

# Advanced C++ programming

- ◆ **Software Engineering: why ?**
- ◆ **Using UML to model a solution**
  - Design guidelines
  - Why UML?
  - Modeling...
- ◆ **Some UML diagrams**
  - Use case diagram
  - Class diagram
  - Sequence diagram
  - Practice: from text to UML
- ◆ **Design Patterns**
  - Goals
  - Examples

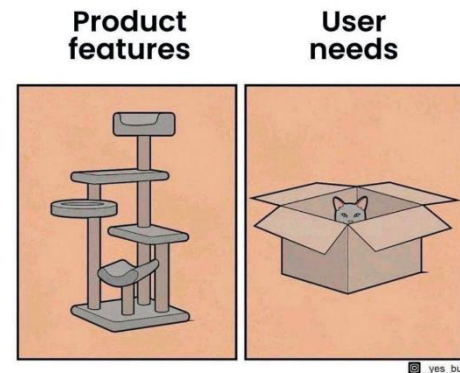
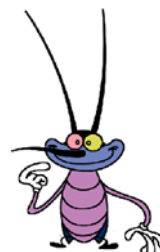
# Some facts...

- **Software life cycle (year 2000)**

- 70 % of the costs ► maintenance
- 80 % of developers ► maintenance

- **IT project management ?**

- 25 % of software projects never ended
- 1 % of the “big” software projects ended on time
- The others were 1 year late on average
- 2% of software really matched actual user needs ...



**How to produce a “working” good quality software?**



# Why is it so hard?

- **Software context**

- Wide coverage ► specific application/business/scientific domain **and** computer science
- Client-dependent constraints on the final software

- **Software system**

- Interconnected subsystem hierarchy
- Arbitrary definition of what is a software component

- **Development process**

- Team development issues (organization, human factor)
- Splitting into independent tasks?



# Software engineering... why?

## • Produce a good quality software ... according to which point of view ?

- **External**: match the requirements, resources fair use, target versability  
▶ efficiency, portability, reliability, robustness
- **Internal**: easy to read, “obvious” design, well-documented, data / processing separation  
▶ intelligibility, reusability, compatibility, verifiability
- **Maintenance**: bug correction, evolution (new functions, hardware, environment)

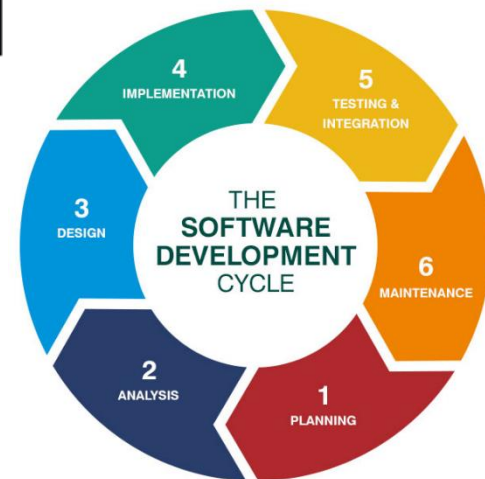
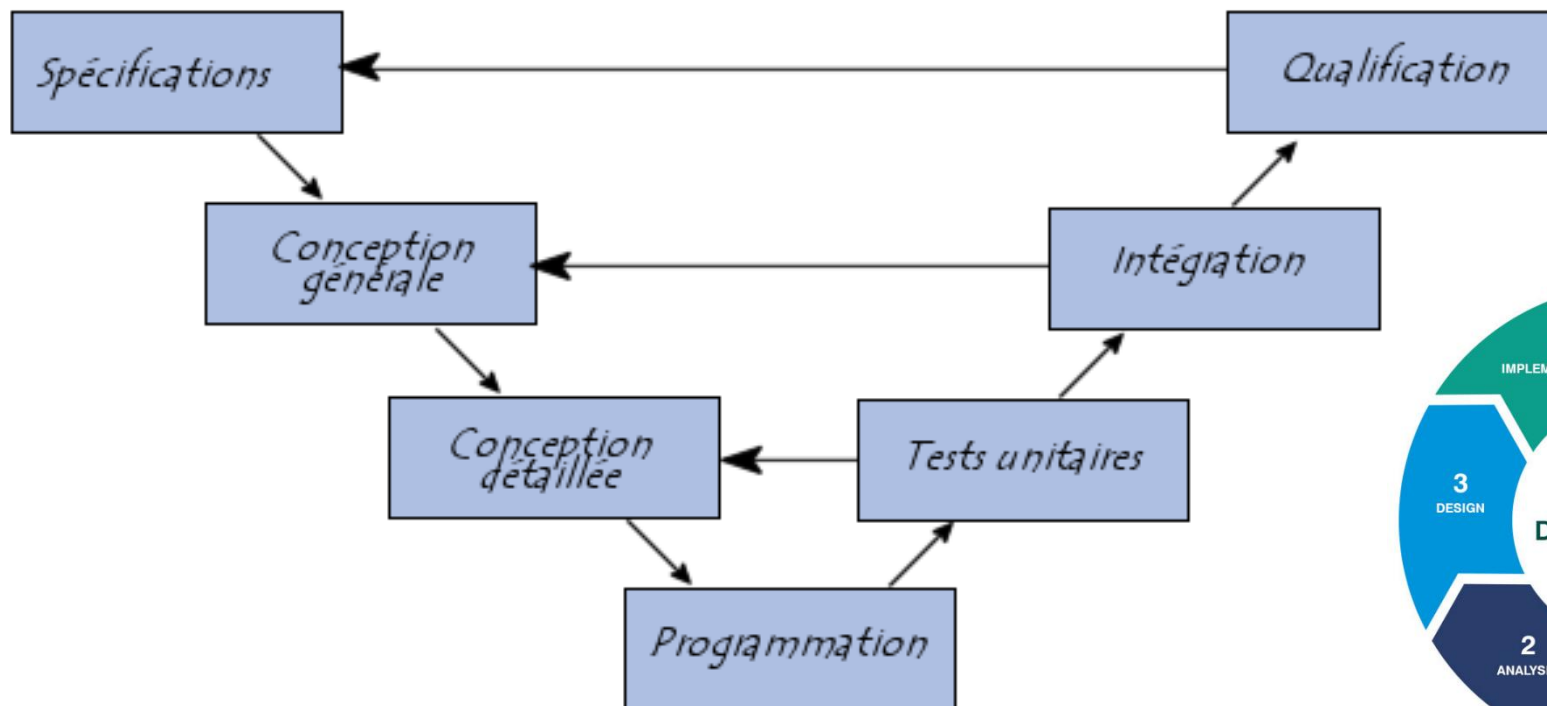


**Focus: Object Oriented Design aiming at a C++ implementation**



# Software life cycle

- V-cycle software development
  - Old fashioned BUT highlights sanctions in case of inadequate analysis



# Some design guidelines (1/2)

- **SOLID** ▶ from Martin, Robert C. (2000). "Design Principles and Design Patterns"
  - Single responsibility principle ▶ one class, one responsibility (only one reason to change)
  - Open-closed principle ▶ open for extension, closed for modification
  - Liskov substitution principle ▶ an object (from a class) and a sub-object (from a class that extends the first class) must be interchangeable without breaking the program
  - Interface segregation principle ▶ no code should be forced to depend on methods it does not use (interface splitting to prevent coupling dependency)
  - Dependency inversion principle ▶ loosely coupling through interface (high-level classes access to their low-level classes through interfaces reducing coupling)



# Some design guidelines (2/2)

## • Other references

- Separation of concerns ► one part, one concern
- Reuse (“Don’t reinvent the wheel”)
- Maximum encapsulation
- Weak coupling between parts / modules
- Strong cohesion for parts / modules ► degree to which the elements inside a module belong together
- DRY (“Don’t Repeat Yourself”)
- KISS (“Keep It Simple, Stupid”)
- YAGNI (“You Aren’t Gonna Need It”)





# Unified Modeling Language

## UML

- **What is UML?**

- Industry standard to visualize the design of a system
- General purpose, development language agnostic
- Fit for OOM (Object Oriented Modeling)

- **Language purpose**

- Be able to **represent** full systems (beyond software only) through object paradigms
- Modeling language for both humans and machines
- From concepts to final executable products





# UML strength

- **UML is a formal language with specifications**
  - Accurate through a rich **graphical** representation
  - Help formalizing analysis
  - Allow the use of software tools (consistency check, ...)
- **UML is also a means of communication**
  - Between client / design team / software team
  - Help to grasp complex abstract representations
  - System description goals... not only for software development



# A model ?

- **Reality abstraction**

- Make a full system easier to understand by focusing on its relevant behavior for the problem ► reduce real world unnecessary complexity
- A model represent the studied system and reproduces its behaviors ► allow simulation and widen the coverage analysis of the studied system through mathematical or computer means



- **UML allows model representation BUT does not define how to implement them**



# UML diagrams

- **System dynamic views: behavioral diagrams**
  - **Use case** diagram ► possible user interaction with the system (functional point of view)
  - **Sequence** diagram ► process interactions in a time sequence to solve a function (use case realizations time)
  - Activity, Communication, State, Timing, ... diagrams
- **System static views: structural diagrams**
  - **Class** diagram ► modeled system structure through class (data and operations) and relationships among objects
  - Component, Deployment, Object, Package, ... diagrams



