ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Chapitre 3
Le langage UML

1I2AC1 : Génie logiciel et Conception par objet

Régis Clouard, ENSICAEN - GREYC

« Il existe deux manières de concevoir un logiciel.
La première, c'est de le faire si simple qu'il est évident qu'il ne présente aucun problème.
La seconde, c'est de le faire si compliqué qu'il ne présente aucun problème évident.
La première méthode est de loin la plus complexe. »

Antony R. Hoare, prix Turing 1980

Objectif du chapitre

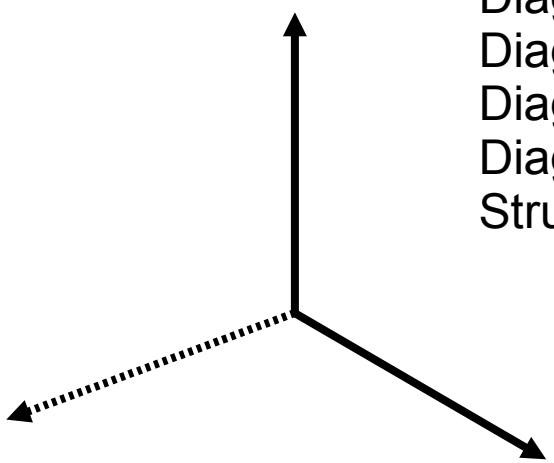
- Présentation du langage UML à travers ses principaux diagrammes.
  - Démonstration de la puissance du langage pour décrire un logiciel malgré sa simplicité.
- À l'issue de ce chapitre, vous serez en mesure de :
  - Écrire une modélisation partielle ou complète.
  - Relire une modélisation.
  - Échanger sur un problème ou sur une solution de modélisation avec d’autres développeurs.

UML

- Langage de modélisation de logiciels.
  - Orienté objet.
  - Diagrammatique.
  - Normalisé (OMG).
  - Indépendant des langages de programmation.
- Importance
  - Il faut utiliser le langage UML pour parler des logiciels.
  - Il est partagé par toute la communauté mondiale des développeurs.

Trois vues sur la modélisation

- 13 diagrammes



Structurel

Diagramme de classes
Diagramme d'objets
Diagramme de paquets
Diagramme de composants
Diagramme de déploiement
Structure de composite

Fonctionnel

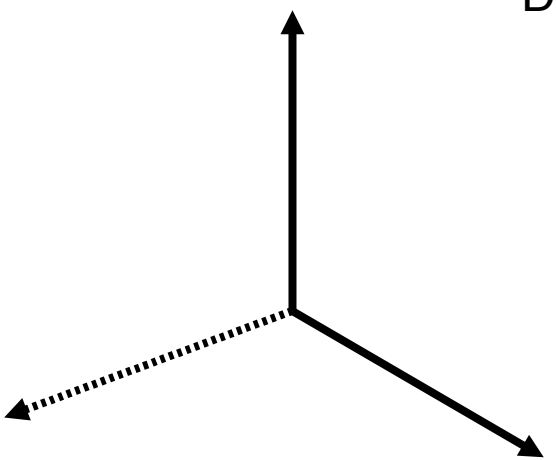
Diagramme des cas d'utilisation

Comportemental

Diagramme d'activités
Diagramme de séquence
Diagramme de communication
Diagramme d'états-transitions
Vue d'ensemble des interactions
Diagramme de timing

