

# JAVA / UML

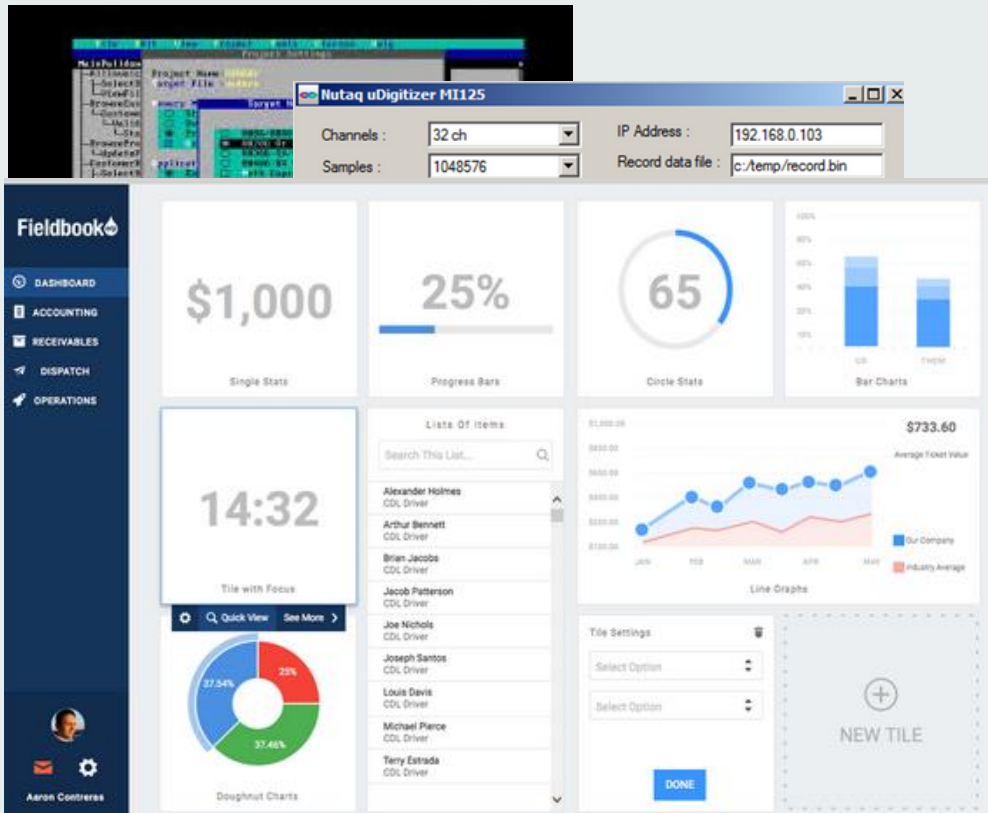
# LES APPLICATIONS

# UNE APPLICATION

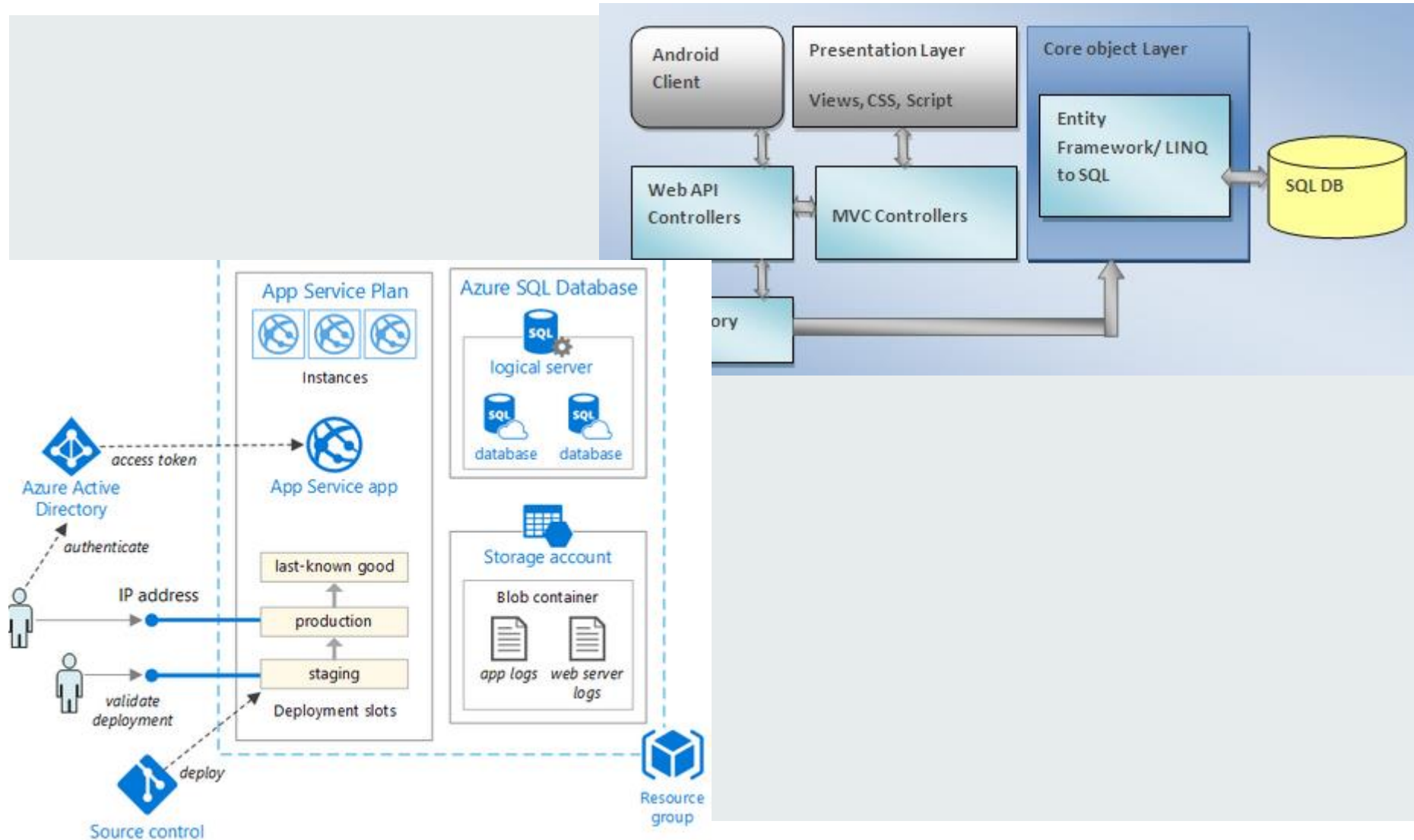
- Doit répondre à un besoin d'un utilisateur
- Permet d'automatiser des traitements : calcul, affichage de données, enregistrement de données...



# EVOLUTION DES APPLICATIONS



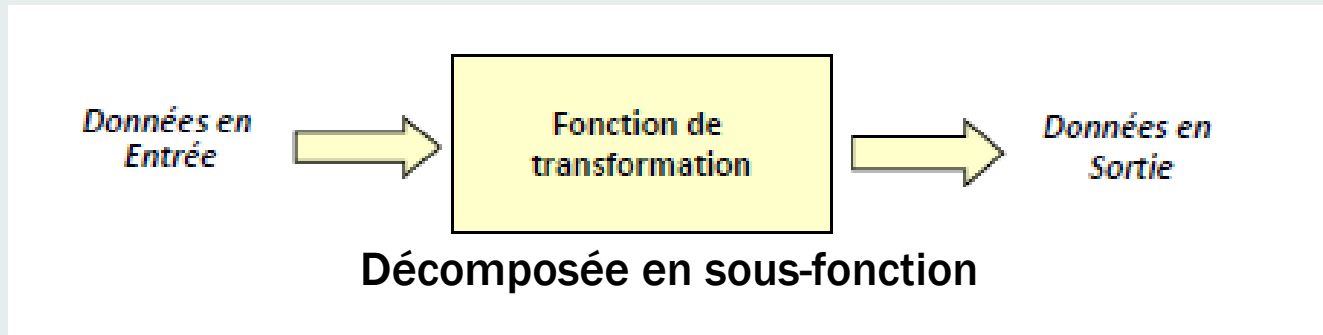
# COMPLEXITÉ CROISSANTE



# L'ARCHITECTURE OBJET

# DÉVELOPPEMENT PROCÉDURAL

- Démarche descendante



- MCD – structuration des données → stockage en BD
- MCT – définition des traitements → programmes

# LIMITES DU PROCÉDURAL

- Séparation des données et des traitements
- Architecture basée sur les fonctions et non sur les données
- Difficile réutilisabilité d'une application à une autre
- Evolutivité limitée



# L'ARCHITECTURE OBJET

- S'intéresse aux caractéristiques des éléments gérés par l'application
- Combine les données et les traitements
  - Un objet contient des informations
  - Un objet est à la base de la réalisation de traitements

# L'ARCHITECTURE OBJET

- L'application gère des objets structurés
- Les objets peuvent être liés à d'autres objets et interagir entre eux
- La réalisation d'une action est directement liée aux données de l'objet

# OBJECTIFS

## ■ Réutilisabilité

- Une structure d'objet peut être reprise en ajoutant des informations spécifiques

## ■ Evolutivité

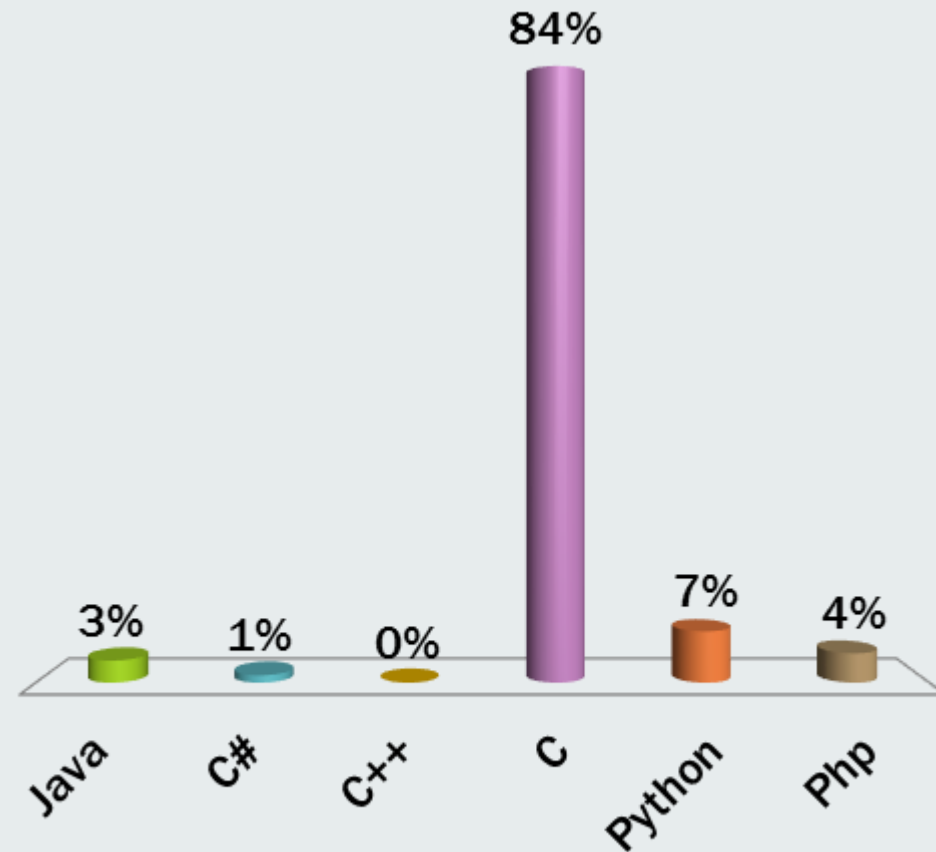
- Ajout de fonctionnalités par la création de nouveaux objets
- Ajout de services liés à un objet, sans impact sur le reste de l'application
- ...