# Use case diagrams

- Objectives

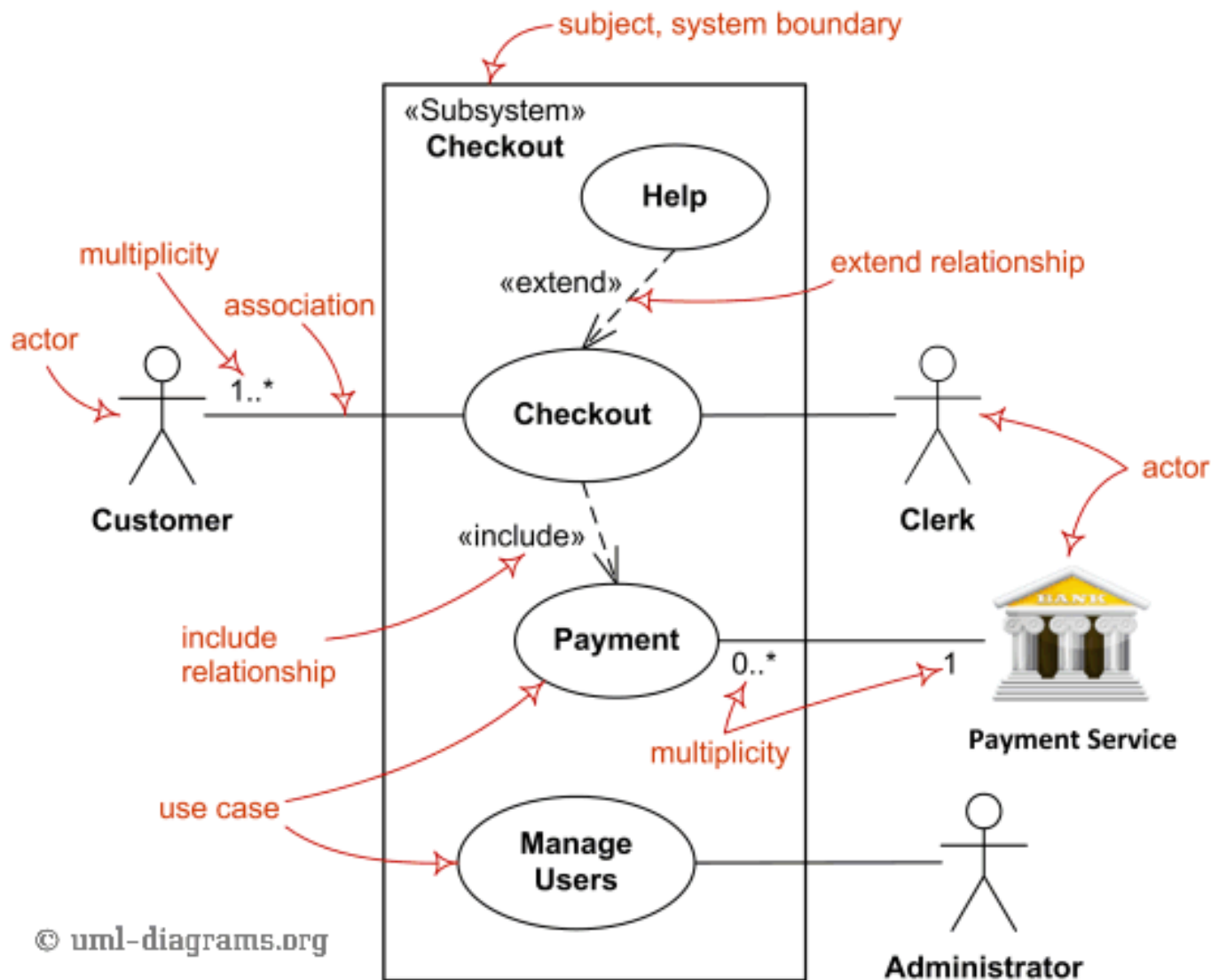
- “Who” / “what” is interacting with the system?
  - ▶ **system boundaries**
- To do what? Expectations from a user point of view? ▶ **use cases**

- Interesting features

- How to grasp the system environment? What is expected from the system? ▶ User can be part of the process, refining the description
- The use cases are the base for functional tests




# Use case diagram example



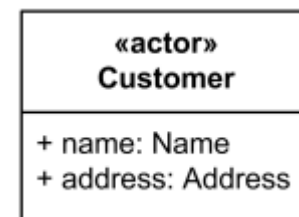
# Use case diagram entities (1/2)

- **Actors: external entity interacting with the system**

- 
- Exchanging data
  - Accessing or modifying the system state
  - Asking for a service solving its need
- Student



Bank



Web Client

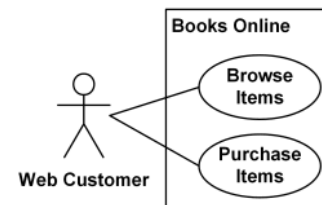
- **Use cases: actions to be done by the system in response to a specific interaction from one actor**
  - A system objective is described by a set of use cases



# Use case diagram entities (2/2)

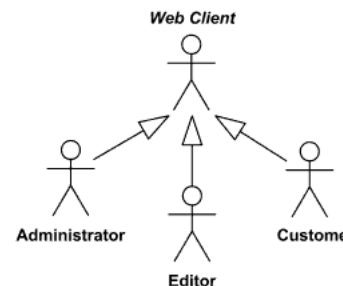
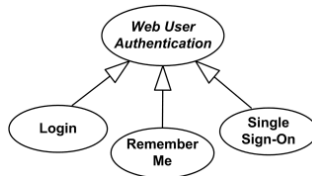
## • Association between actor and use case

- Communication between an actor and a use case



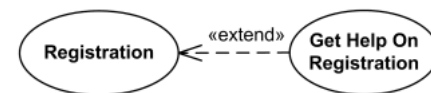
## • Generalization of an actor / of a use case

- « same » meaning as generalization between classes



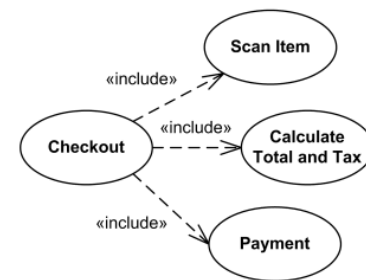
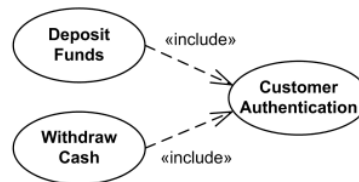
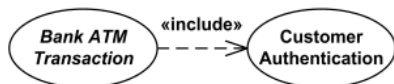
## • Extend relationship between use cases

- Extra optional behavior ("Registration" is complete on its own)



## • Include relationship between use cases

- Mandatory behavior (« Checkout » always needs « ScanItem »)





# Application (1 / 6)

- Use case from client requirements

## Achat sur un site Web marchand (par Internet) :

Tout visiteur (même anonyme) peut parcourir le site et trouver des articles à acheter.

Pour effectuer un achat, le visiteur doit avoir un compte client et doit donc auparavant s'enregistrer auprès du site.

L'achat d'un ou plusieurs articles par un client consiste à trouver le ou les articles souhaités, les placer dans un panier et à effectuer le règlement qui correspond au prix des articles du panier.

Les articles montrés sont obligatoirement disponibles.

### Question :

Dessiner un diagramme de cas d'utilisation correspondant à la description donnée.



# Application (2/6)

## • Use case from client requirements

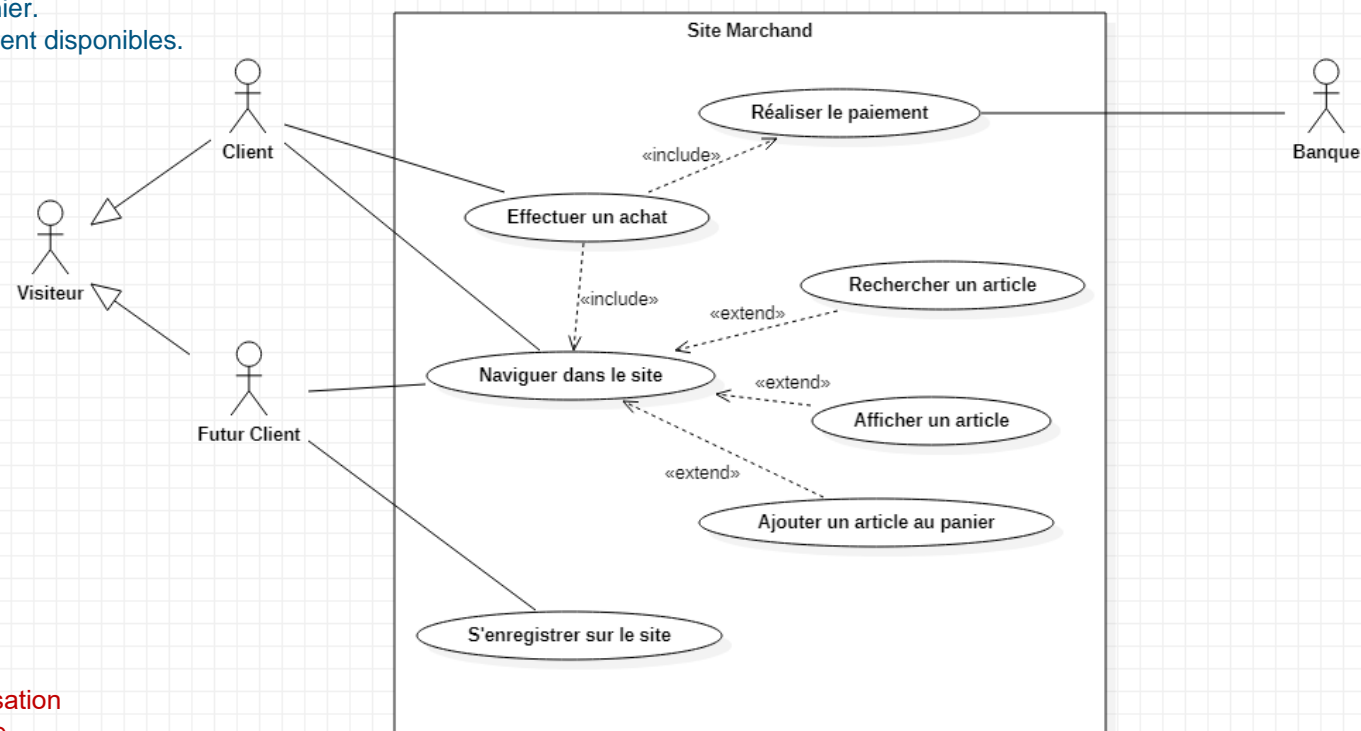
### Achat sur un site Web marchand (par Internet) :

Tout visiteur (même anonyme) peut parcourir le site et trouver des articles à acheter.

