



Résumé du cours

1I2AC1 : Génie logiciel et Conception orientée objet

Régis Clouard, ENSICAEN - GREYC

Génie logiciel / Software engineering

2

- Le génie logiciel est une science de génie industriel qui étudie les **méthodes** de travail et les **bonnes pratiques** des ingénieurs pour développer des logiciels de grande taille (plusieurs années -hommes) :
 - correspondant aux attentes du client
 - fiables
 - performants
 - respectant les délais et les coûts
 - coût de maintenance réduit

Génie logiciel aujourd'hui

3

- 1 **Paradigme** : conception orientée objet
- 2 **Formalisme** : UML
- 3 **Méthode** : Agilité

Génie logiciel aujourd'hui

4

- 1 Paradigme : conception orientée objet**
- 2 Formalisme : UML
- 3 Méthode : Agilité

Paradigme : Conception par objet

5

- Vision duale du paradigme procédural
 - ▶ On s'intéresse aux données et plus aux procédures.
- **Objet** : entité capable de réaliser des services
- **Programme** : coopération d'objets entre eux par échange de services
- **Programmation** : quels objets seraient nécessaires à la réalisation du logiciel ?

Paradigme : Conception par objet

6

- **6** concepts de base :

- 1/ **Modularité** → fonctions, classes, paquets, nœuds

- 2/ **Abstraction** → simplification du problème à l'essentiel.

- 3/ **Encapsulation** → protéger l'implémentation d'une classe du regard extérieur