# OBJECTIFS

- **Qualité, sécurité, robustesse**
  - Chaque objet est testé et validé (tests unitaires)
- **Maintenance facilitée**
  - Structuration explicite

# LEQUEL DE CES LANGAGES N'EST PAS OBJET ?

- A. Java
- B. C#
- C. C++
- D. C
- E. Python
- F. Php



# LA CONCEPTION OBJET

# L'APPROCHE OBJET

- Réflexion centrée sur l'application
- Structuration pour optimiser le code
- Architecture favorisant la réutilisabilité

# ANALYSE / CONCEPTION

- Identifier les besoins
- Formalisés dans un cahier des charges

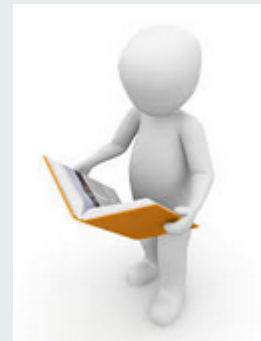
Je veux un logiciel ergonomique



ARGH



Je mets  
un bouton  
multicolore



Je mets des  
boutons  
standards



Je mets  
de gros boutons  
en couleur



# ANALYSE/CONCEPTION

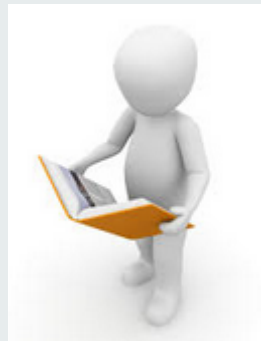
- Basée sur des modèles standardisés à base de schémas



**Maquette  
Écrans + charte  
graphique**



**Un bouton  
ne suffira  
pas**



**Il faut des boutons  
spécifiques**

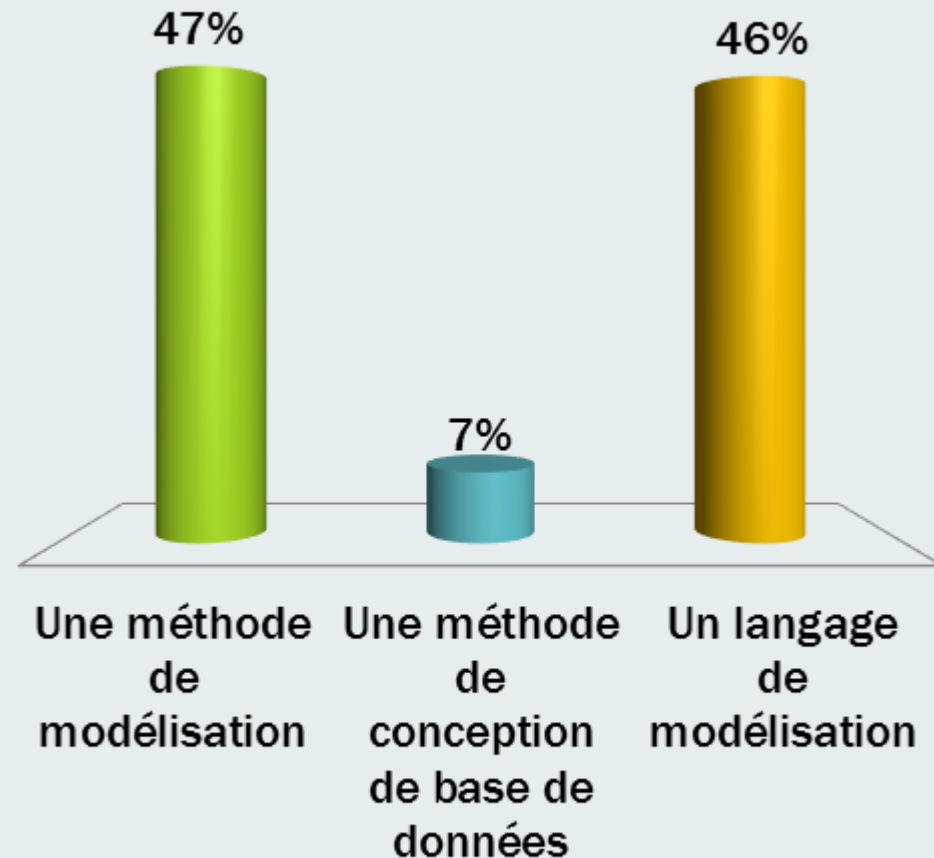


**Trop de couleurs  
nuisent à l'ergonomie**

UML

# QU'EST-CE QU'UML ?

- A. Une méthode de modélisation
- B. Une méthode de conception de base de données
- C. Un langage de modélisation



# UML

- Langage graphique de conception de logiciel
- Défini par l'Object Management Group (OMG) :  
<http://www.omg.org/>
  - Association fondée aux Etats-Unis
  - Objectif : mettre en place des standards pour les applications orientées objet

# HISTORIQUE

- Méthode construite sur la base de
  - OMT (centre de développement de General Electrics)
  - OOD (définie pour le département américain de la défense)
  - OOSE (centre de développement d'Ericsson)
- Novembre 1997 : version 1.1 d'UML
- Actuellement version 2.5

# UML : TREIZE DIAGRAMMES

- **Diagrammes comportementaux ou diagrammes dynamiques (*UML Behavior*)**
  - diagramme de cas d'utilisation (*Use case diagram*)
  - diagramme d'activités (*Activity diagram*)
  - diagramme d'états-transitions (*State machine diagram*)

# UML : TREIZE DIAGRAMMES

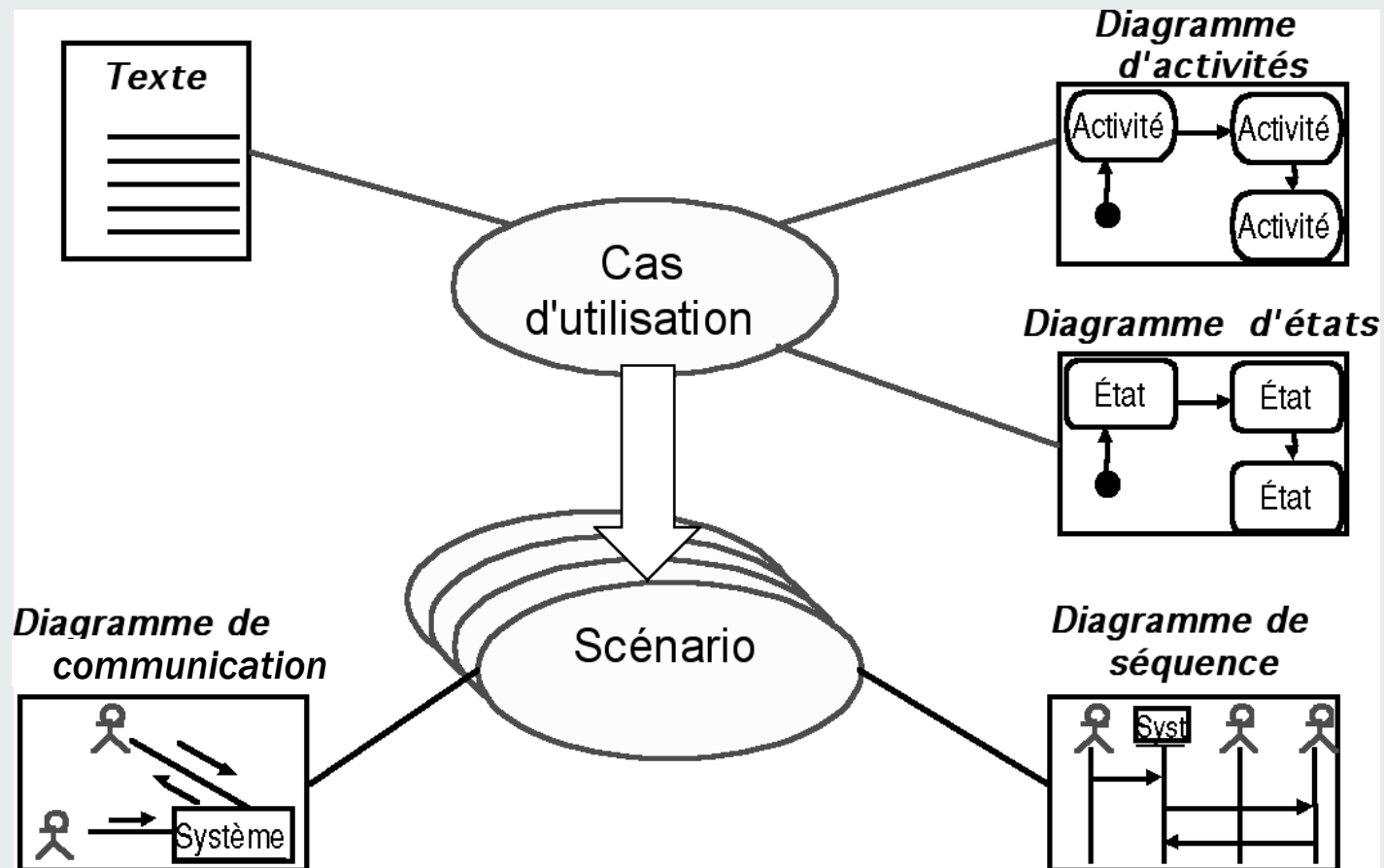
- Diagrammes d'interaction (*Interaction diagram*)
  - diagramme de séquence (*Sequence diagram*)
  - diagramme de communication (*Communication diagram*)
  - diagramme global d'interaction (*Interaction overview diagram*)
  - diagramme de temps (*Timing diagram*)