Pour effectuer un achat, le visiteur doit avoir un compte client et doit donc auparavant s'enregistrer auprès du site.

L'achat d'un ou plusieurs articles par un client consiste à trouver le ou les articles souhaités, les placer dans un panier et à effectuer le règlement qui correspond au prix des articles du panier.

Les articles montrés sont obligatoirement disponibles.



### Question :

Dessiner un diagramme de cas d'utilisation correspondant à la description donnée.



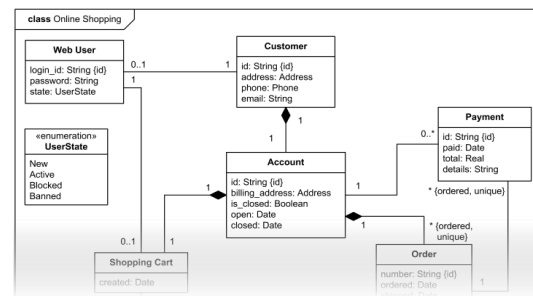
# Class diagrams (1/2)

- Part of the structure diagrams family

- Package diagram, component diagram, deployment diagram, ...

- Objectives

- Represent the structure of the system at the level of classes and interfaces, giving their features (attributes and operations), constraints and relationships
- May propose several partial views of the system
  - ▶ editorial choice for clarity (one system functionality)
    - only a subset of **relevant** classes, features, constraints, relationships
    - missing elements does not mean they don't exist in the full system

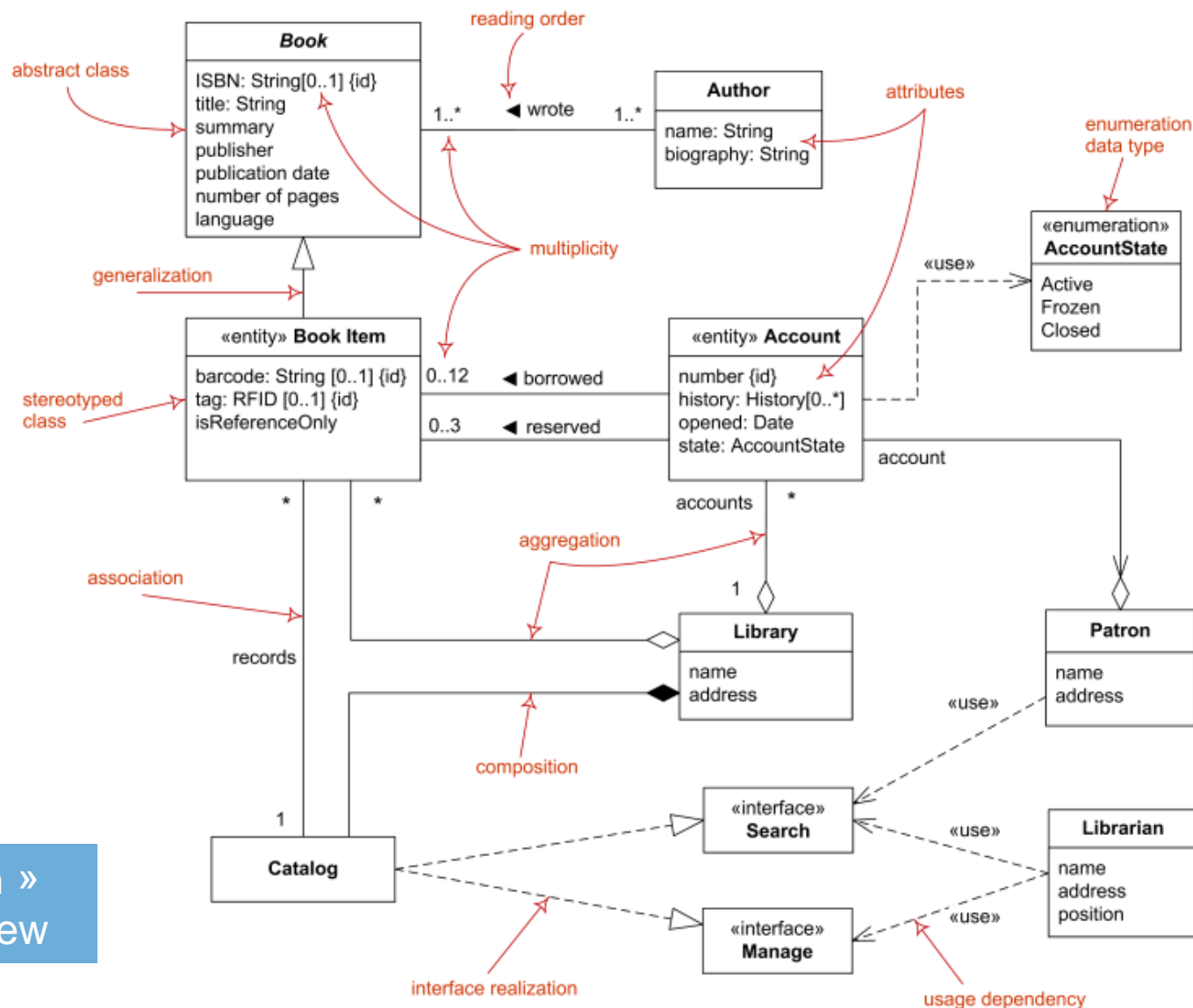


## Class diagrams (2/2)

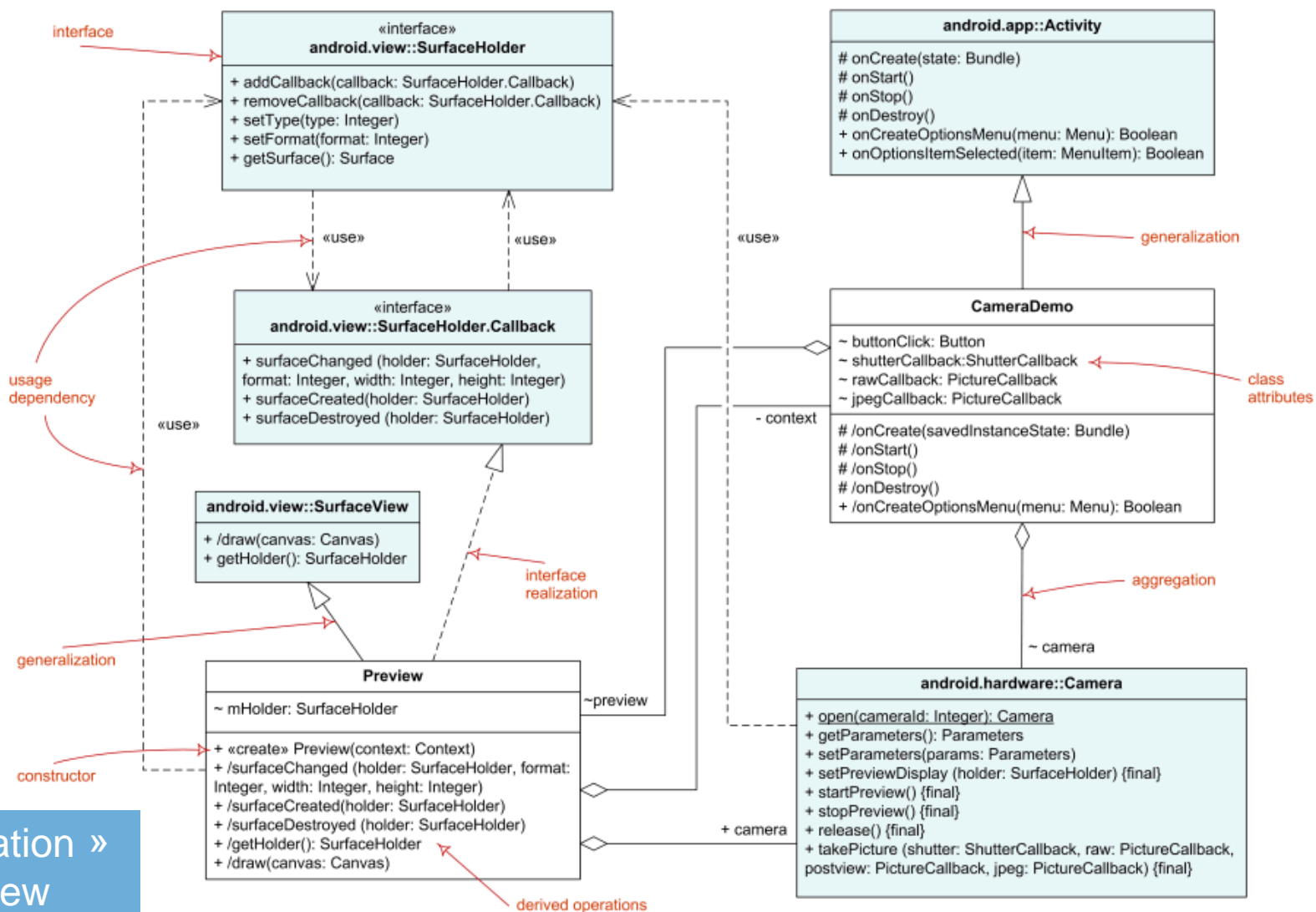
- How to design such a structure diagram (design the classes)?
  - Identify data types manipulated in the system
  - Identify the useful operations on these data ► classes (attributes and operations)
  - Identify constraints and relationships between these classes
  - Is it possible to “translate” the use cases using these classes through their relationships and potential synchronization ?
- Keep in mind the design guidelines