- ▶ **Classe** : un ensemble de services (méthodes publiques).

- ▶ **Attribut** : n'existe qu'au moment de la programmation de la classe et parce que la réalisation d'un service le réclame.

- Un attribut (donnée membre) n'existe pas en conception orientée objet.

Paradigme : Conception par objet

7

- Les concepts de base (suite) :

4/ **Héritage** → construction d'une classe par généralisation / spécification d'une autre classe.

5/ **Polymorphisme** → dans une hiérarchie, le même service peut avoir un comportement différent selon l'objet réel qui est appelé.

→ programmation au niveau le plus abstrait.

6/ **Association** → relation entre classes

- ▶ Types : normal, agrégation, composition.
- ▶ NB : En conception, une association n'est pas un attribut.

Génie logiciel aujourd'hui

- 1 Paradigme : conception orientée objet
- 2 **Formalisme : UML**
- 3 Méthode : Agilité

Formalisme : UML

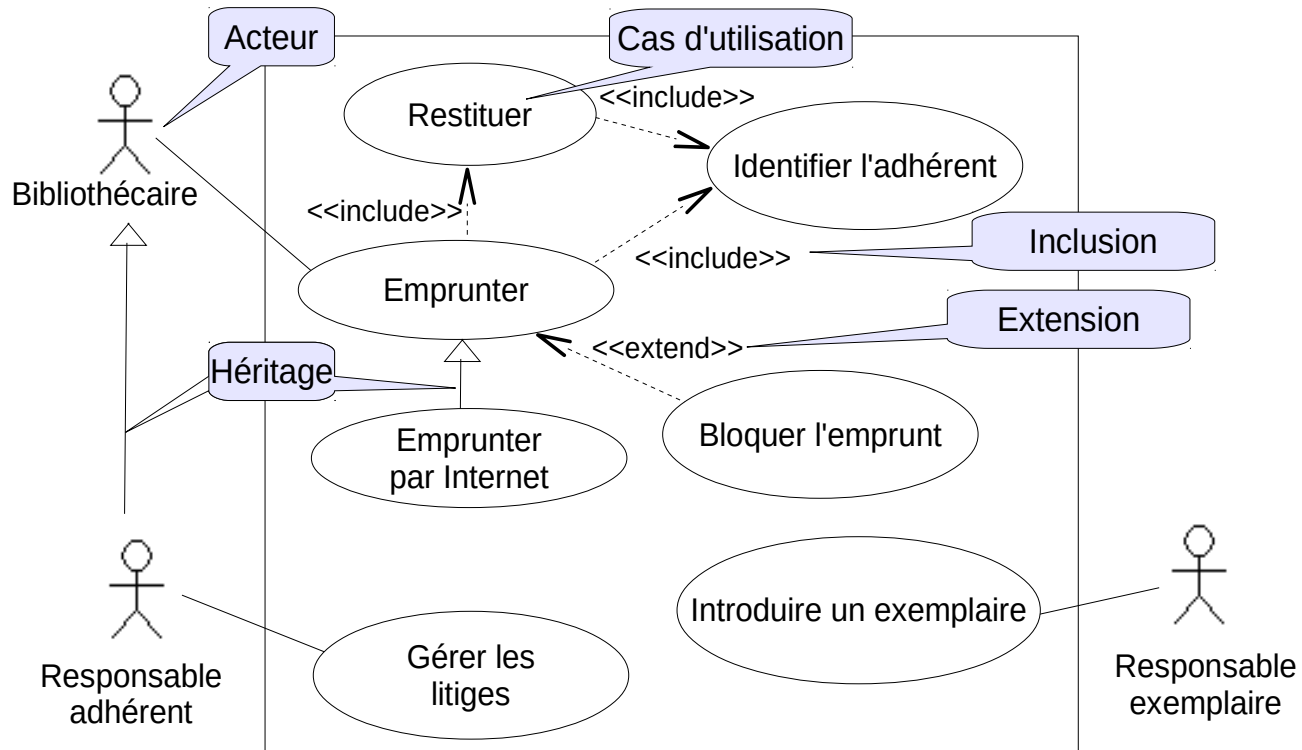
- Langage qui permet de parler de la conception d'un logiciel :
 - **Haut niveau** : plus abstrait que l'algorithmique. Il permet d'exprimer les besoins et les solutions.
 - **Diagrammatique** : peu de syntaxe.
 - **Normalisé** : partagé par tous les développeurs du monde.
 - **Translangage** : y compris les langages non orientés objet (p. ex. langage C).