# UML : TREIZE DIAGRAMMES

- Diagrammes structurels ou diagrammes statiques (*UML Structure*)
  - diagramme de classes (*Class diagram*)
  - diagramme d'objets (*Object diagram*)
  - diagramme de composants (*Component diagram*)
  - diagramme de déploiement (*Deployment diagram*)
  - diagramme de paquetages (*Package diagram*)
  - diagramme de structures composites (*Composite structure diagram*)



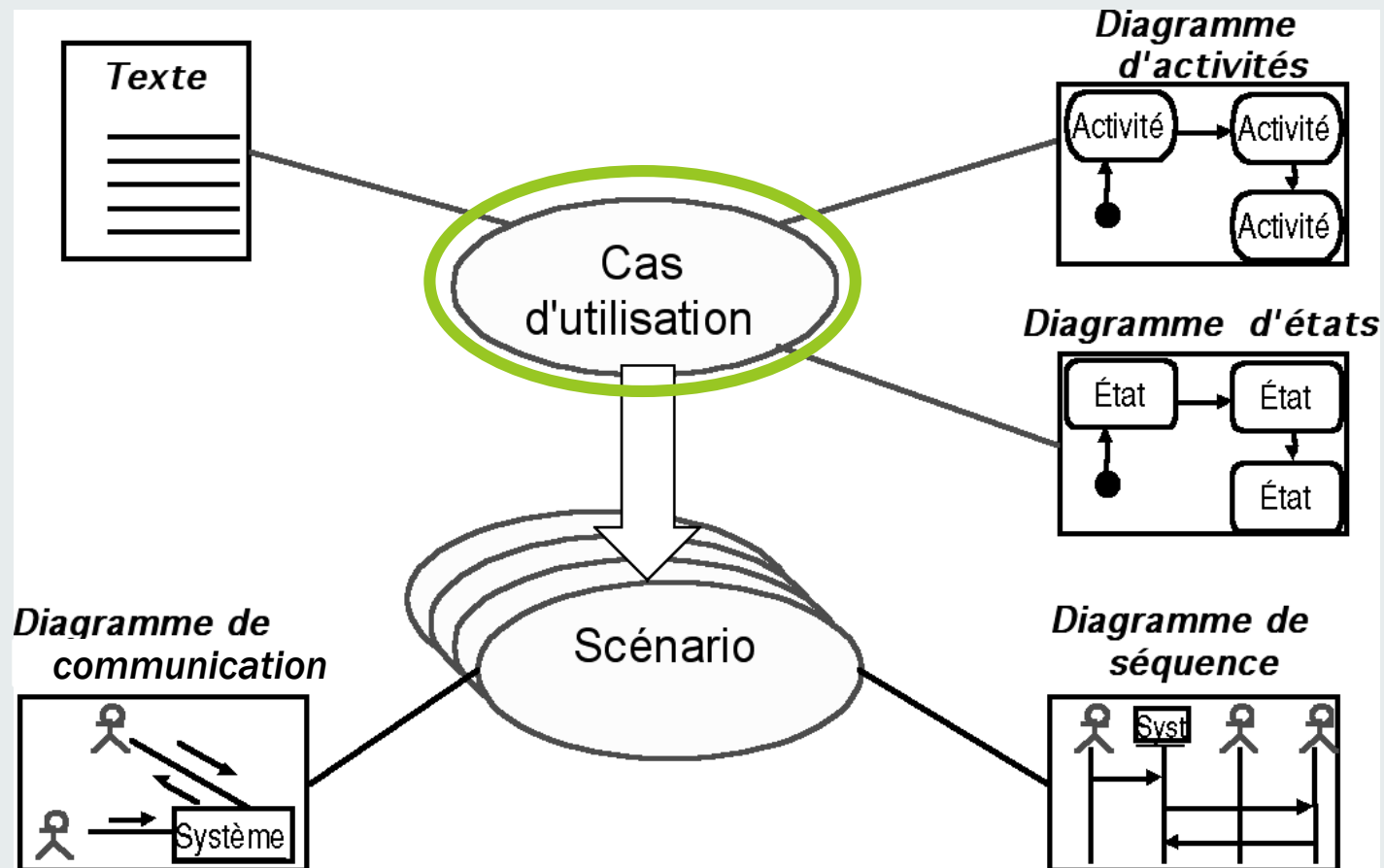
# RELATIONS ENTRE DIAGRAMMES



# CONCEPTION D'UNE APPLICATION

Diagrammes  
comportementaux

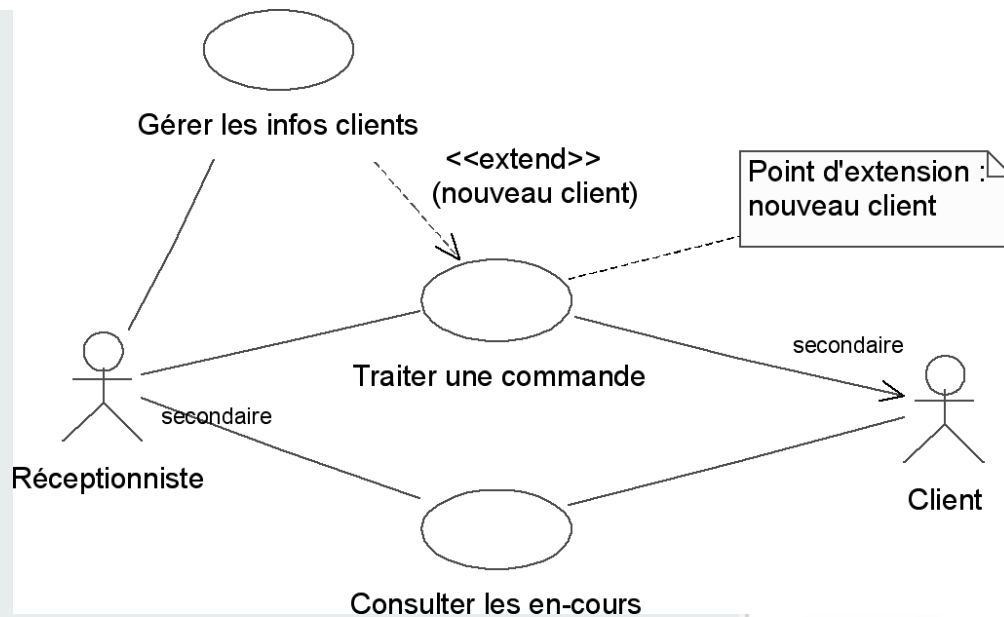
# RELATIONS ENTRE DIAGRAMMES



# DIAGRAMME DES CAS D'UTILISATION

- Permet de modéliser
  - Les acteurs en lien avec l'application
  - Les fonctionnalités de l'application