# Class diagram overview (1/2)



# Class diagram overview (2/2)

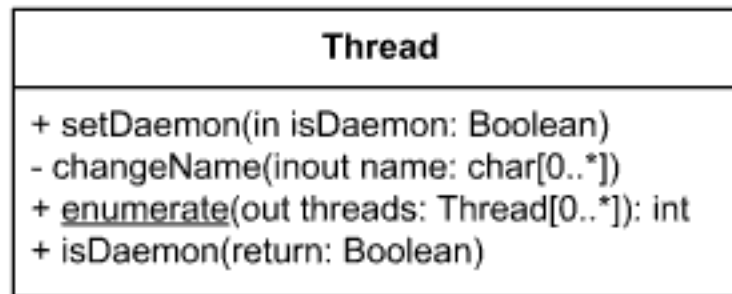
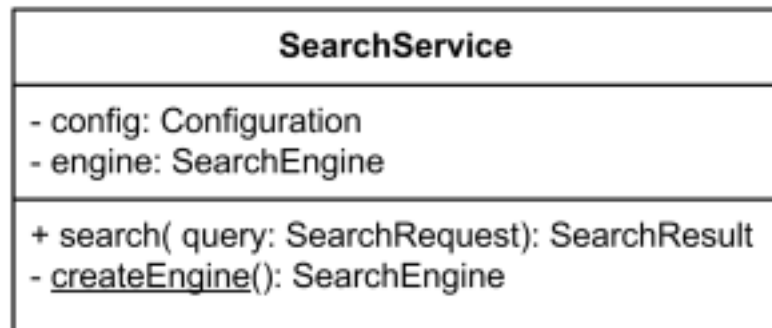


« Implementation »  
point of view

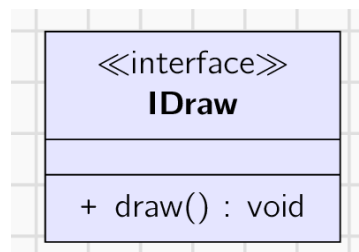
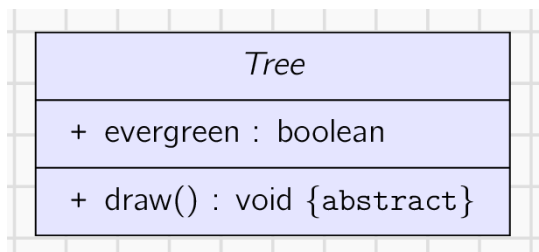
# Class diagram entities (1/3)

## • Classes

- Attributes & operations definition and visibility
- Regular classes ► object creation allowed



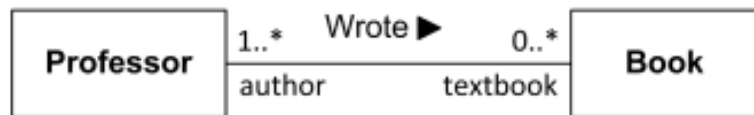
- Abstract classes, interfaces





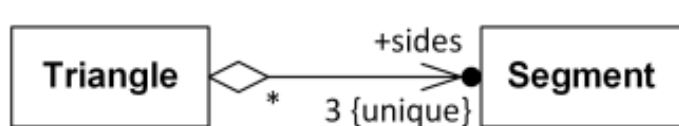
# Class diagram entities (2/3)

- Association relation
  - Type, navigability, arity, end ?
  - Association



*Professor "playing the role" of **author** is associated with **textbook** end typed as **Book***

- Shared aggregation (or aggregation)



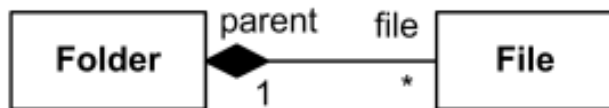
*Triangle has 'sides' collection of three unique line Segments.*

**Optional:**

*Line segments are navigable from Triangle.*

*Association end 'sides' is owned by Triangle, not by association itself.*

- Composite aggregation (composition)



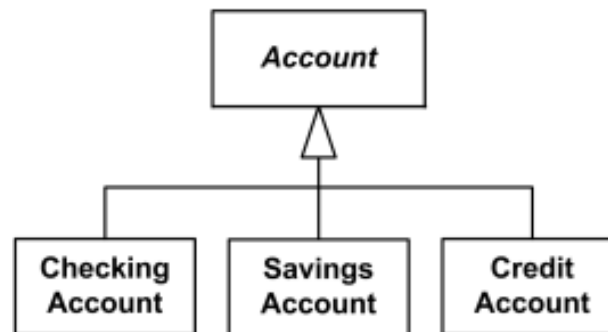
*Folder could contain many files, while each File has exactly one Folder parent.  
If Folder is deleted, all contained Files are deleted as well*



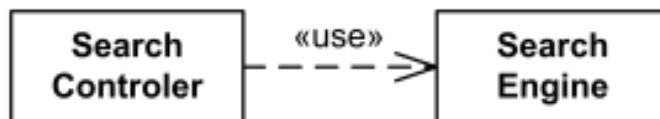
# Class diagram entities (3/3)

- Other relations

- Generalization



- Dependency



- Interface realization



# Application (3/6)

- **Class diagram from client requirements**

## Achat sur un site Web marchand (par Internet) :

Tout visiteur (même anonyme) peut parcourir le site et trouver des articles à acheter.

Pour effectuer un achat, le visiteur doit avoir un compte client et doit donc s'enregistrer auprès du site.

L'achat d'un ou plusieurs articles par un client consiste à trouver le ou les articles souhaités, les placer dans un panier et à effectuer le règlement qui correspond au prix des articles du panier.

Les articles montrés sont obligatoirement disponibles.

### Modélisation :

Chaque article a un id unique, un nom, un prix, une photo et une description.

Un panier peut contenir plusieurs articles (plusieurs fois le même et/ou différents articles).

Chaque client est défini par ses infos personnelles (nom, prénom, adresse, ...).

A chaque client est associé un compte unique qui possède un panier.

### **Question :**

Dessiner un diagramme de classes correspondant à la description donnée.



# Application (4/6)

## • Class diagram from client requirements

### Achat sur un site Web marchand (par Internet) :

Tout visiteur (même anonyme) peut parcourir le site et trouver des articles à acheter.

Pour effectuer un achat, le visiteur doit avoir un compte client et doit donc s'enregistrer auprès du site.

L'achat d'un ou plusieurs articles par un client consiste à trouver le ou les articles souhaités, les placer dans un panier et à effectuer le règlement qui correspond au prix des articles du panier.

Les articles montrés sont obligatoirement disponibles.

#### Modélisation :

Chaque article a un id unique, un nom, un prix, une photo et une description.

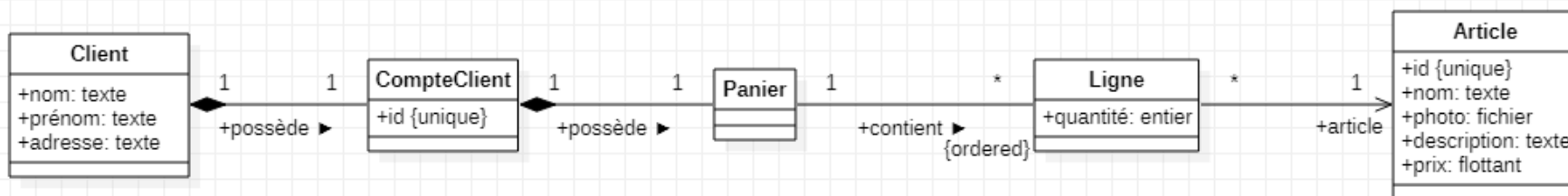
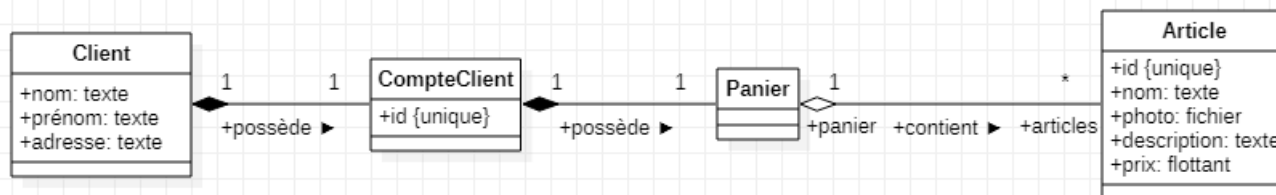
Un panier peut contenir plusieurs articles (plusieurs fois le même et/ou différents articles).

Chaque client est défini par ses infos personnelles (nom, prénom, adresse, ...).

A chaque client est associé un compte unique qui possède un panier.

#### Question :

Dessiner un diagramme de classes correspondant à la description donnée.