Trois vues sur la modélisation

- 6 diagrammes principaux



Structurel

Diagramme de classes
Diagramme de paquets

Fonctionnel

Diagramme des cas d'utilisation

Comportemental

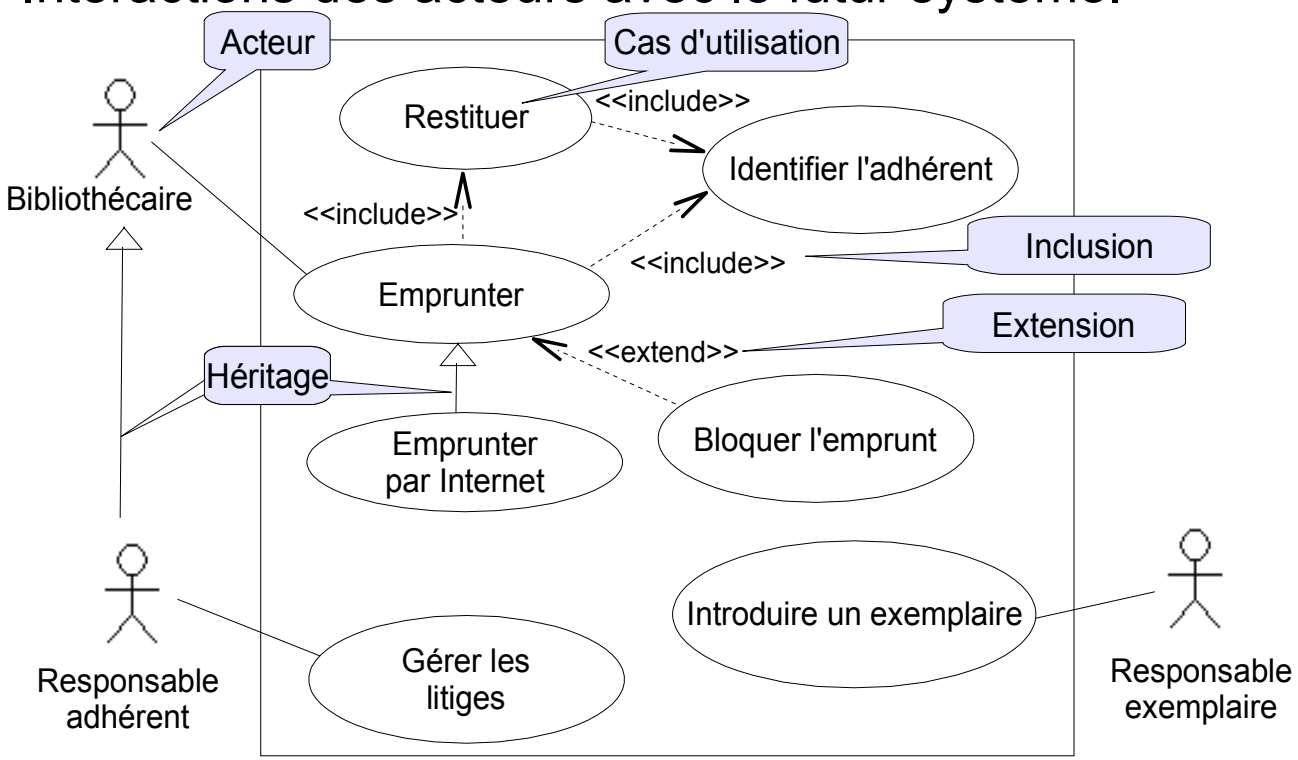
Diagramme d'activités
Diagramme de séquence
Diagramme d'états-transitions

Six diagrammes principaux

- **Diagramme des cas d'utilisation**
  - Définition du problème.
- **Diagramme de classes**
  - Description statique du logiciel.
- **Diagramme de paquets**
  - Description organisationnelle du projet.
- **Diagrammes de séquence** / communication
  - Description de la séquence de messages échangés pour réaliser une fonctionnalité (analyse des besoins et conception).
- **Diagramme d'activités**
  - Description d'un algorithme.
- **Diagramme d'états-transitions**
  - Description du comportement d'une classe en fonction de son état.

1- Diagramme des cas d'utilisation

- Aspect fonctionnel
  - **Intention** : Définir les fonctionnalités du futur système.
  - Vision anthropo-centrée du logiciel.
    - Interactions des acteurs avec le futur système.

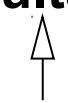


Syntaxe

- Acteur 
  - Interagit avec le système (humain ou un système que l'on développe pas)
- Cas 

  - Fonctionnalité du système.
- Relations
  - include 

    - Inclure **obligatoirement** un sous-cas.
  - extends 

    - Inclure **facultativement** un sous-cas.
  - héritage 
    - cas : le sous-cas est une variante du super-cas.
    - acteur : le sous-acteur bénéficie de tous les cas du super-acteur. Le sous-acteur possède plus de cas d'utilisation que le super-acteur.

Cas d'utilisation

- Démarche
  1. Définir les acteurs.
  2. Définir les cas d'utilisation nominaux.
  3. Les faire valider par l'équipe.