Formalisme : UML

- 6 diagrammes principaux :
 - 1) Cas d'utilisation
 - 2) Classe
 - 3) Paquet
 - 4) Activité
 - 5) Séquence
 - 6) États-transitions

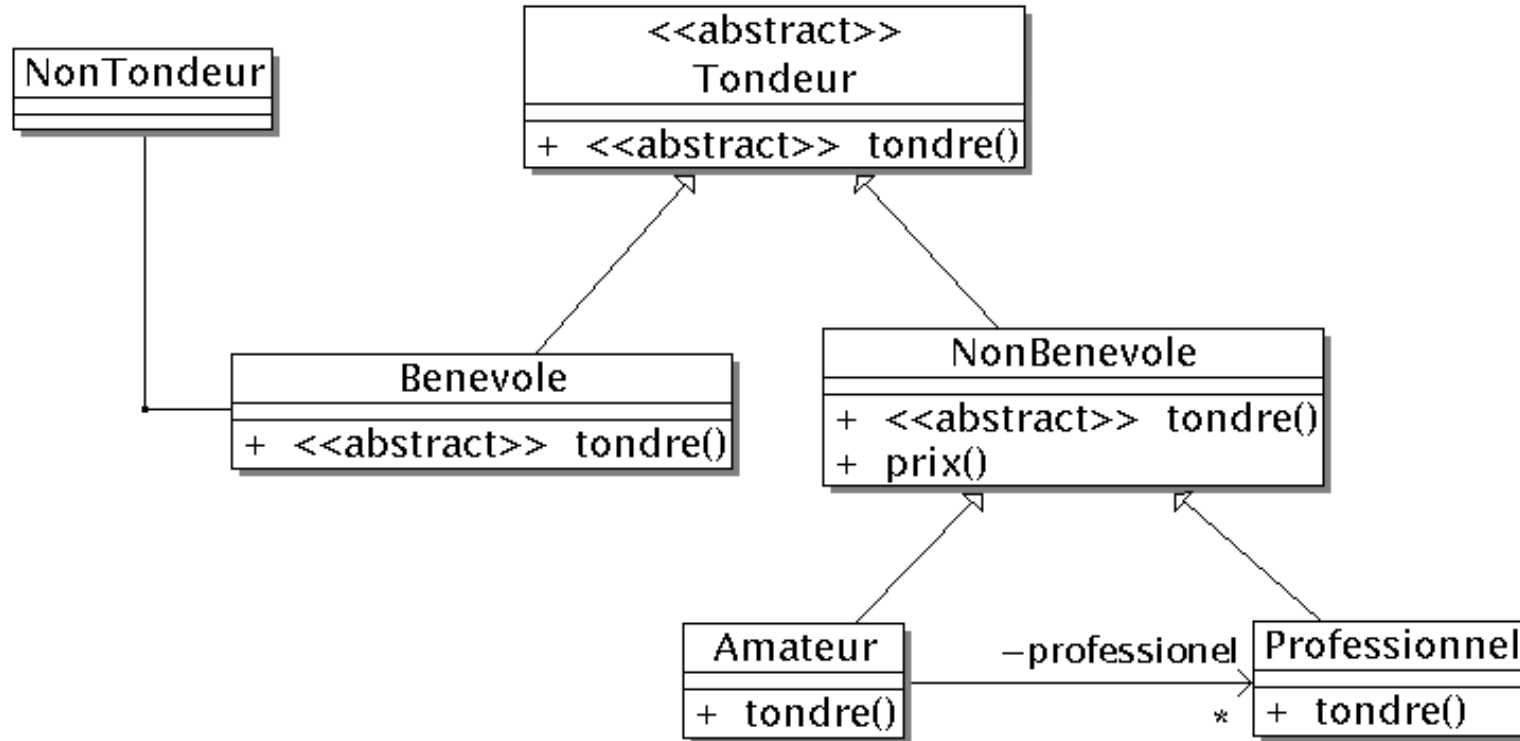
1- Diagramme des cas d'utilisation

- Définir le problème par les interactions entre les acteurs extérieurs et le système à développer.



2- Diagramme de classes

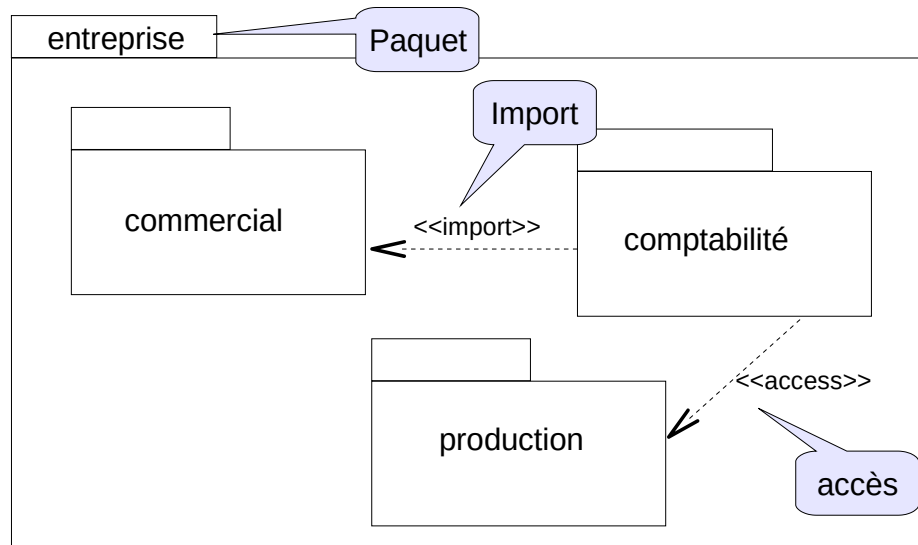
- **Vision statique du logiciel** → modéliser les classes qui seraient nécessaires à la réalisation du logiciel.