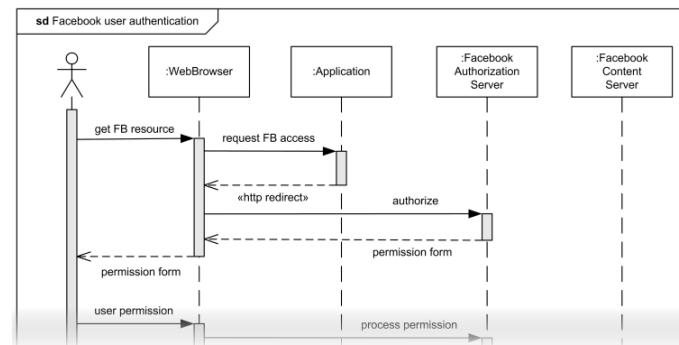
# Sequence diagrams

- **Part of interaction diagrams family**

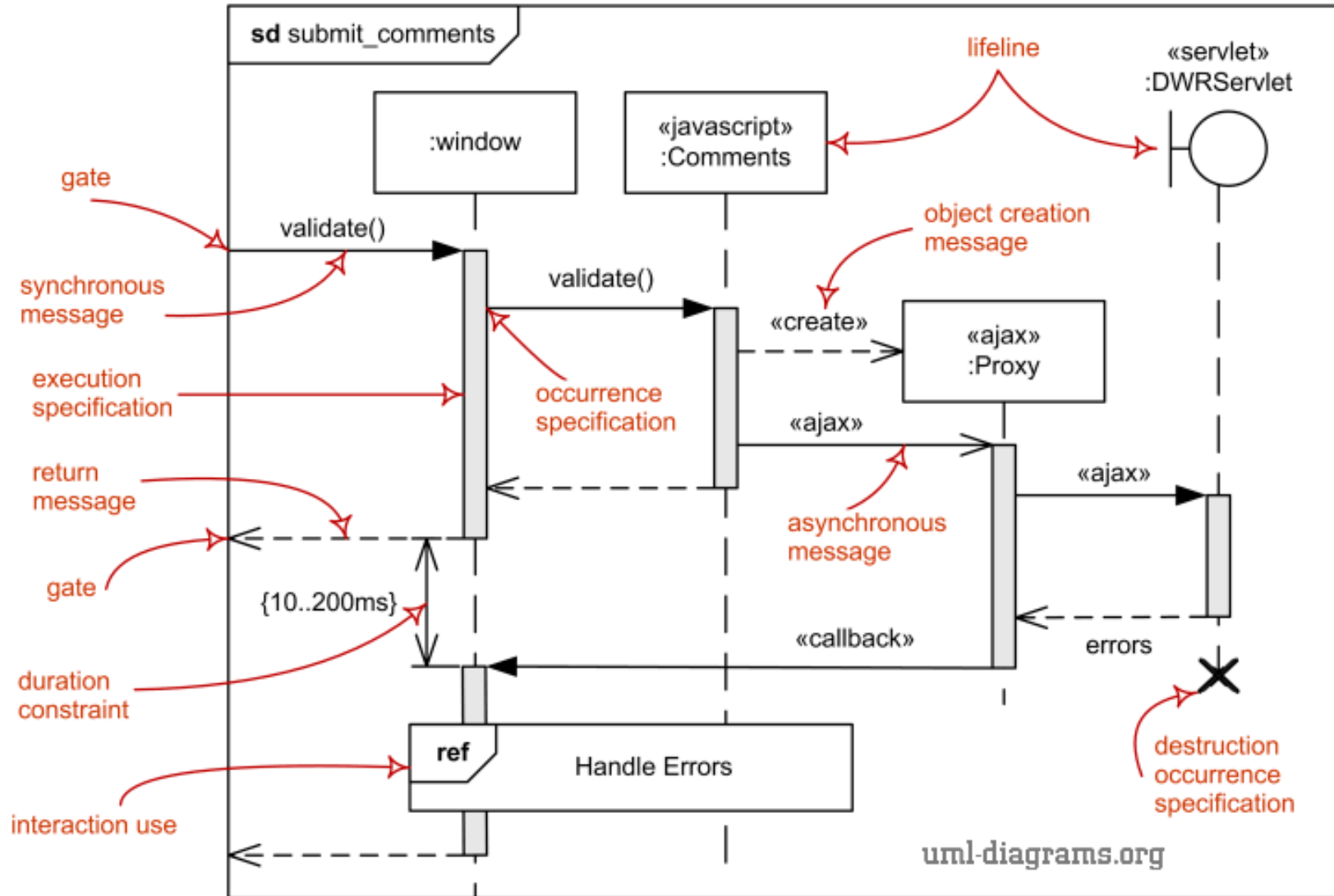
- Communication (collaboration) diagrams, state machine diagrams (protocol), timing diagrams, ...

- **Objectives**

- Represent a **series of interactions** between objects (and actors) in order to solve a specific functionality / goal
- Provide a **chronology** of these interactions through message exchanges arranged in time sequence
- Complementary diagram to use jointly with case & class diagrams ► add a time organized dimension to them



# Sequence diagram overview



# Sequence diagram entities (1/2)

## • Lifeline

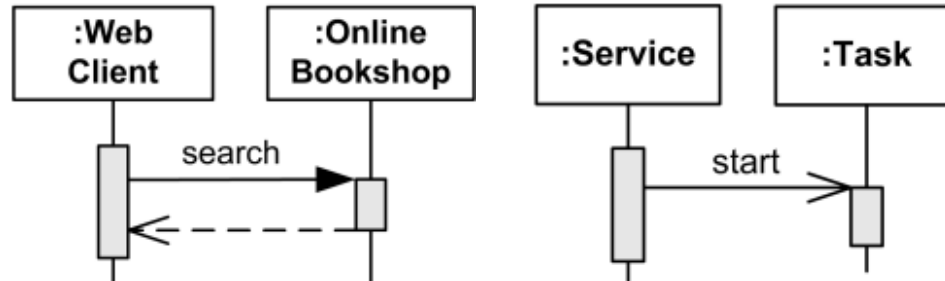
- Anonymous
- Named or with selector



## • Message & execution

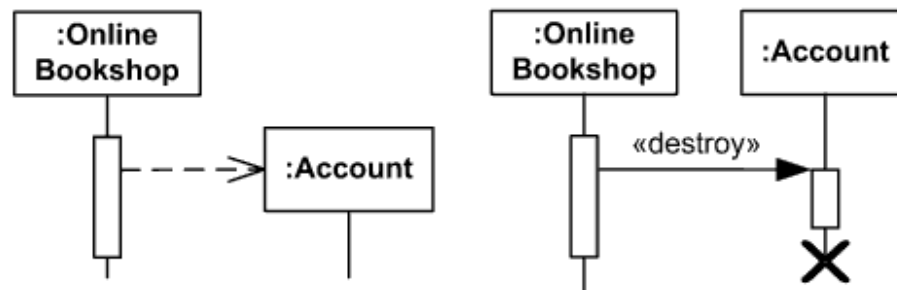
### • Synchronization

- Synchronous call
- Asynchronous call
- Reply message



### • Lifeline lifetime

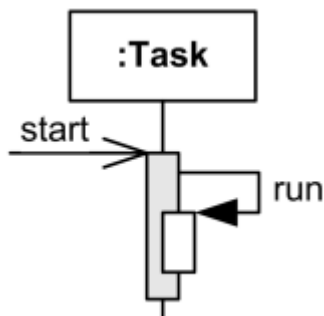
- Create
- Delete



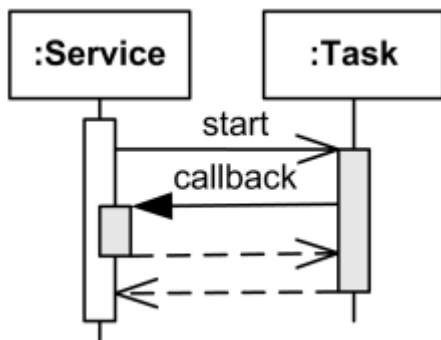


# Sequence diagram entities (2/2)

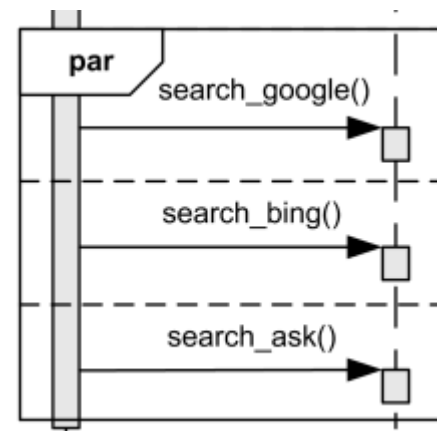
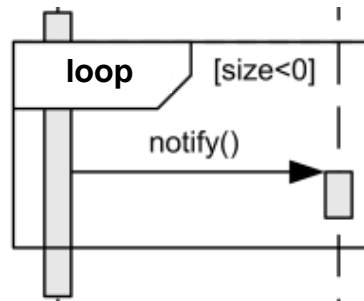
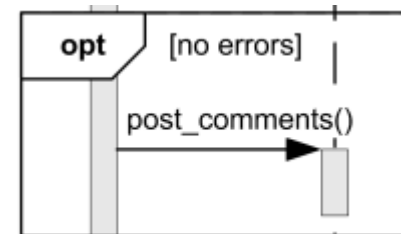
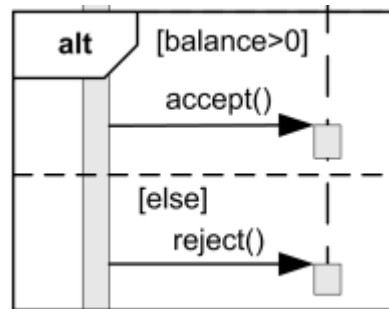
- Message to self



- Callback mechanism



- Combined fragments



# Application (5/6)

- Sequence diagrams from client requirements

## Achat sur un site Web marchand (par Internet) :

Tout visiteur (même anonyme) peut parcourir le site et trouver des articles à acheter.

Pour effectuer un achat, le visiteur doit avoir un compte client et doit donc s'enregistrer auprès du site.

L'achat d'un ou plusieurs articles par un client consiste à trouver le ou les articles souhaités, les placer dans un panier et à effectuer le règlement qui correspond au prix des articles du panier.

Les articles montrés sont obligatoirement disponibles.

### Question :

Dessiner un diagramme de séquence modélisant un achat sur ce site.



# Application (6/6)

## • Sequence diagrams from client requirements

Un client est déjà enregistré sur le site (≠ visiteur)

### Achat sur un site Web marchand (par Internet) :

Tout visiteur (même anonyme) peut parcourir le site et trouver des articles à acheter.