2- Diagramme de classes

- Vision statique du logiciel.
  - Intention : décrire la structure du logiciel.
    - Classes et relations (association, héritage).

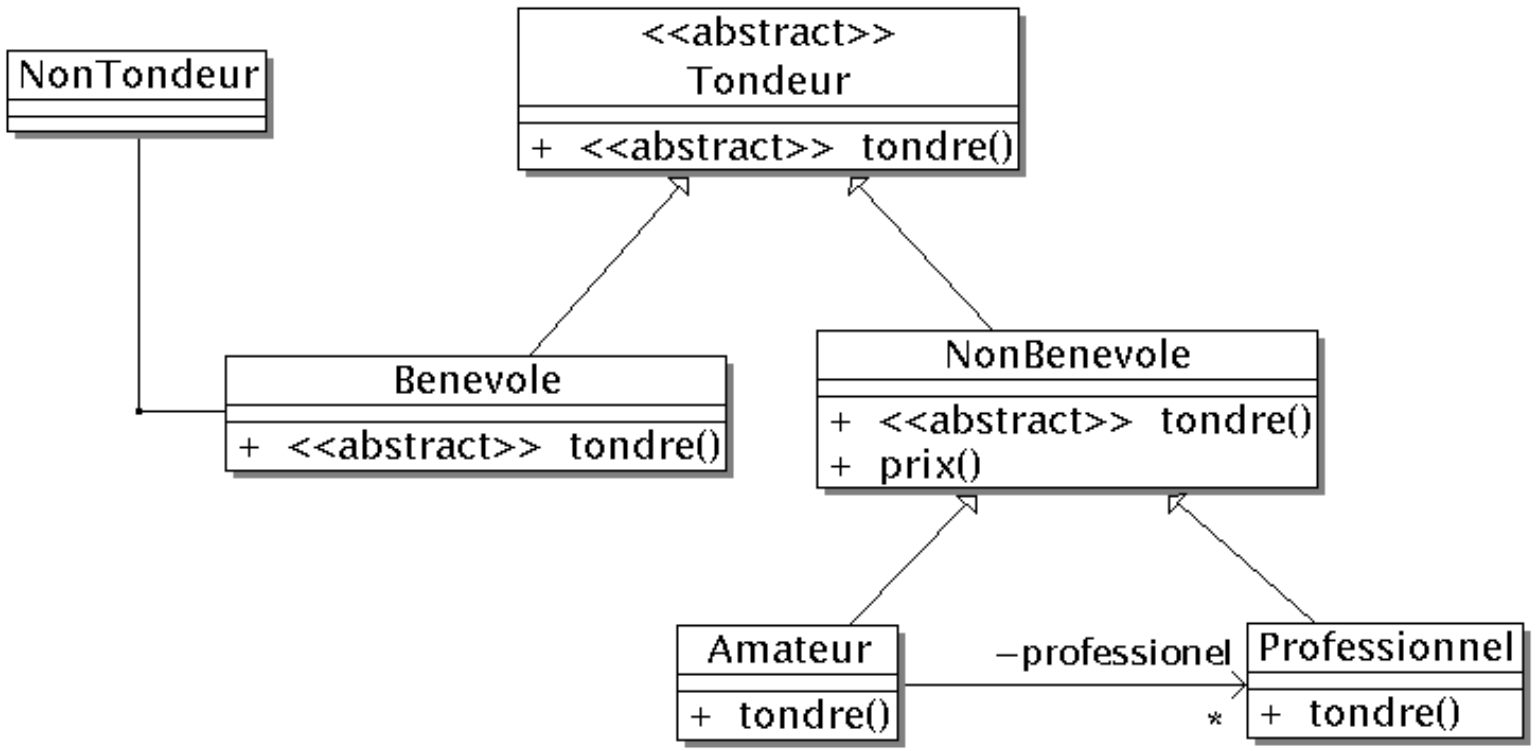


Diagramme de classes

- Attention à la lisibilité !