- NB : Ne pas mettre les attributs → rompt l'encapsulation

3- Diagramme de paquets

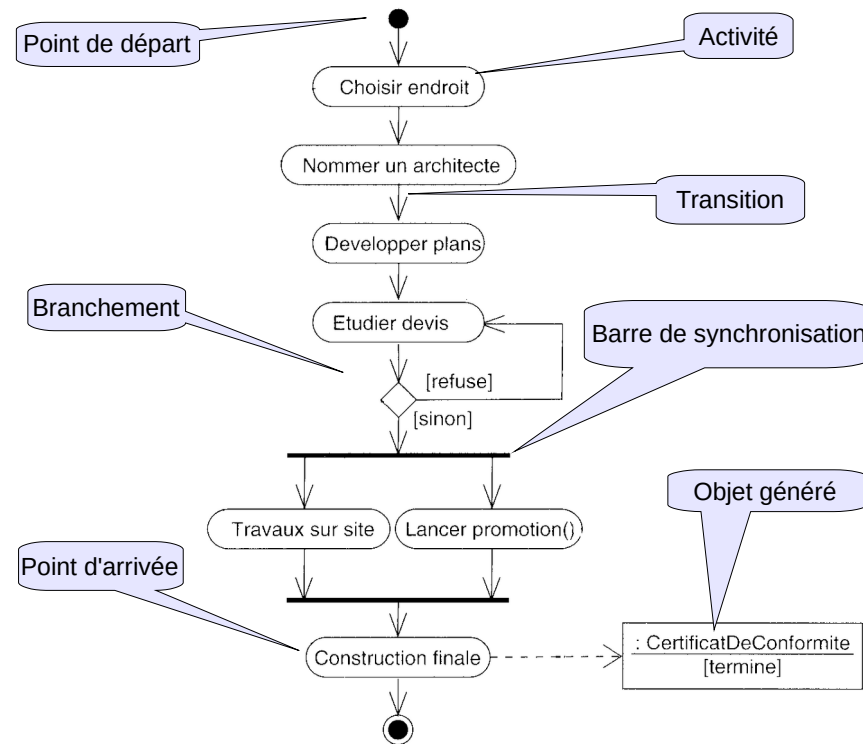
- **Vision organisationnelle du logiciel** → importance pour la vision et l'organisation du développement.



- Les paquets en Java : **fr.ensicaen.ecole.projet**

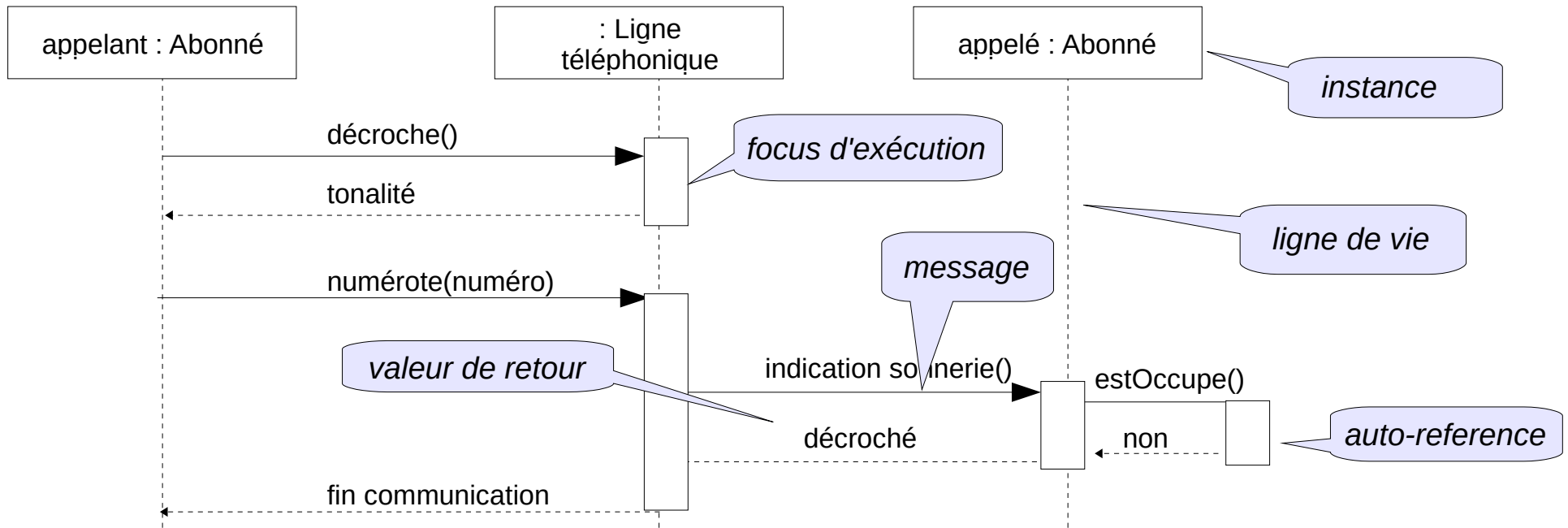
4- Diagramme d'activités

- **Vision dynamique** → décrire les activités liées à la réalisation d'un cas d'utilisation ou d'un algorithme.
 - NB : ne fait pas intervenir les classes.