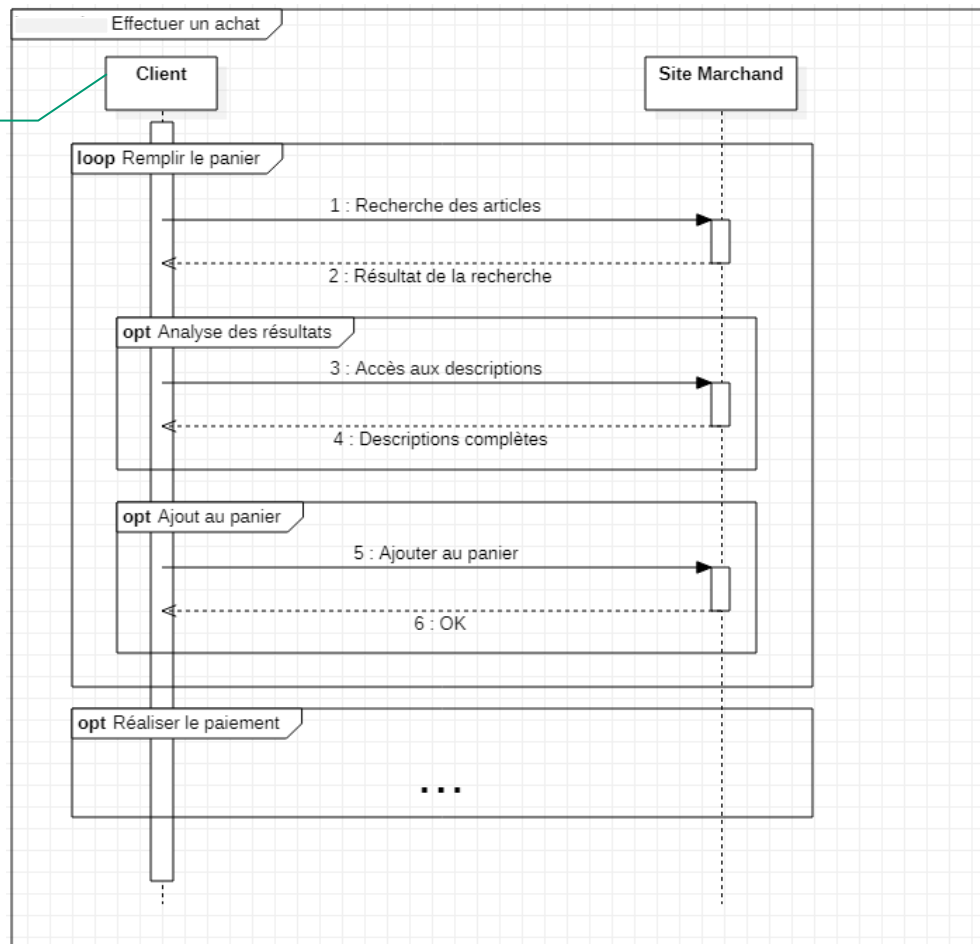
Pour effectuer un achat, le visiteur doit avoir un compte client et doit donc s'enregistrer auprès du site.

L'achat d'un ou plusieurs articles par un client consiste à trouver le ou les articles souhaités, les placer dans un panier et à effectuer le règlement qui correspond au prix des articles du panier.

Les articles montrés sont obligatoirement disponibles.

### Question :

Dessiner un diagramme de séquence modélisant un achat sur ce site.



# Diagrams consistency!

## • Several diagrams or views BUT one system only

- Complementary diagrams
- Name / signature must match between diagrams

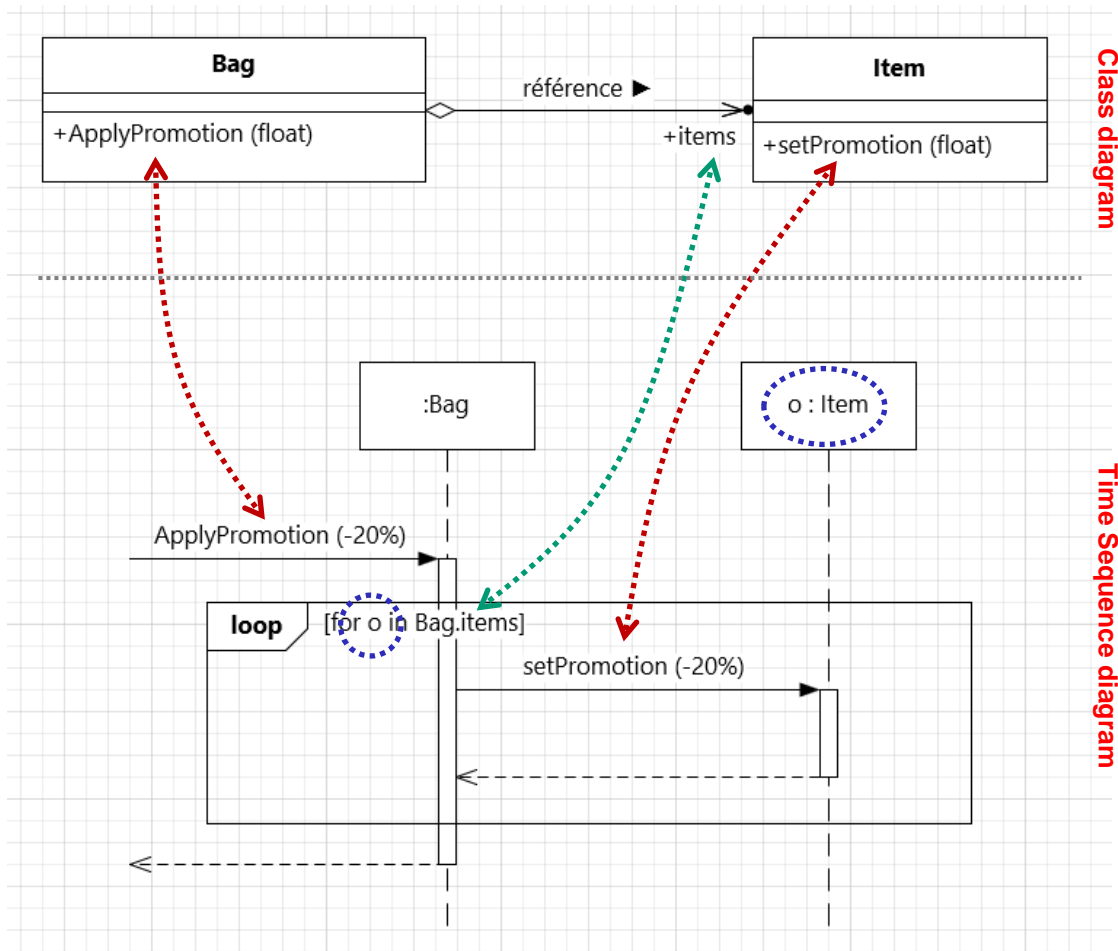
### Exemple :

Un « Bag » référence un ensemble d'items.

Appliquer une promotion de -20% sur un « Bag » conduit à fixer une promotion de -20% sur chaque item référencé par ce « Bag »

### Question :

Dessiner un diagramme de classes ET un diagramme de séquence modélisant un tel système.



# Practice (1/5)

## 1 Analyse d'un système bancaire

---

### *Diagramme UML : cas d'utilisation*

Pour le problème de traitement de données bancaires suivant, appliquez les premières étapes de la démarche d'analyse pour proposer un/des schéma(s) de cas d'utilisation :

Chaque banque fournit son propre ordinateur pour gérer ses propres comptes et ses propres transactions. Les différentes caisses sont la propriété des différentes banques et communiquent avec l'ordinateur de la banque. Les caissiers entrent les numéros de compte et les données de transaction.

Les caisses automatiques communiquent avec un ordinateur central qui route les transactions vers la banque appropriée. La caisse automatique accepte les cartes de crédit, interagit avec l'utilisateur, communique avec l'ordinateur central pour effectuer la transaction, délivre l'argent et imprime un reçu.

Les banques fournissent leur propre logiciel pour leurs propres ordinateurs. Vous aurez donc à définir uniquement le diagramme de gestion des guichet automatique de banque (GAB, distributeur de billet ... enfin les machines à cracher des sous, quoi :) !) et du réseau.



# Practice (2/5)

## 2 Les objets, c'est la classe !

Diagramme UML : diagramme de classes

... « je sais, elle est nulle ! »

Dessiner les diagrammes de classes correspondant aux situations suivantes :

- (a) La France est frontalière de l'Espagne. L'Algérie est frontalière du Maroc. Pour ce cas, proposez également un diagramme d'objets possible (une instanciation des classes soit représentative de la situation décrite).
- (b) Un polygone est constitué de points. Un point possède une abscisse et une ordonnée.
- (c) Une médiathèque possède des médias, empruntables par les abonnés.
- (d) Un client demande une réparation. Une réparation est effectuée par un mécanicien. Elle nécessite des compétences, que possède le mécanicien.
- (e) Une galerie expose des œuvres, faites par des artistes, et représentant des thèmes. Des clients, accueillis par la galerie, achètent des œuvres.
- (f) Un dessin est une forme géométrique, un texte, ou un groupe de dessins.

... « mais je fais ce que je veux !!! » :)

- (g) Un hôtel emploie du personnel et loge des clients





# Practice (3/5)

## 3 Architectures objets simples

---

### *Diagramme UML : diagramme de classes*

Modélisez les situations (indépendantes) suivantes :

1. Un éditeur de documents graphiques supporte le groupement d'objets graphiques. Un document se compose de plusieurs feuilles, chacune contenant des objets graphiques (texte, forme géométrique et groupe d'objets). Un groupe est un ensemble d'objets pouvant contenir d'autres groupes. Un groupe doit contenir au moins deux éléments. Les formes géométriques comprennent les cercles, les ellipses, les rectangles, les carrés, les lignes ...
2. Une personne physique peut avoir jusqu'à trois sociétés (personnes morales) qui l'emploient. Chaque personne physique possède un numéro de sécurité sociale qui l'identifie. Une voiture a un numéro d'immatriculation. Une voiture est la propriété d'une personne (physique ou morale). Un emprunt dans une banque peut être demandé pour l'achat d'une voiture.

### Quelques éléments de réflexion :

1. Dans un système domotique, un interrupteur peut allumer et éteindre une lampe.
2. Un carré est un rectangle dont les 4 cotés ont la même longueur.





# Practice (4/5)

## 4 Retrait d'argent au guichet automatique

---

*Diagramme UML : diagrammes de classes et de séquence*

Reprendre l'énoncé de l'exercice 1. Complétez l'étude en déduisant un diagramme de classes (logiciel actif sur le distributeur) et un diagramme de séquence (scénario de retrait d'argent au guichet) à partir de vos cas d'utilisation.