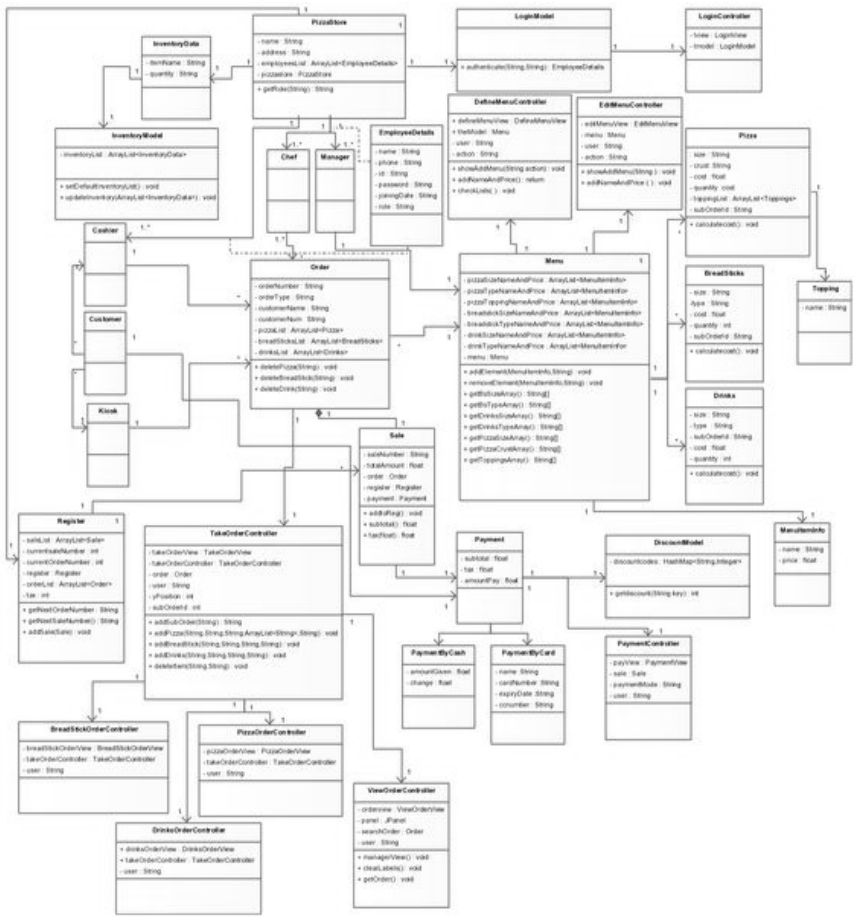
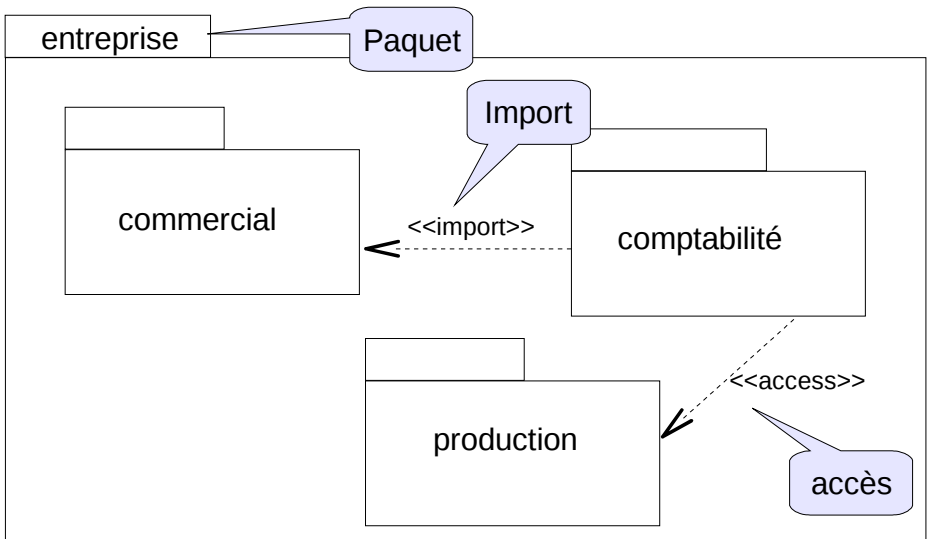


Diagramme de classes

- Principes à respecter :
  - Maximiser la notation statique.
  - **Ne pas mettre les attributs** (respecter le principe d'encapsulation).
  - Au pire, quand cela est nécessaire, ne faire apparaître que les méthodes publiques ; ie. les services.
  - Exception : diagrammes à usage interne entre développeurs.
    - On peut lever le masque sur les attributs privés et les méthodes privées et protégées puisque cela ne constitue pas une documentation.