# DIAGRAMME DES CAS D'UTILISATION



## Mes clients



Créer



Modifier



Imprimer



Effacer



Appeler



Localiser

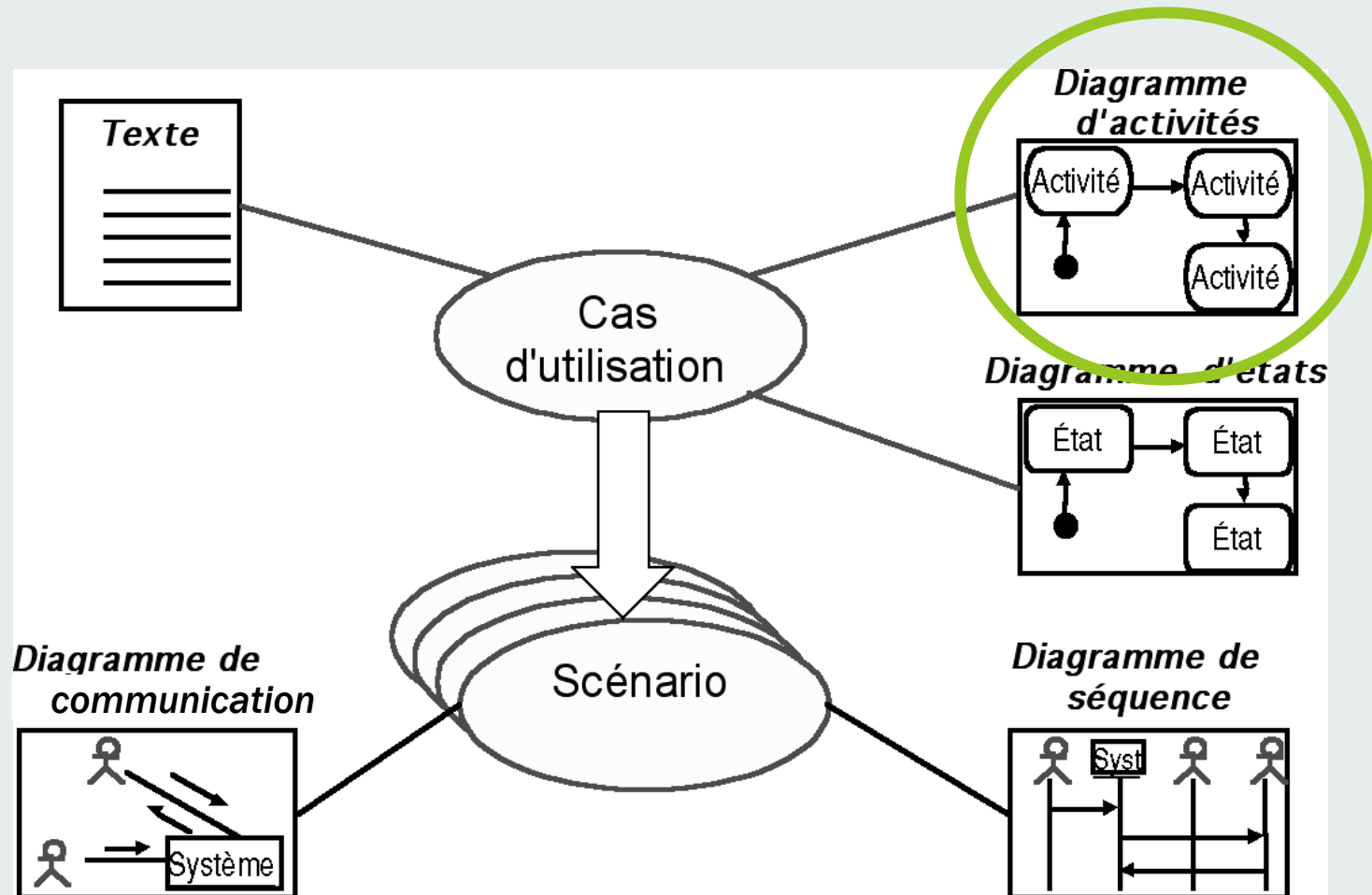


Voir planning



Voir dossier

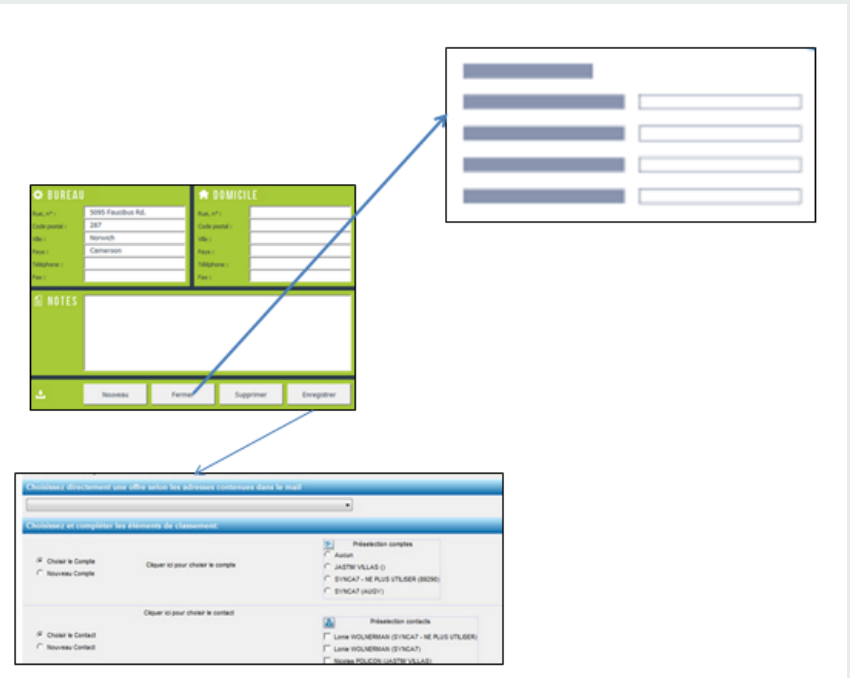
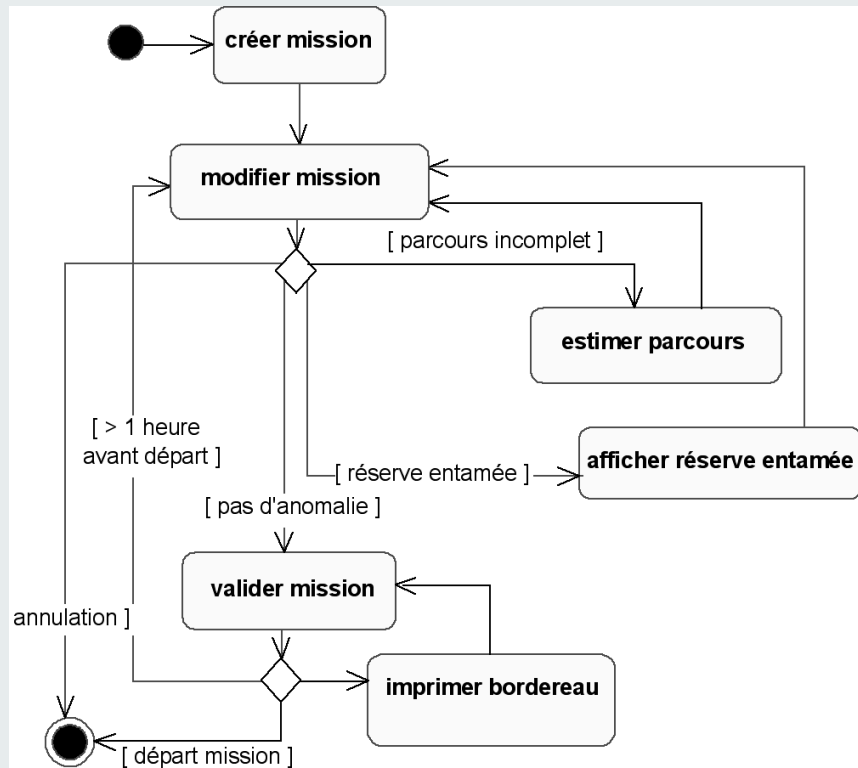
# RELATIONS ENTRE DIAGRAMMES



# DIAGRAMME D'ACTIVITÉ

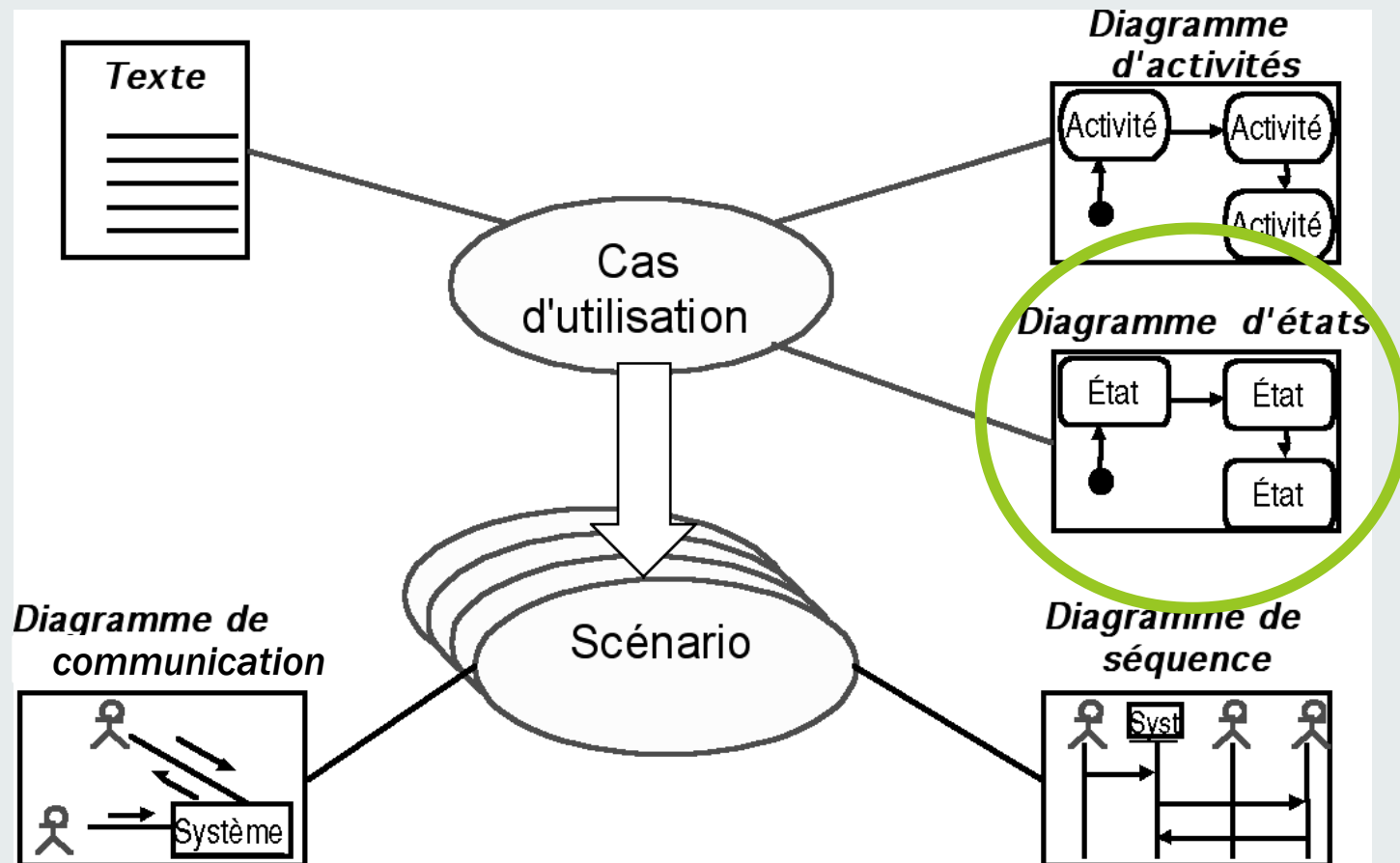
- Permet de modéliser
  - Le scénario d'un cas d'utilisation
  - Les exceptions

# DIAGRAMME D'ACTIVITÉ





# RELATIONS ENTRE DIAGRAMMES

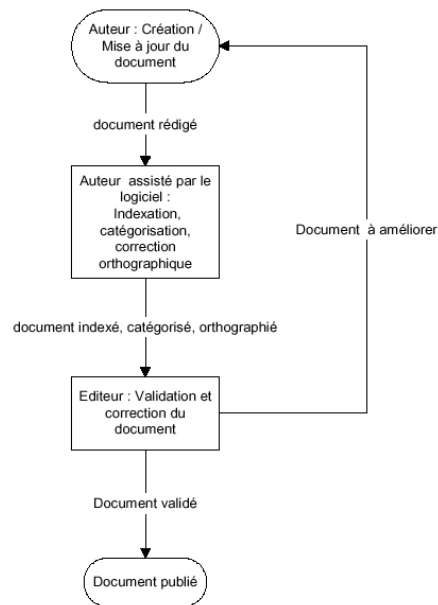


# DIAGRAMME D'ÉTATS

- Permet de modéliser
  - Le comportement du système
  - L'enchaînement des états des objets de l'application

# DIAGRAMME D'ÉTATS

Figure 2 : exemple de diagramme état-transition de spécification d'une chaîne d'édition



Exemple de formulaire de saisie de données :

Le formulaire est divisé en plusieurs sections :