# Practice (5/5)

## Modélisation statique et dynamique :

Le logiciel PASS assure la gestion des notes obtenues par les élèves d'une école. Il permet donc à un membre de la scolarité d'avoir accès à la liste des notes obtenues par un élève (via son nom), pour une matière donnée (définie par une chaîne de caractères).

Pour ce faire, le PASS est en relation avec deux entités :

- le *Student Directory* qui contient pour chaque élève, son nom et son ID (un entier),
- le *Marks Data Storage* qui contient les notes de tous les élèves par matière ; les élèves y sont référencés par leur ID et non par leur nom (Le *Marks Data Storage* ne contient donc pas le nom des élèves).

Donner un diagramme de classes représentant l'ensemble du système décrit ; ne pas oublier d'y inclure pour chaque classe, les attributs et méthodes pertinents.

Donner le diagramme de séquences correspondant à l'interrogation de PASS par un personnel de la scolarité pour récupérer les notes de l'élève "Bjarne Stroustrup" dans la matière "Advanced C++ Programming".

# Design patterns (1/2)?

- **Architecture building (after functional analysis)**

- ▶ Some standard steps, for a subset of related functions:

- 1) Identify the data to be manipulated

- 2) Identify the **useful** operations on these data

- 3) Identify relations between classes (constraints, ...)

- 4) Translate scenarios / algorithms:

- ▶ use the relations and synchronize all the interactions

Class diagram

Sequence diagram

- **Standard solutions for architecture problems**

- Goal: to face recurrent issues, through object paradigm

- Design patterns are **models** for proposing « good » solutions to identified problems (or sub-problems)

- ▶ however, no implementation enforced (the developer is free to make relevant technical choices)



# Design patterns (2/2)?

- **Three main families of patterns**

- Creational patterns ▶ **Object creation mechanisms**
- Structural patterns ▶ **Classes assembly into larger structures**
- Behavioral patterns ▶ **Algorithms and assignment of responsibilities between objects**

- **They help the design (and then the dev) process**

- Make the design faster and safer as it relies on tried and tested blocks
  - ▶ **Do not reinvent the wheel / avoid following “antipatterns”**
- Make the decomposition easier to design, understand and share as these blocks have well-known purposes and behaviors
  - ▶ **High level design**
- Help focusing on value-added business topics
  - ▶ **Actual problem to solve?**



# Antipatterns (1/2)

- **Definition**

- “Common defective processes and implementations within organizations” [SourceMaking]
- It looks like a “solution” but it will generate negative consequences (good pattern in the wrong context, lack of knowledge or experience in solving a specific problem, ...)
  - ▶ Learn to **identify** them within a beginning or on-going process

- **Where can they be found?**

- Project management
- Architecture
- Development



**Refactoring plan!**



# Antipatterns (2/2)

- Some classic examples:

- See [SourceMaking] (with **applied refactoring solutions**)
- Project management
  - “Analysis paralysis” (redoing / rebooting the analysis phase)
  - “Death by planning” (over planning)
- Architecture
  - “Swiss army knife” or “Kitchen sink” (huge amount of method signatures for one class ...  $\neq$  one class, one purpose)
  - “Reinvent the wheel” (“our problem is unique”)
- Development
  - “Spaghetti code” (unstructured code, lack of clarity, ...)
  - “Poltergeist classes” (short lifetime, too limited responsibilities, ...)
  - “Boat anchor” (code with no useful purpose for the problem)
  - “Cut-and-paste programming” (degenerate form of software reuse)



# Exploring some design patterns

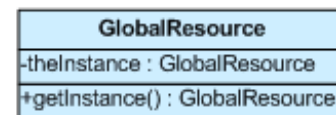
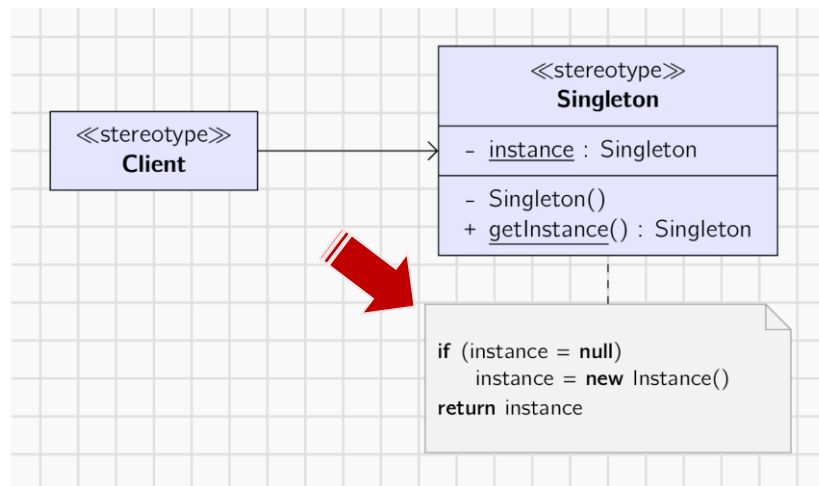
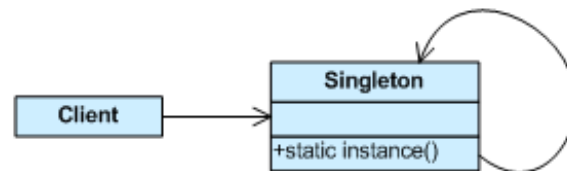
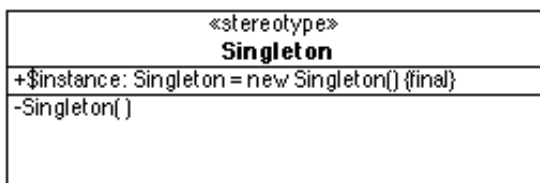
- **Three main families of patterns**
  - **Creational patterns** ▶ **Object creation mechanisms**
    - Singleton,
    - Factory method,
    - ...
  - **Structural patterns** ▶ **Classes assembly into larger structures**
    - Adapter,
    - Composite,
    - ...
  - **Behavioral patterns** ▶ **Algorithms and assignment of responsibilities**
    - Observer,
    - Strategy,
    - ...



# Singleton

Creational

- **Goal**
  - Only **one** object can be created for the class
  - Give an access to this unique object



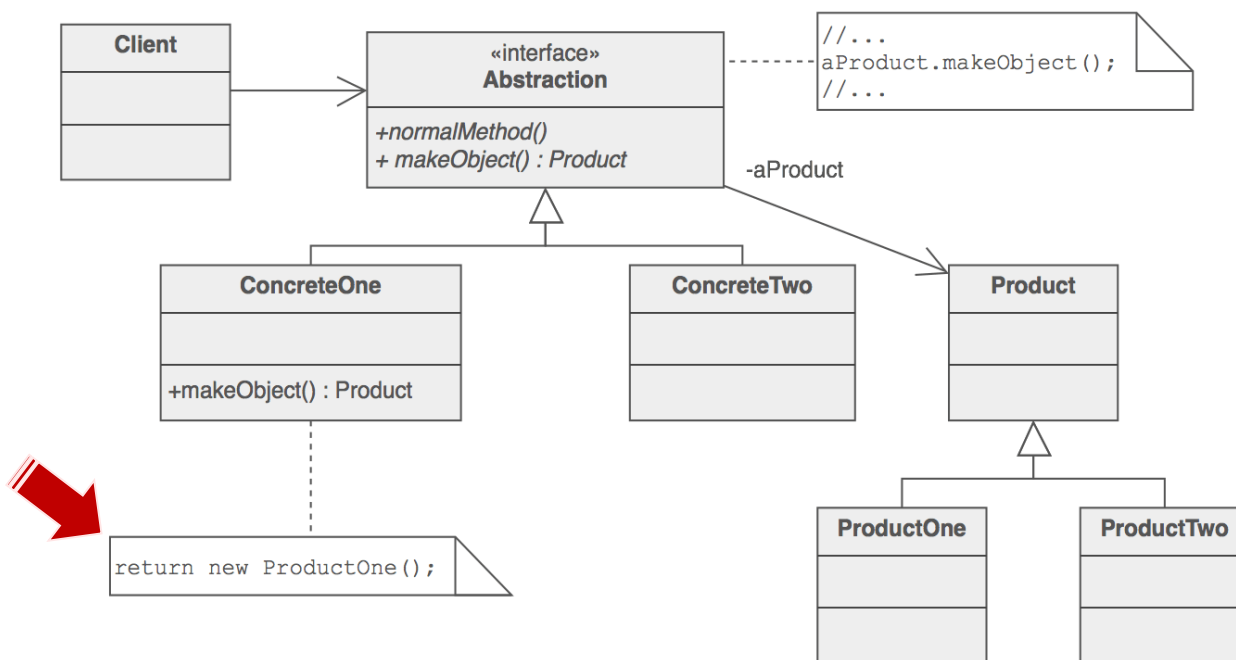


# Factory method

Creational

- **Goal**

- Generic interface for **creating** objects (in a superclass)
- Allow subclasses to alter the type of objects being created