3- Diagramme de paquets

- Vision structurelle du logiciel
  - Intention : décrire l'organisation des classes.
    - C'est par exemple le reflet de l'architecture.
- Paquet
  - Mécanisme de regroupement d'éléments qui ont un rapport.
    - classes et interfaces.
  - Imbrications possibles.
- Importance pour :
  - organisation du développement (p. ex. en équipes de dév.).
  - maintenance.

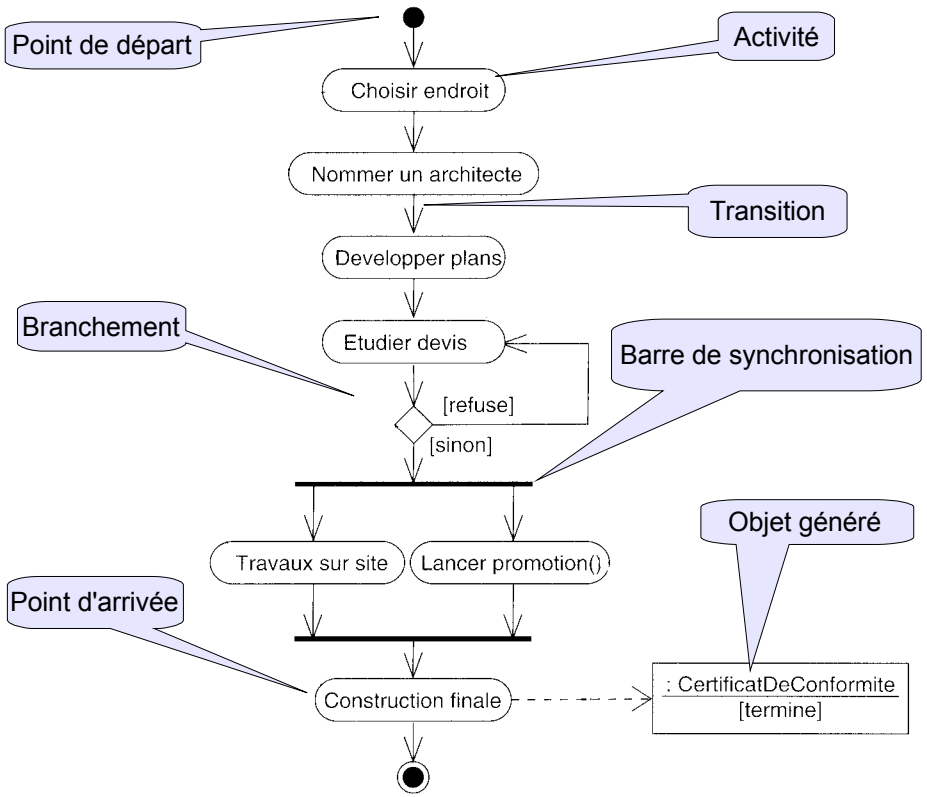


Un paquet définit un espace de nom

- Espace de noms
  - import : le contenu est ajouté à l'espace de nom.
    - Java : `import paquet; import static paquet;`
  - access : pas de rupture de l'espace de nom.
    - Java : `paquet.classe`

4- Diagramme d'activités

- Aspect dynamique
  - Intention : Décrire les activités liées à la réalisation d'un cas d'utilisation ou d'un algorithme.