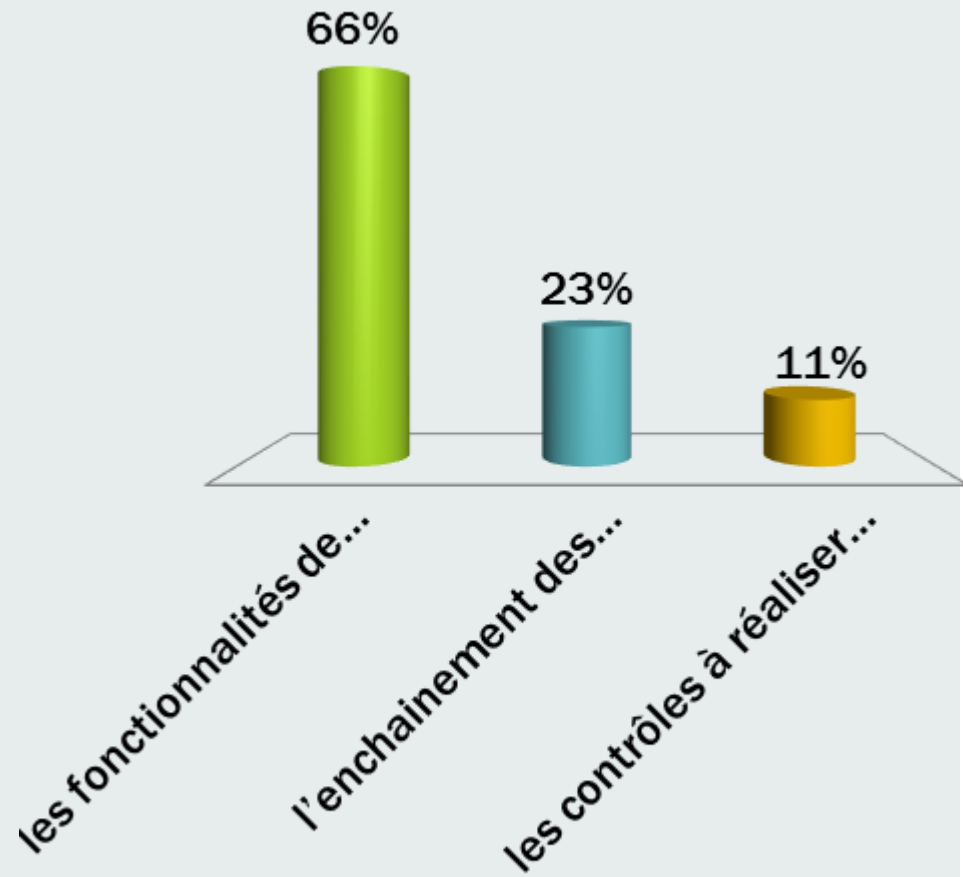
- BUREAU** : Saisie des coordonnées professionnelles (Nom, Adresse, Téléphone, Fax, Courriel).
- DOMICILE** : Saisie des coordonnées personnelles (Nom, Adresse, Téléphone, Fax, Courriel).
- NOTES** : Zone de saisie libre pour les commentaires.
- Actions** : Boutons pour sauvegarder, annuler, supprimer ou enregistrer les données.

En bas du formulaire, il y a des sections pour la sélection de contacts et de comptes :

- Choisissez et complétez une offre selon les adresses contenues dans le mail** : Sélection d'une offre à partir d'une liste déroulante.
- Choisissez et complétez les éléments du classement** : Sélection de contacts et de comptes à partir de listes déroulantes.

# QUE REPRÉSENTE LE DIAGRAMME DES CAS D'UTILISATION ?

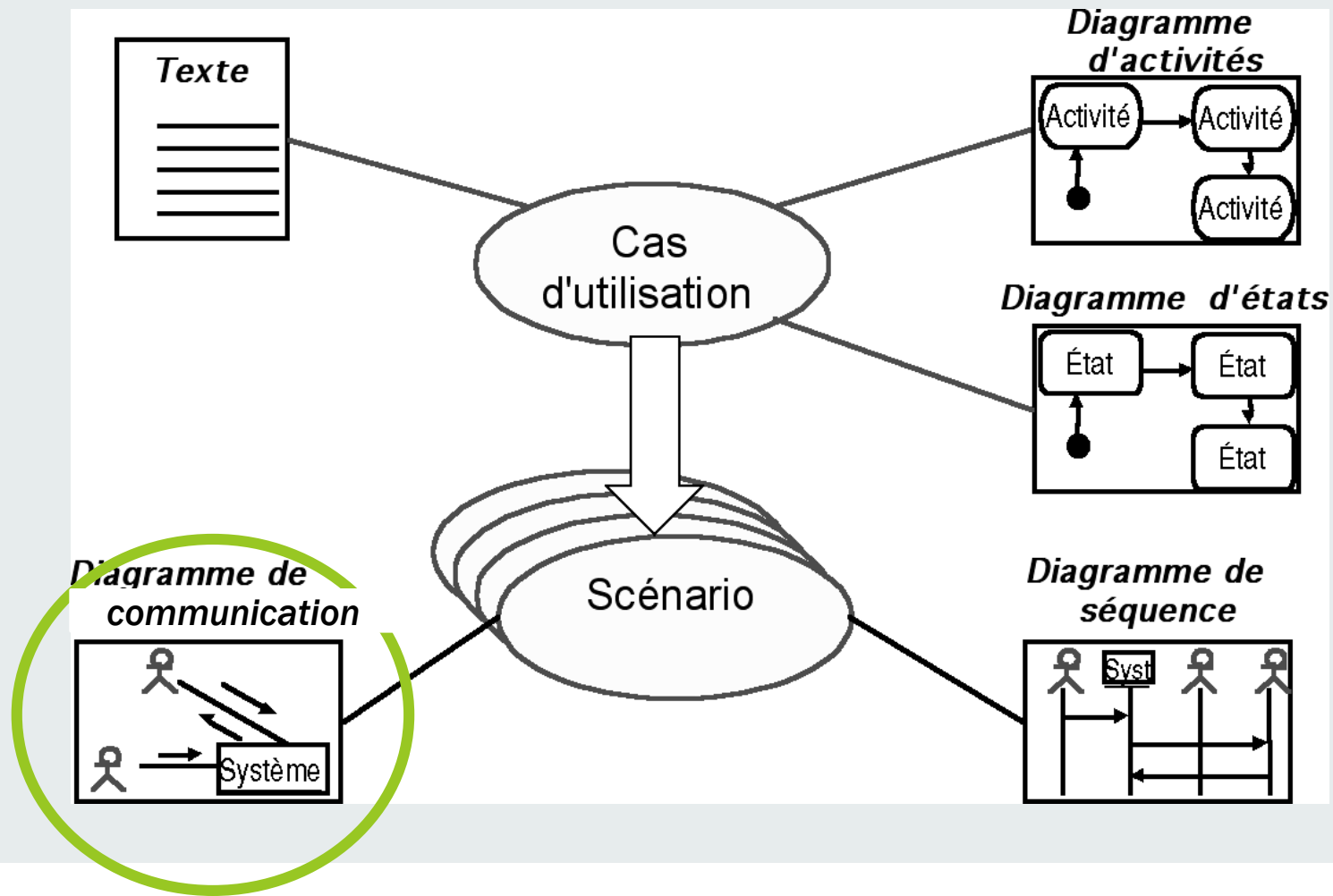
- A. les fonctionnalités de l'application
- B. l'enchaînement des activités de l'application
- C. les contrôles à réaliser dans l'application



# CONCEPTION D'UNE APPLICATION

Diagrammes d'interaction

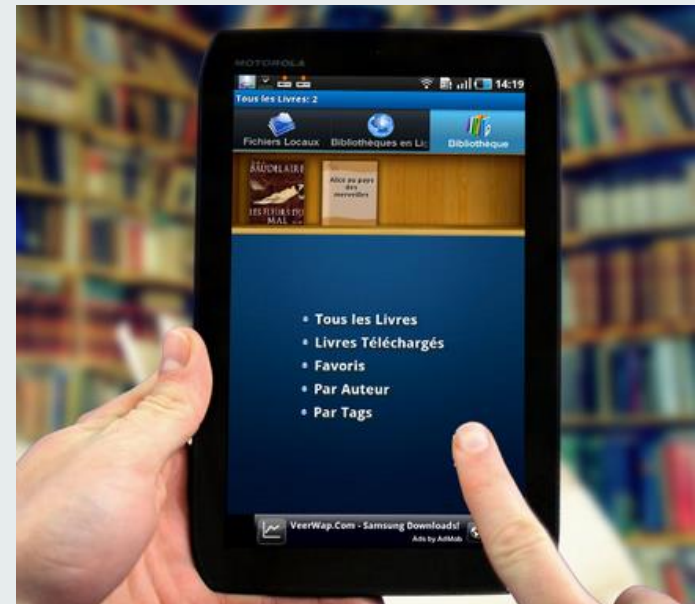
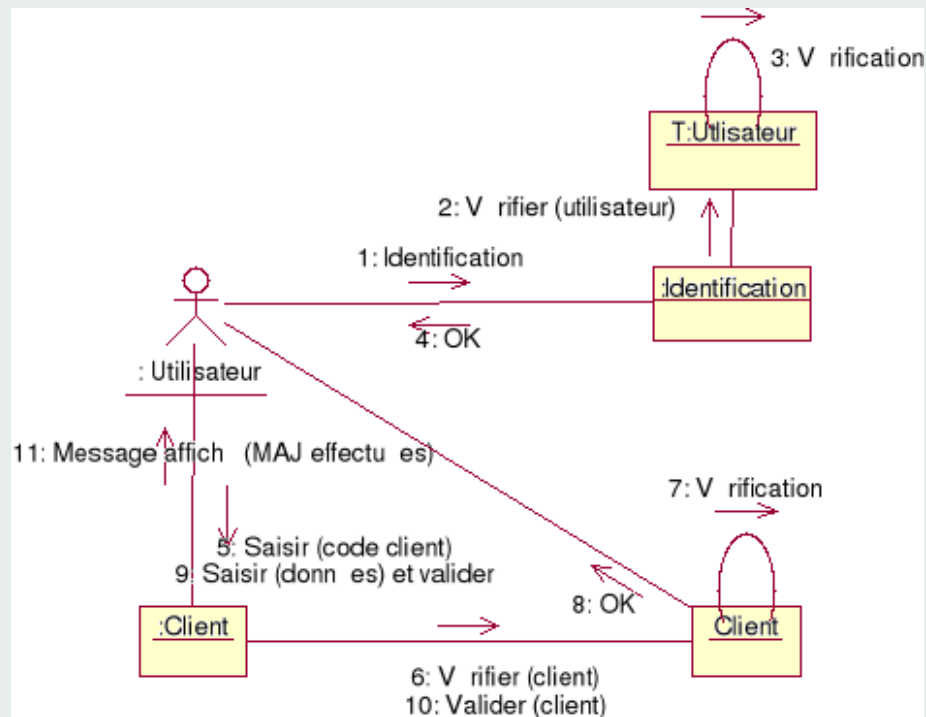
# RELATIONS ENTRE DIAGRAMMES