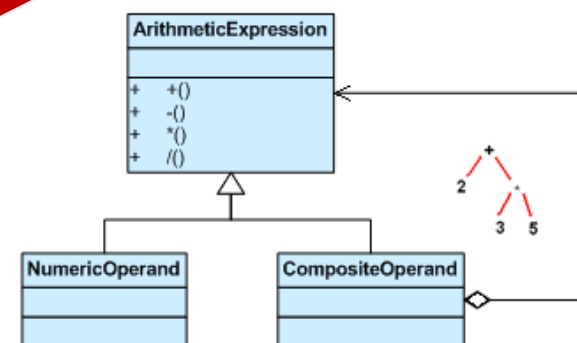
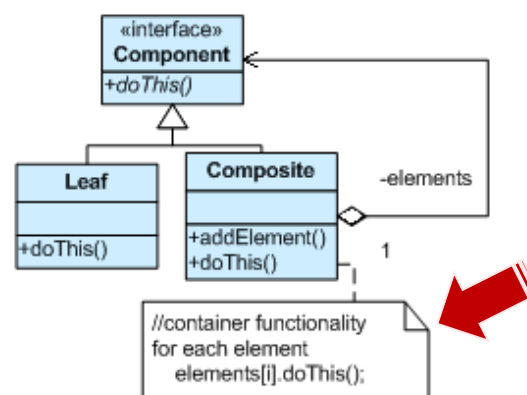
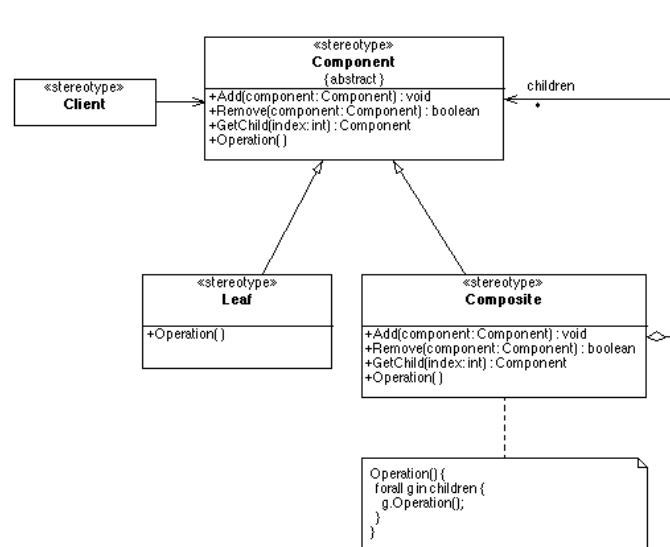


# Composite

Structural

## • Goal

- Interface to manipulate a set of objects part of a hierarchy, as one object
- Allow defining how methods should behave (leaf or node)

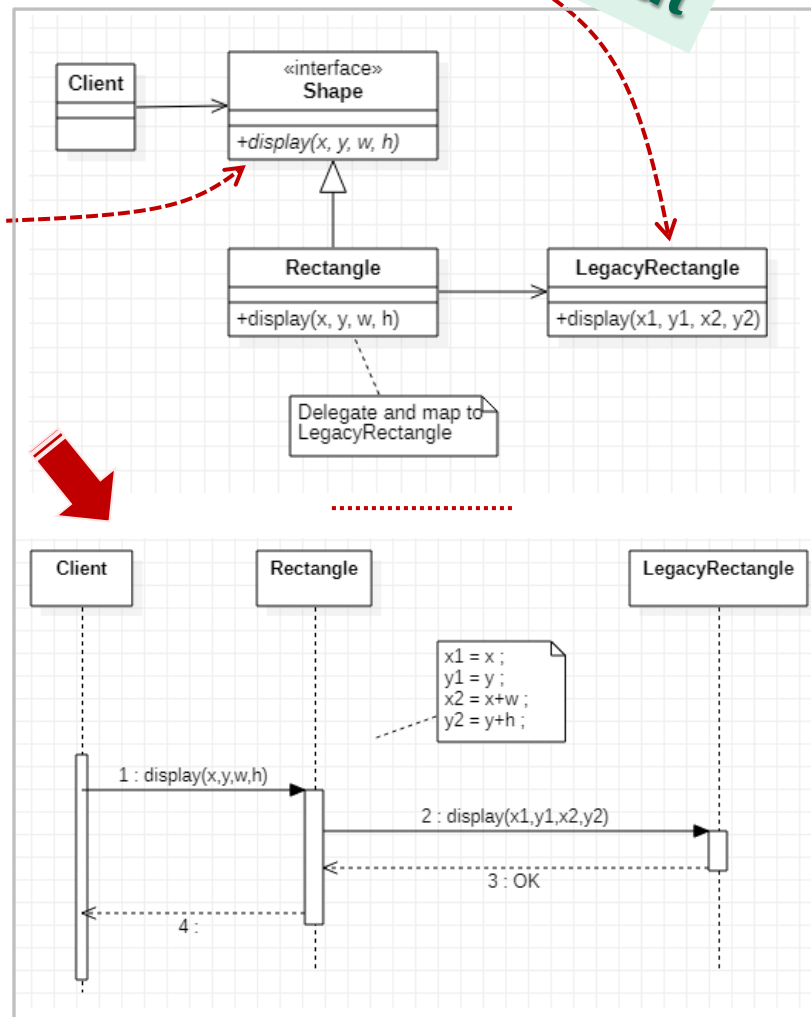
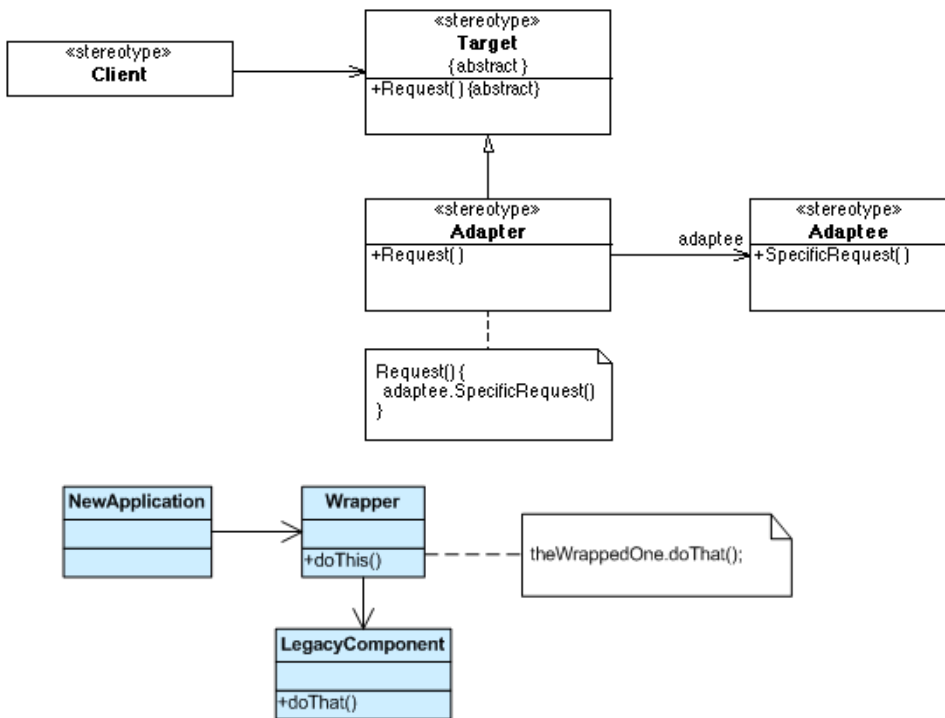


# Adapter

Structural

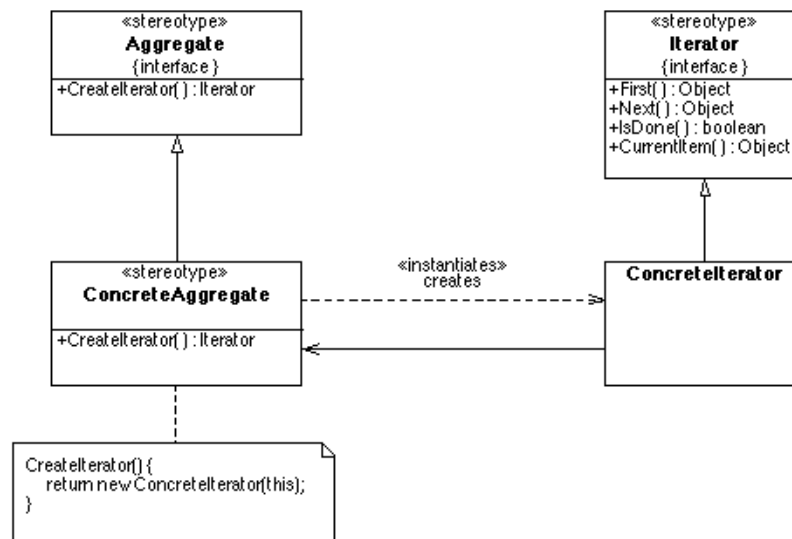
## • Goal

- “Old” API access encapsulation
- Bend an “old” API to your need



## Behavioral

- Give access to each element within a collection
- Generic interface to browse the collection (actual implementation hidden) ► **STL**

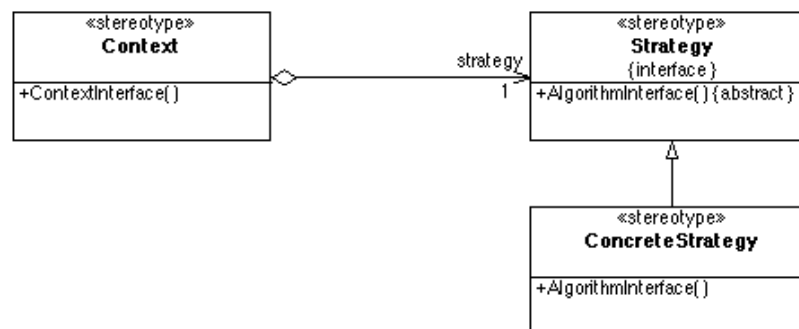


# Strategy

Behavioral

- **Goal**

- Encapsulating **algorithms** in classes sharing the same interface
- Make easier to add new algorithms, to update existing algorithms as the generic interface remains unchanged



# Useful references

- **Practice on one specific design pattern**
  - Select a practical and simple problem
  - Design a solution using this design pattern
  - Code the solution using C++
  - Make a simple demonstration illustrating/proving it is interesting to use this design pattern for your problem
- **Useful references**
  - [SourceMaking] Antipatterns & refactoring:  
<https://sourcemaking.com/antipattern>
  - [CodeGuru] Design patterns:  
<https://refactoring.guru/design-patterns>