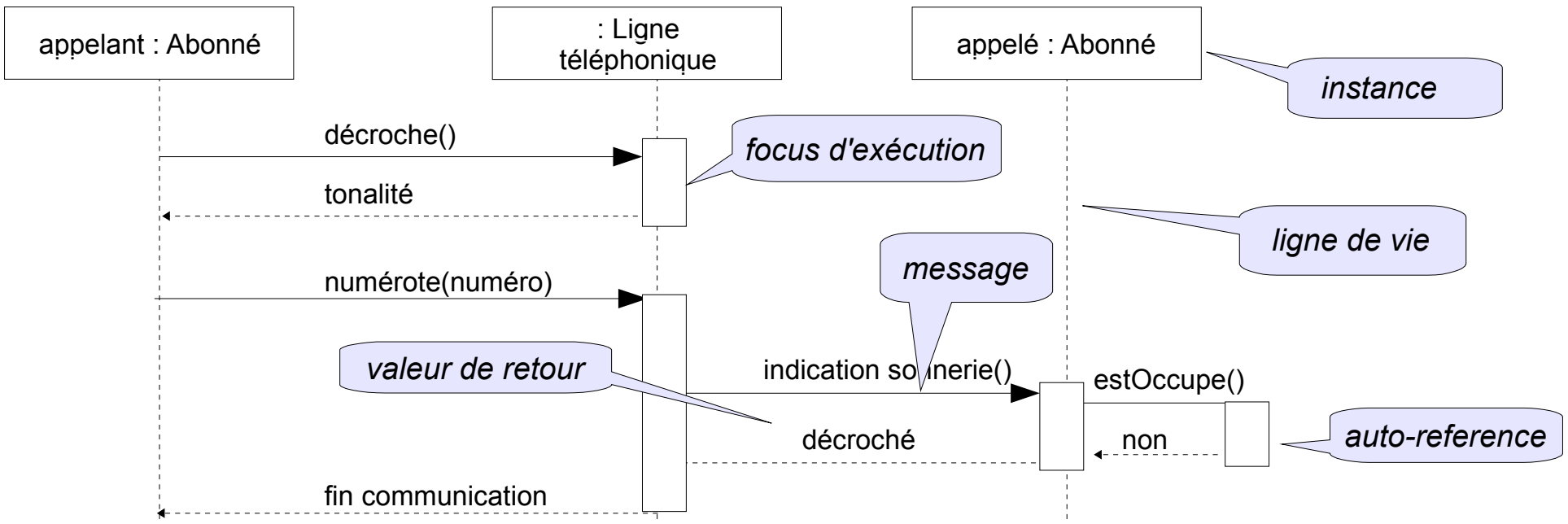


Syntaxe

- Activité
  - Étape particulière dans l'exécution.
    - l'appel d'une opération, l'envoi d'un signal, la création/la destruction d'un objet ou un simple calcul.
- Transition
  - Passage automatique d'une activité à une autre.
- Branchement conditionnel
  - Garde : Conditions exprimées par des expressions booléennes sur la transition d'un branchement.
  - Syntaxe : expression booléenne.
    - Exemples : [t = 15s] ; [code incorrect && essai > 3]
- Synchronisation
  - La barre de synchronisation permet d'ouvrir et de fermer des branches parallèles au sein d'un flot d'activités.

