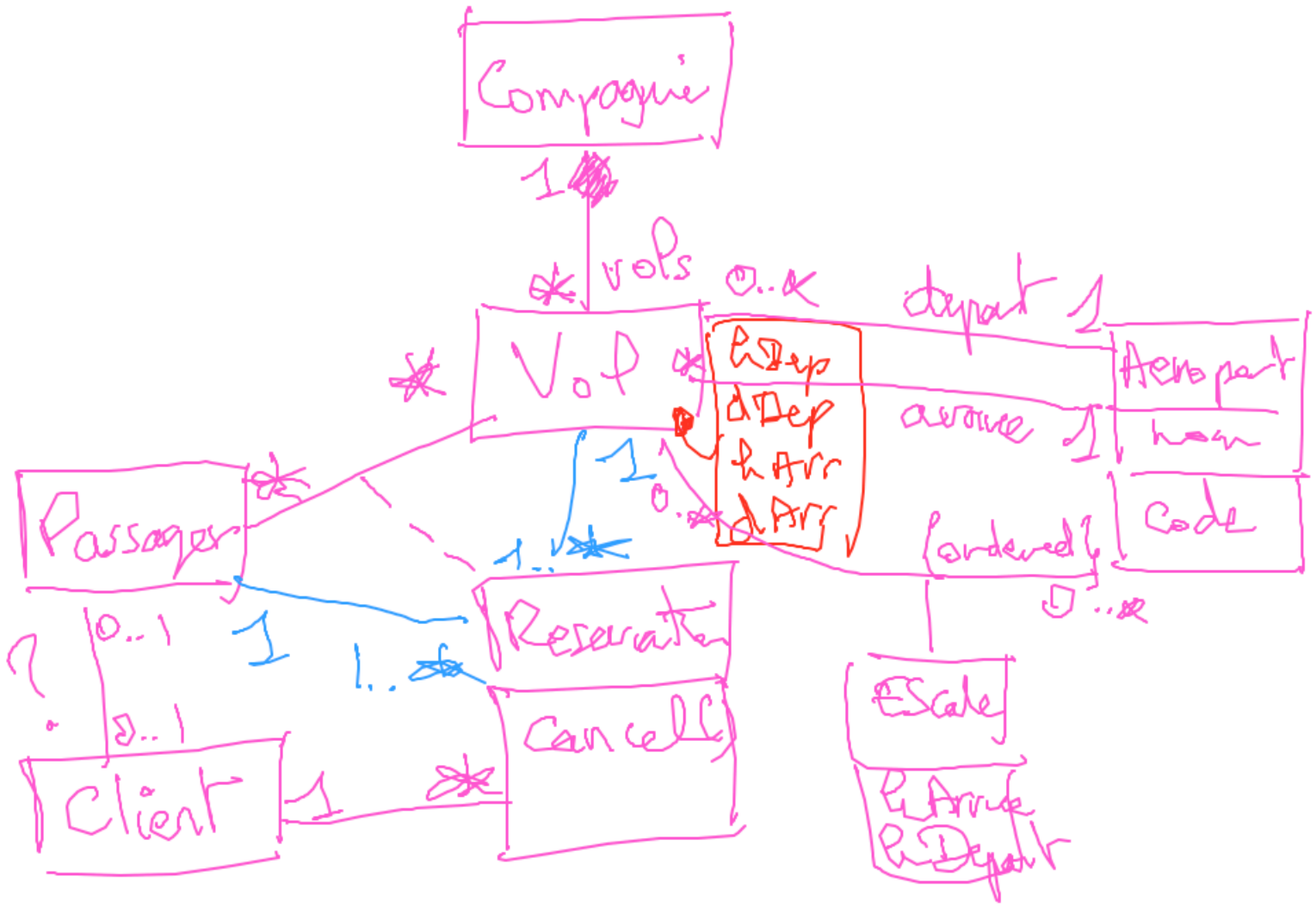


Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Massachusetts: Addison Wesley Longman, Inc..