# DIAGRAMME DE COMMUNICATION

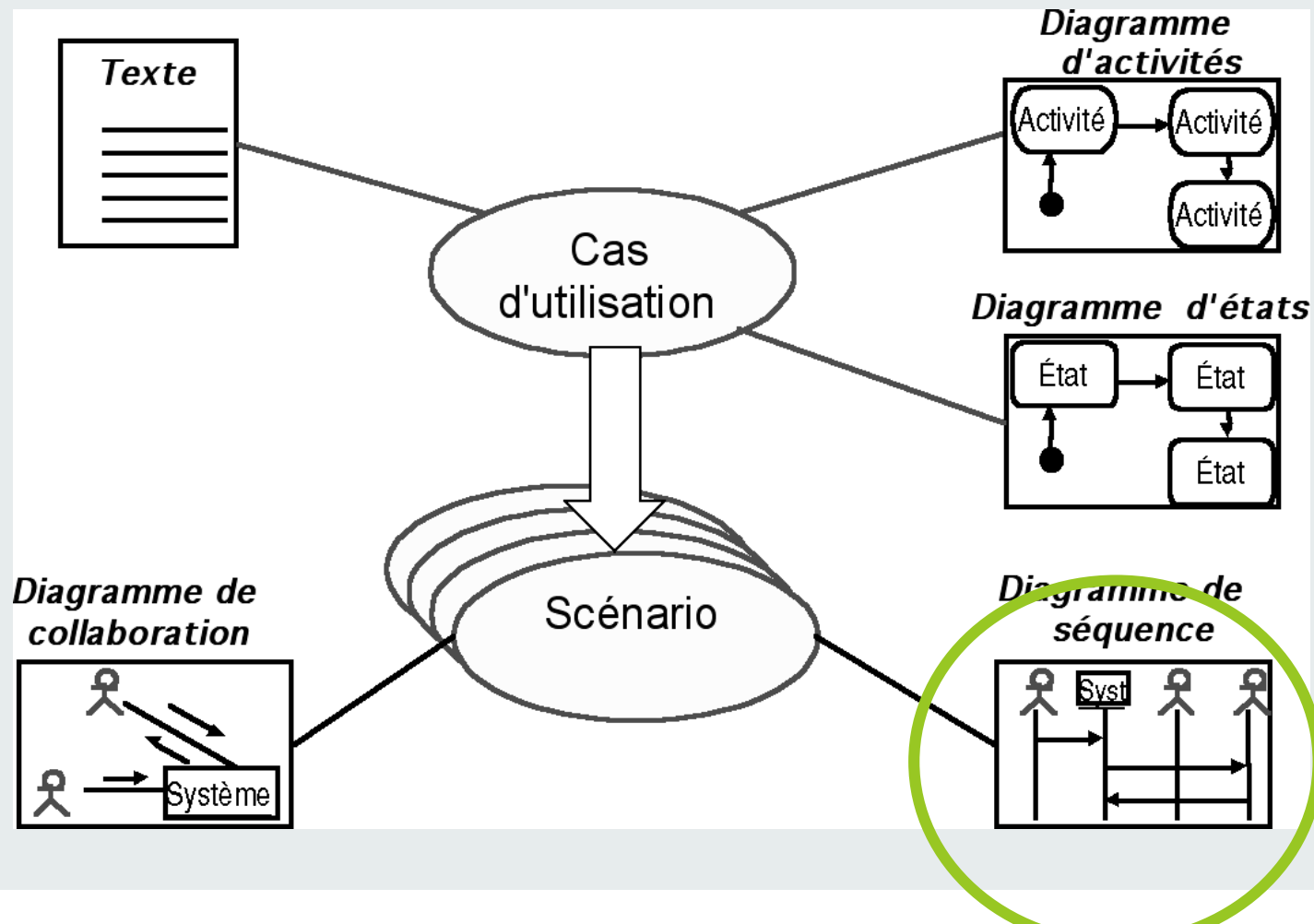
- Permet de modéliser
  - Les interactions entre le système et les acteurs

# DIAGRAMME DE COMMUNICATION





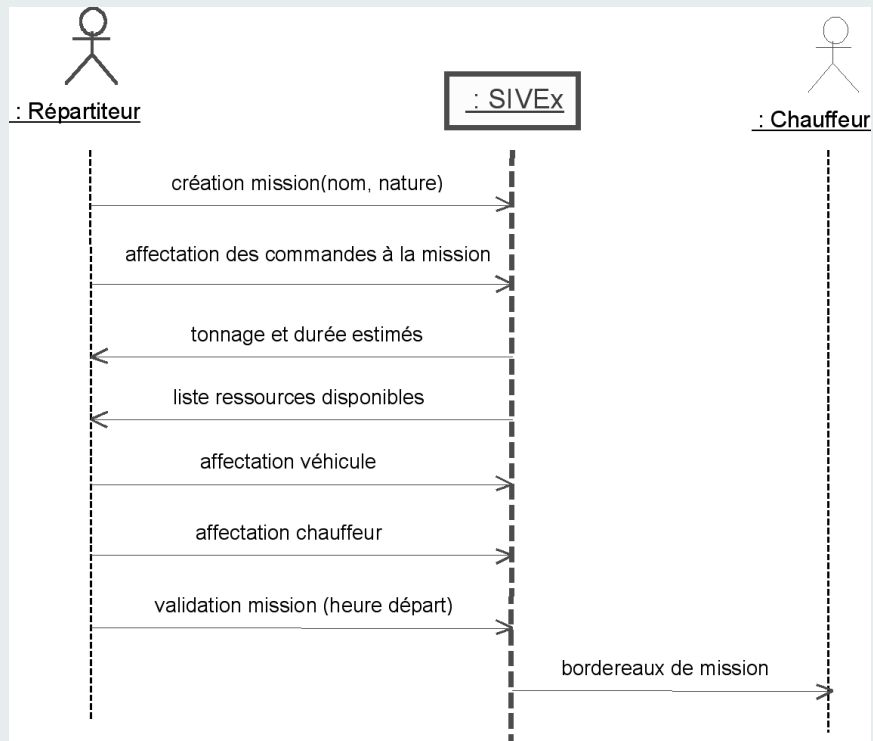
# RELATIONS ENTRE DIAGRAMMES



# DIAGRAMME DE SÉQUENCE

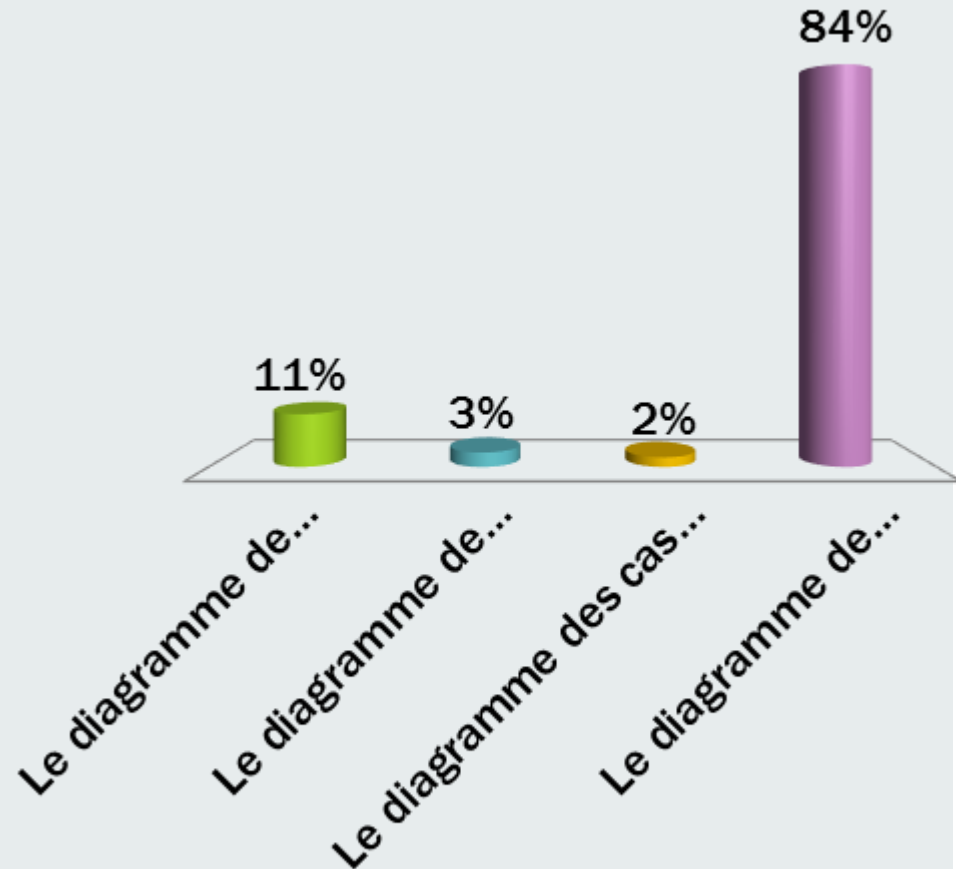
- Permet de modéliser
  - Le déroulement des traitements
  - Les interactions entre le système et les acteurs

# DIAGRAMME DE SÉQUENCE



# QUEL DIAGRAMME REPRÉSENTE L'ENCHAINEMENT DES TRAITEMENTS ?

- A. Le diagramme de traitement
- B. Le diagramme de collaboration
- C. Le diagramme des cas d'utilisation
- D. Le diagramme de séquence



# CONCEPTION D'UNE APPLICATION

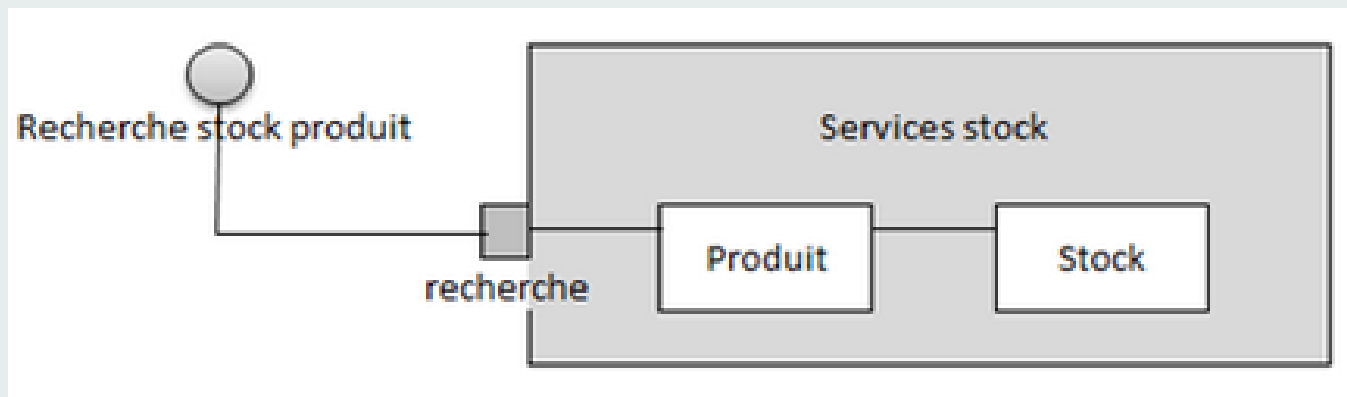
Diagrammes statiques

# AU CŒUR DE L'APPLICATION

- La conception d'une application permet de définir
  - Les packages
  - Les classes
  - L'organisation des classes
  - L'architecture de l'application...