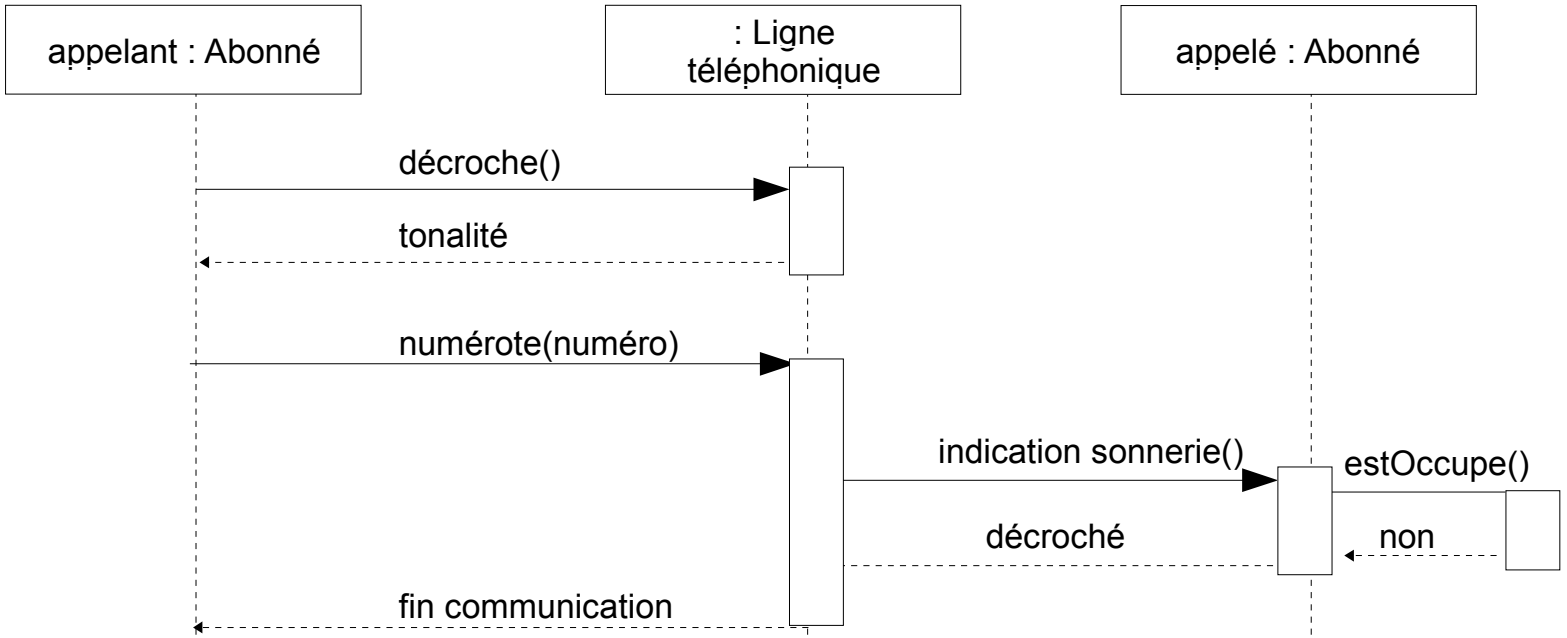
5- Diagramme de séquence

- Aspect dynamique.
  - Intention : décrire une séquence particulière d'interaction entre plusieurs **objets**, dans un seul contexte d'exécution du système.
  - Privilégie le point de vue temporel : lecture du haut vers le bas.



Utilisation pour le développement

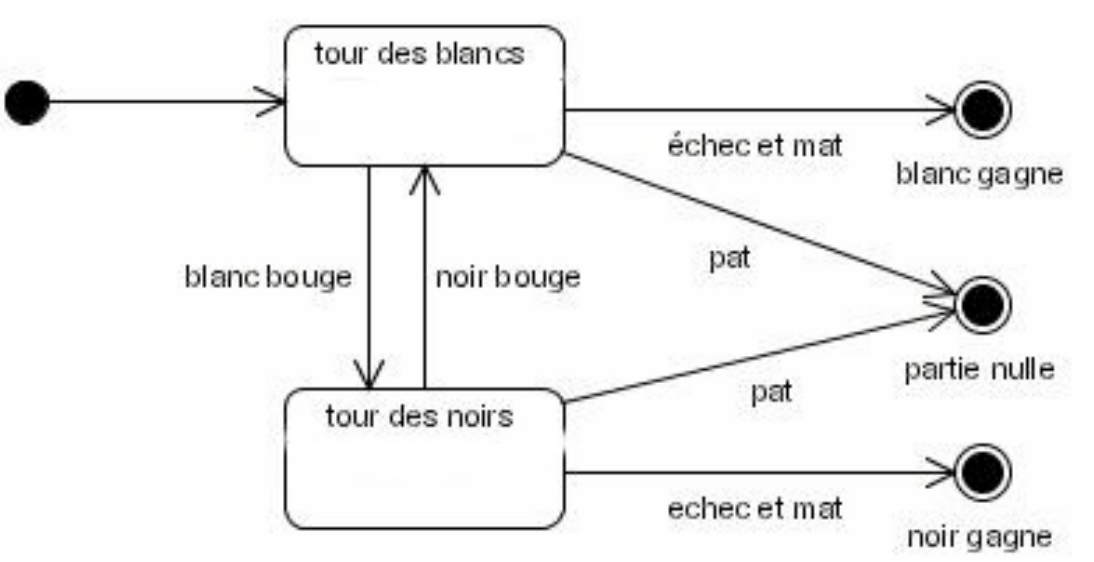
- Ajout des méthodes dans le diagramme de classes