5- Diagramme de séquence

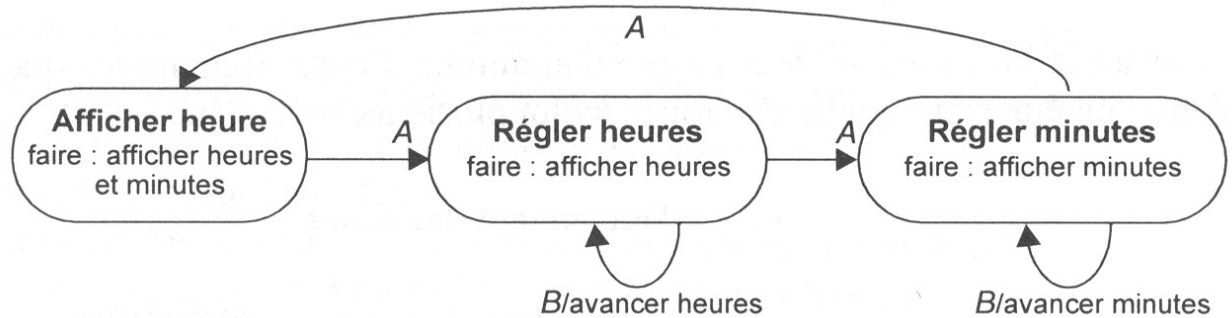
- **Vision dynamique** → identifier l'échange de services entre les classes nécessaires à la résolution d'un cas d'utilisation.



6- Diagramme d'états-transitions

- **Vision dynamique liée à une classe** → modéliser les états que peuvent prendre une classe et qui induisent une variation de ses services.

| |
|--|
| Montre |
| + appuyerBoutonA() + appuyerBoutonB() |



Génie logiciel aujourd'hui

17

- 1 Paradigme : conception orientée objet
- 2 Formalisme : UML
- 3 **Méthode : Agilité**

Méthode : Agilité

18

- **Itération** : Découper le temps de développement en courtes périodes de temps de durée fixe.
 - Rythme régulier.
 - Planifier court : adaptabilité.
 - Une itération est un mini-projet à part entière sur 2 semaines.
 - ▶ Une liste de tâches réalisables en 2 semaines.
 - ▶ Une version opérationnelle **montrable** au client.
 - Code propre
 - Code testé

Méthode : Agilité

- **Incrémentation** : MVP (Minimum Valuable Product)
 - Une voiture commence par un skate-board.
 - Produit orienté vers l'utilisateur.
 - Reprise de code continue : malléabilité du code.

Les méthodes agiles

- Personnellement, deux méthodes agiles conjointes :
 - **SCRUM** : axé sur l'organisation du projet (planification, réunions)
 - ▶ (Voir le cours « ingénierie des systèmes logiciels » l'année prochaine)
 - **eXtreme Programming (XP)** : axé sur le développement
 - ▶ Le code est la seule chose qui compte.

XP : eXtreme Programming

■ Développement

- Centré sur le code.
- Rendre la documentation superflue
 - ▶ Code-auto-documenté
 - ▶ Si besoin de documentation utiliser la rétro-ingénierie.
- Centré sur les besoins des utilisateurs
 - ▶ L'utilisateur est un partenaire du projet (retour fréquent voire quotidien).
 - ▶ Toujours avoir un produit exécutable à montrer au client.

■ Tests automatiques

- Limiter les bugs à partir des cas de test liés au contrat de ce que l'on teste.
- Éviter que les mêmes bugs se reproduisent.
- Installer des gardes-fous pour se prémunir contre la régression.

exTreme Programming

- Pratique du développement classique
 - 1) Quick and dirty
 - 2) Test
 - 3) Refonte
- TDD
 - 1) Test
 - 2) Quick and dirty
 - 3) Refonte
- Développement des tests
 - TDD : quand la fonctionnalité est claire.
 - Classique : quand la fonctionnalité est à découvrir.

Artisan du code

- Être un artisan du code, c'est :
 - Maîtriser son environnement de développement (IDE + outils)
 - ▶ Katas.
 - Amour du code propre.
 - Accepter les critiques sur son code dans le but de s'améliorer.
 - Partager son savoir.
 - Chercher à rendre la documentation et les commentaires superflus dans ses développements.

Examen

-
- Une feuille A4 manuscrite recto verso
 - Partie cours / partie exercices