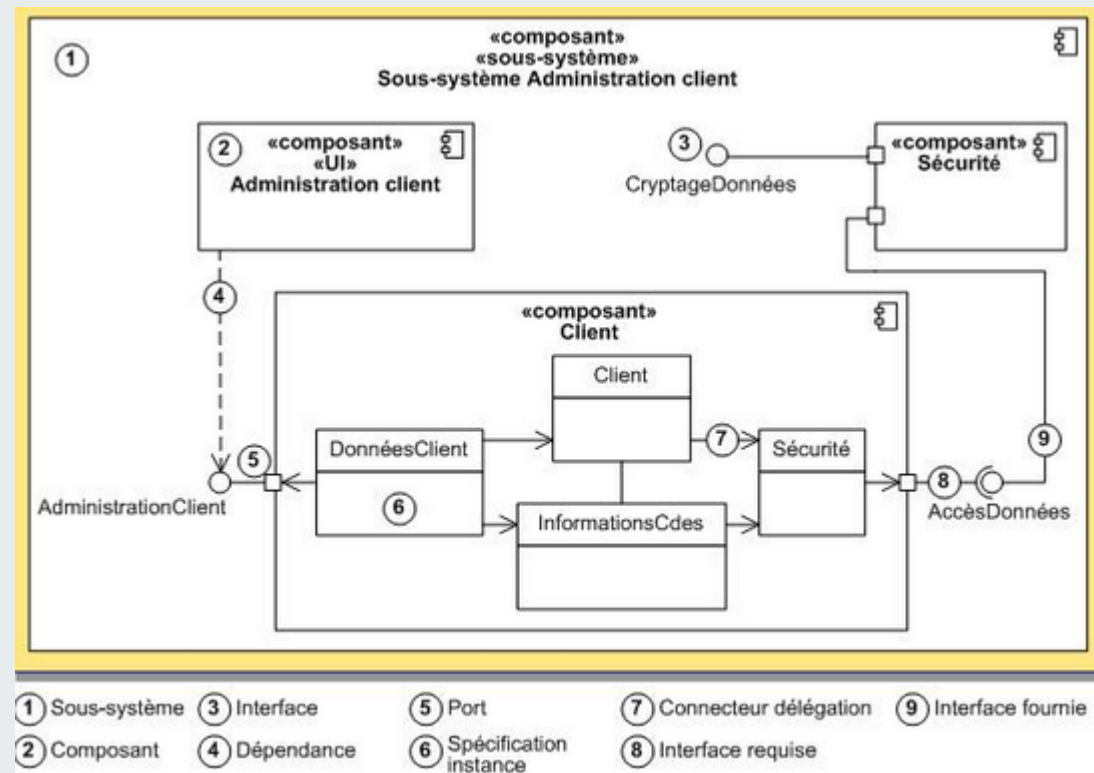
# DIAGRAMME DE STRUCTURE COMPOSITE

- Représente l'organisation interne de la classe
  - Interactions internes
  - Collaborations possibles



# DIAGRAMME DE COMPOSANTS

- Modélise les interactions entre les composants d'une application

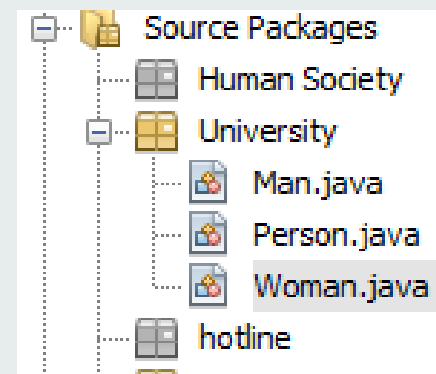
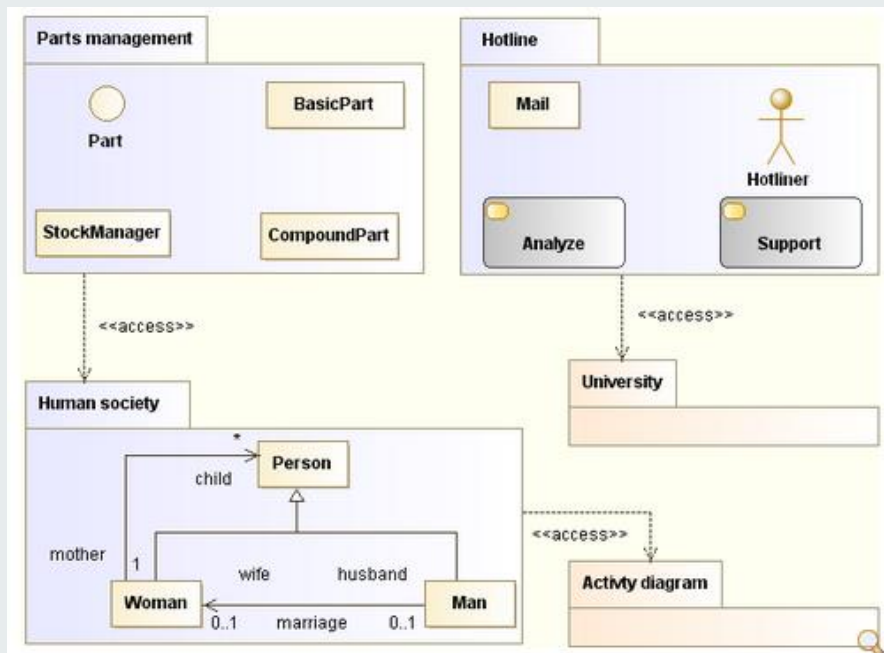




# DIAGRAMME DE PAQUETAGES

- Permet d'organiser l'architecture d'une application
  - Différents paquets
  - Classes de chaque paquet
  - Interaction entre paquets

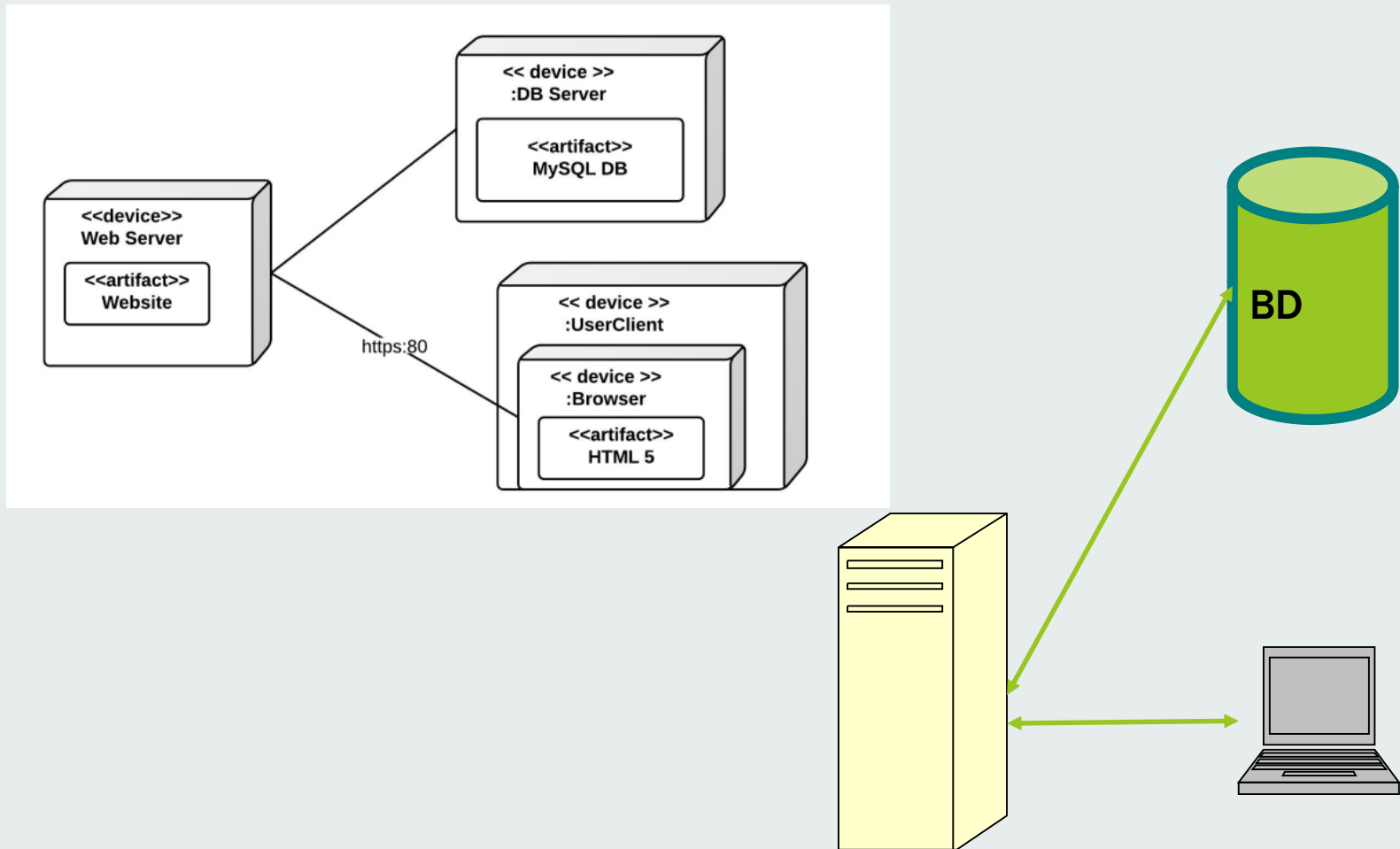
# DIAGRAMME DE PAQUETAGE



# DIAGRAMME DE DÉPLOIEMENT

- Décrit le déploiement physique des composants logiciels de l'application sur les supports matériels (serveurs, postes clients...)