- Définition des méthodes avec leurs paramètres et leur visibilité

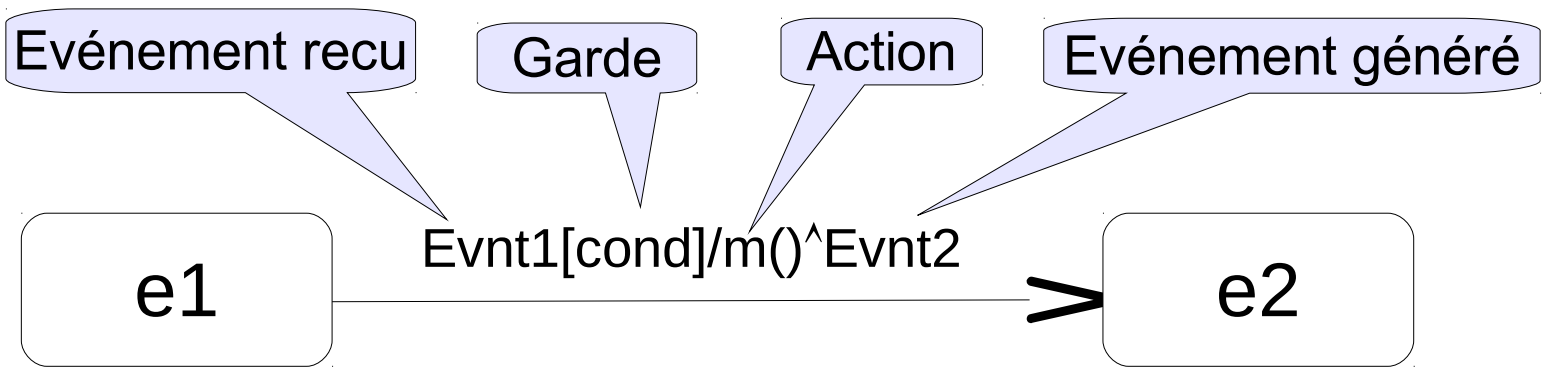
6- Diagramme d'états-transitions

- Aspect dynamique lié à une classe.
  - Intention : décrire les différents états que peuvent prendre une classe en fonction des opérations qui lui sont appliquées.
  - Chaque état d'une instance est caractérisé par un n-uplet unique de valeurs de propriétés.
  - Un diagramme états-transitions est propre à une classe donnée.



6- Diagramme d'états-transitions

- Syntaxe



Diagrammes plus spécifiques

- Diagramme d'Objets.
  - Collecte des données.
- Diagramme de communication.
  - Dual du diagramme de séquence (vision plus organisationnelle).
- Diagramme de Structure Composite.
  - Structure interne d'une classe et ses collaborations (Callback / délégués).
- Diagramme de Composants.
  - Structure des éléments logiciels (fichiers, exécutables, bibliothèques...).
- Diagramme de Déploiement.
  - Infrastructure physique (serveur, routeur, smart phone,...).
- Diagramme de Timing.
  - Synchronisation de tâches.

Utilisation des diagrammes

- Méthodes de génie logiciel classiques
  - Documentation de l'analyse des besoins.
  - Documentation de conception.
- Méthodes de génie logiciel agiles
  - Essentiellement un support de communication entre développeurs.
  - Diagramme de classes avec l'architecture au tableau à la vue de tous.

Atelier de génie logiciel

- Logiciels permettant de produire des modélisations UML.
  - Certains sont capables de générer le code Java/C++ correspondant à une modélisation UML.
  - Certains sont capables de relire du code Java/C++ pour en produire la représentation UML correspondante.
- Logiciels gratuits (voire open-source)
  - DoUML
  - Argouml
  - Umbrello
  - Yuml (version en ligne)

Que retenir de ce chapitre ?

- UML est un langage de modélisation diagrammatique.
  - Très puissant.
  - Couvre tout le cycle de vie.
  - Incontournable pour la modélisation de logiciels.
- Il n’y a que 6 diagrammes principaux.
  - Les autres diagrammes sont plus spécialisés et utilisés que dans certains cas précis.
  - Toute modélisation doit commencer par le diagramme des cas d'utilisation.
- Il y a peu de syntaxe mais elle doit être respectée.