Classes et associations - examen de l'an dernier

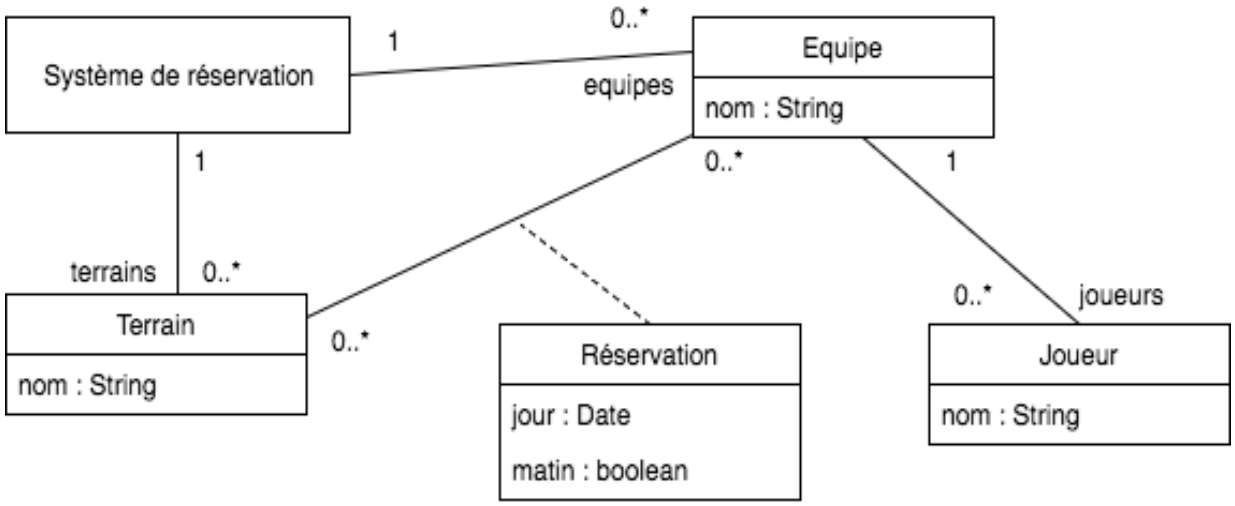
Cette étude de cas concerne un système simplifié de réservation de vols pour une agence de voyages. Des interviews ont permis de mettre en évidence les éléments suivants :

- Des compagnies aériennes proposent différents vols.
- Un vol est ouvert à la réservation et refermé sur ordre de la compagnie.
- Un client peut réserver un ou plusieurs vols, pour des passagers différents.
- Une réservation concerne un seul vol et un seul passager.
- Une réservation peut être annulée ou confirmée.
- Un vol a un aéroport de départ et un aéroport d'arrivée.
- Un vol a une heure de départ et une heure d'arrivée, ainsi qu'une date de départ et une date d'arrivée.
- Un vol peut comporter des escales dans des aéroports.
- Les escales interviennent dans un ordre déterminé.
- Une escale possède jour et heure d'arrivée ainsi que jour et heure de départ.


Compagnie
1
vols
0..*
V.o.f
dDep
dArr
départ
arrivée
Aéroport
nom
Code
ordered
Passagers
Réservation
Cancel
Client
Escale
0..1
1
1..*

Séquences - examen de l'an dernier

On considère le diagramme de classes suivant, version très simplifiée d'un modèle qui prend en compte la gestion des entraînements d'équipes sportives sur des terrains :



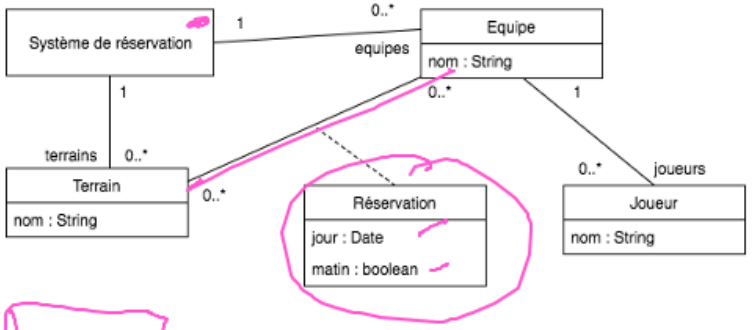
Des équipes, composées de joueurs, peuvent réserver des terrains. La réservation du terrain pour un entraînement se fait pour un jour donné, le matin (booléen *matin* à vrai) ou l'après-midi (booléen à faux).

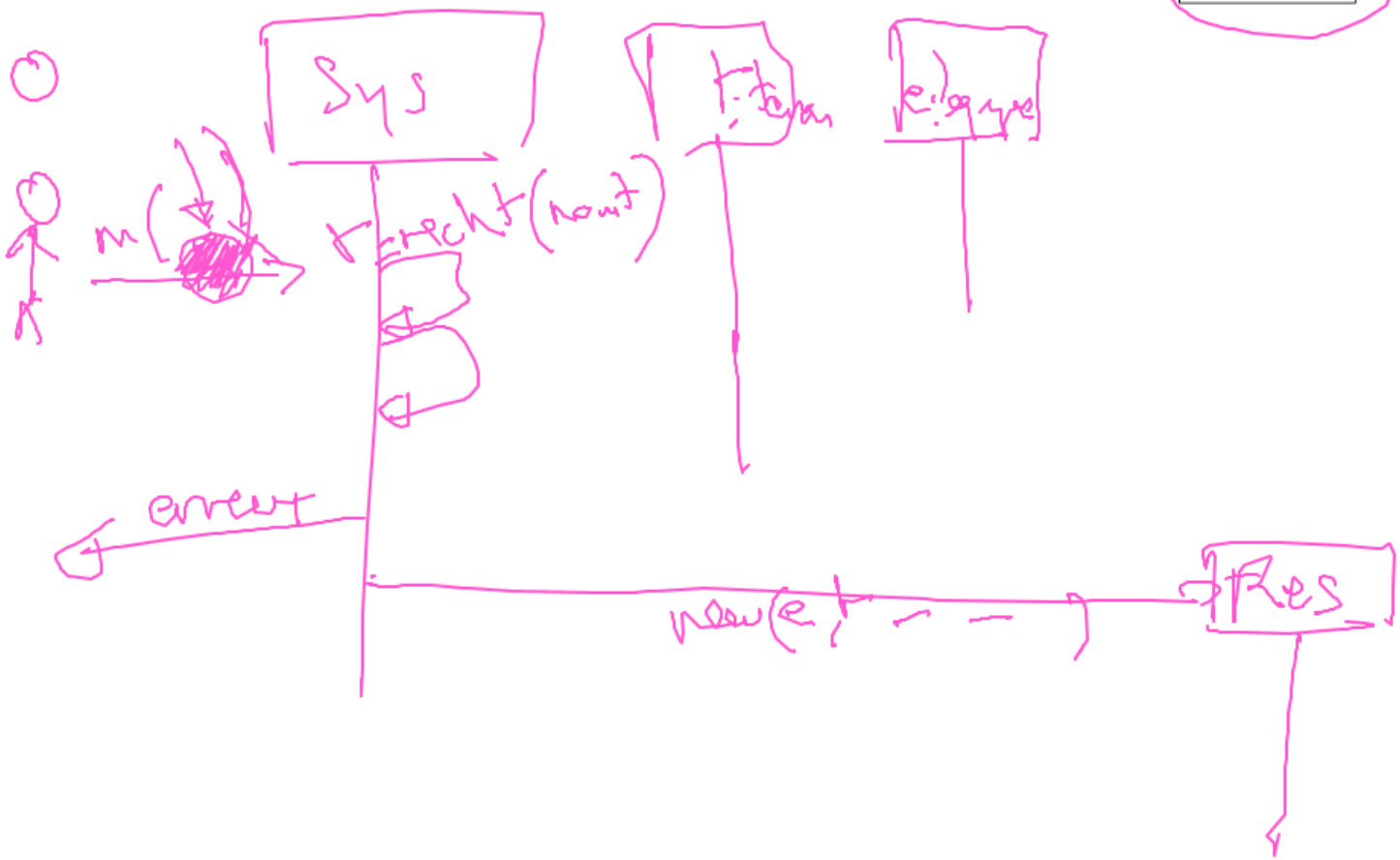
La classe « Système de réservation » est la classe principale, point d'accès des utilisations.

Séquences - examen de l'an dernier

Proposez un diagramme de séquences correspondant au cas d'utilisation de réservation pour un nouvel entraînement : l'acteur qui enclenche ce cas est le responsable des terrains ;

- il saisit pour cela le nom de l'équipe, le nom du terrain, la date souhaitée, et un booléen indiquant si la réservation est prévue le matin (vrai) ou l'après-midi (faux).
- Dans cette séquence, le système vérifie alors si le terrain est disponible et crée un objet réservation correspondant. S'il n'est pas disponible, il remonte une information de refus à l'acteur.


Système de réservation
1
0..*
Equipe
equipes
nom : String
1
0..*
1
terrains
0..*
0..*
0..*
joueurs
Terrain
nom : String
Réservation
jour : Date
matin : boolean
Joueur
nom : String


Sys
erreur