# DIAGRAMME DE DÉPLOIEMENT



# DIAGRAMME DE CLASSE

# DIAGRAMME DE CLASSE

- Modélisation d'une classe
- Relations entre classes
  - Dépendance
  - Agrégation/Composition
  - Héritage : généralisation, spécialisation

# CLASSE / OBJET

## Classe

- Structure des objets de l'application
  - Données
  - Actions
- Ne s'exécute pas

## Objet

- Instance d'une classe
- Possède la structure d'une classe avec des informations spécifiques

# CLASSE / OBJET





# EXEMPLE D'ATTRIBUTS / PROPRIÉTÉS

Structure définie par la classe



Landship



Valeurs  
spécifiques  
de l'objet



Circuit Special



# EXEMPLE D'ACTIONS : MÉTHODES



- Accélérer
- Freiner
- Dérapager
- Tourner à droite
- Tourner à gauche
- Sauter
- Utiliser une option
- ...

# ANALYSE

- Dans l'application de gestion des JO, on enregistre les informations suivantes :
  - L'épreuve de bosses féminine a eu lieu le 11/02/2018
  - Les médaillées sont :
    - Perrine Laffont
    - Justine Dufour-Lapointe
    - Ioulia Galycheva

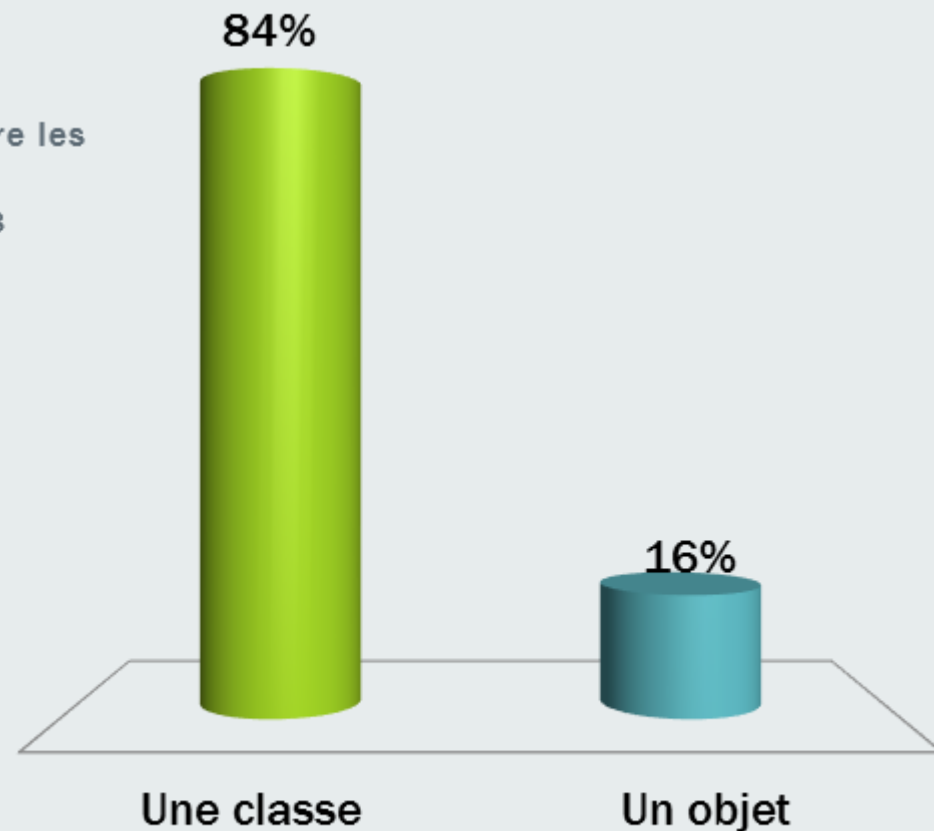
# QUEL SERA LE RÔLE D'ÉPREUVE DANS L'APPLICATION ?

A. Une classe

B. Un objet

■ Dans l'application de gestion des JO, on enregistre les informations suivantes :

- L'épreuve de bosses féminine a eu lieu le 11/02/2018
- Les médaillées sont :
  - Perrine Laffont
  - Justine Dufour-Lapointe
  - Ioulia Galycheva



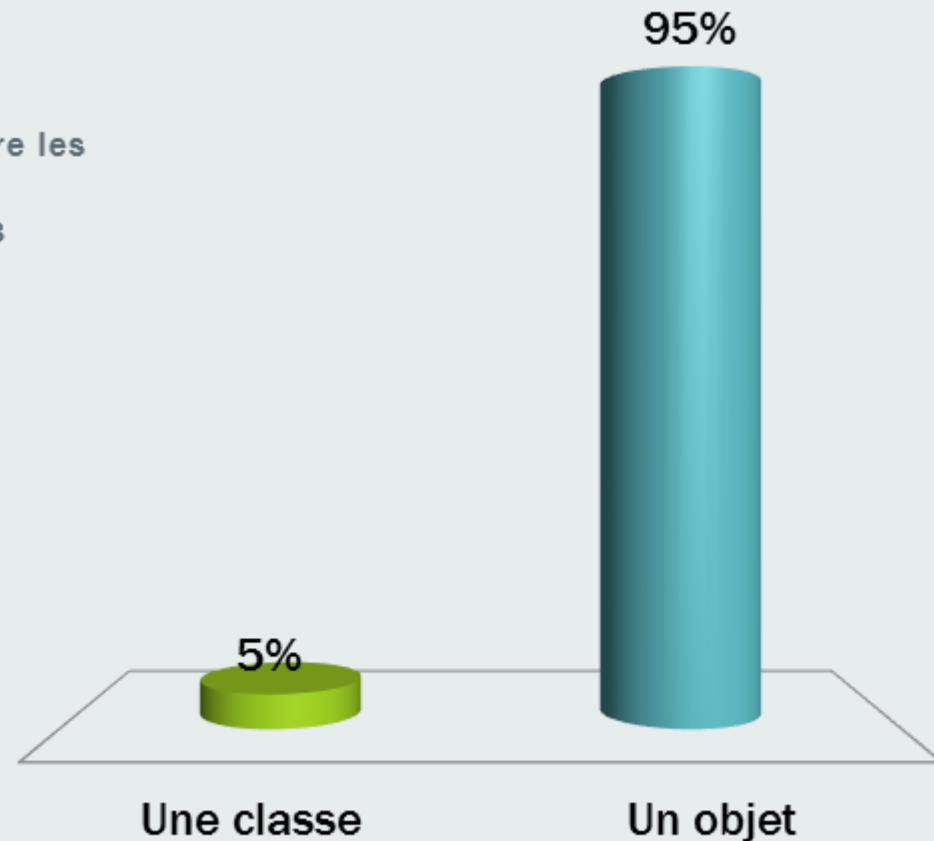
# QUEL SERA LE RÔLE DE PERRINE LAFFONT DANS L'APPLICATION ?

A. Une classe

B. Un objet

■ Dans l'application de gestion des JO, on enregistre les informations suivantes :

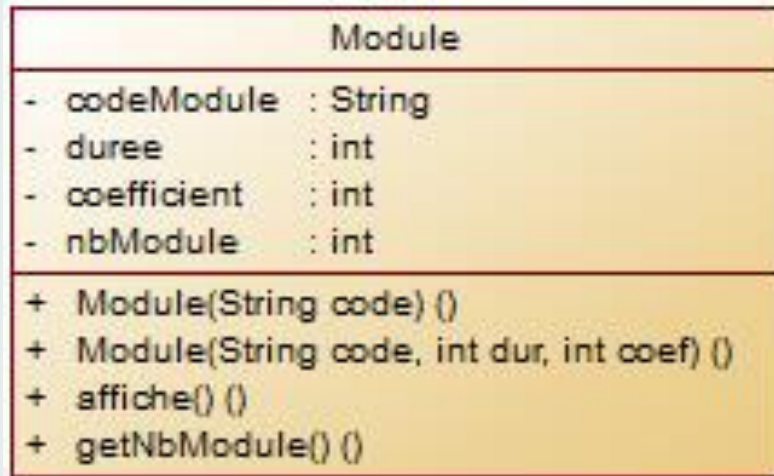
- L'épreuve de bosses féminine a eu lieu le 11/02/2018
- Les médaillées sont :
  - Perrine Laffont
  - Justine Dufour-Lapointe
  - Ioulia Galycheva



# STRUCTURES DES OBJETS : LES CLASSES

# MODÉLISER UNE CLASSE

## UML



## JAVA

■ Nom de la classe

■ Attributs

■ Constructeur

■ Méthodes

■ Portée

■ - : private

■ + : public

■ # : protected

# PORTÉE DES ÉLÉMENTS DE LA CLASSE

- **private** : Utilisé pour encapsuler les attributs et les méthodes internes de la classe
  - Les éléments de cette section sont accessibles uniquement par les méthodes définies dans la classe
  - Permet de contrôler les valeurs des attributs de la classe
- **public** : Utilisé pour la plupart des méthodes
  - Les éléments de cette section sont accessibles par tous les programmes

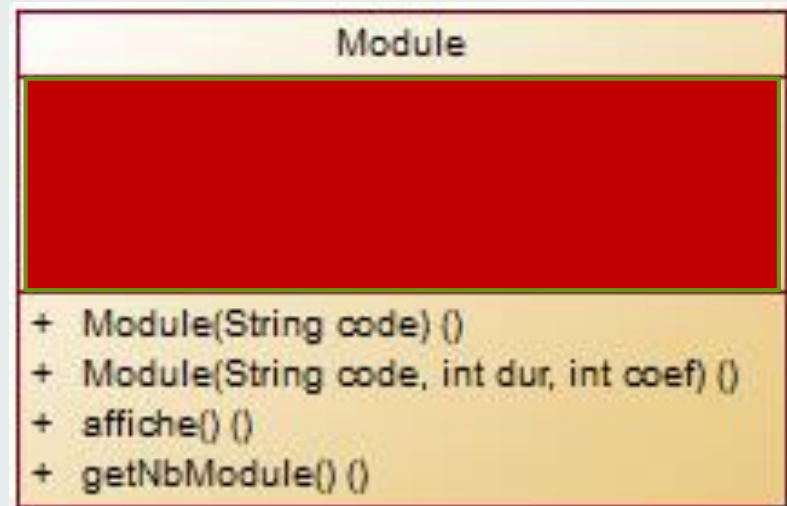


# PORTÉE DES ÉLÉMENTS DE LA CLASSE

- **protected** : Utilisé par les attributs lorsque la classe peut être héritée
  - Les éléments de cette section sont accessibles uniquement par les méthodes définies dans la classe et dans les classes qui en héritent
- **Remarque** : en java la portée **protected** définit que les éléments sont accessibles par la classe, les classes qui en héritent et les classes présentes dans le même package.

# PROGRAMME

- Le programme peut utiliser uniquement les éléments de portée public de la classe



# CRÉATION DES OBJETS

# UTILISATION DES CONSTRUCTEURS

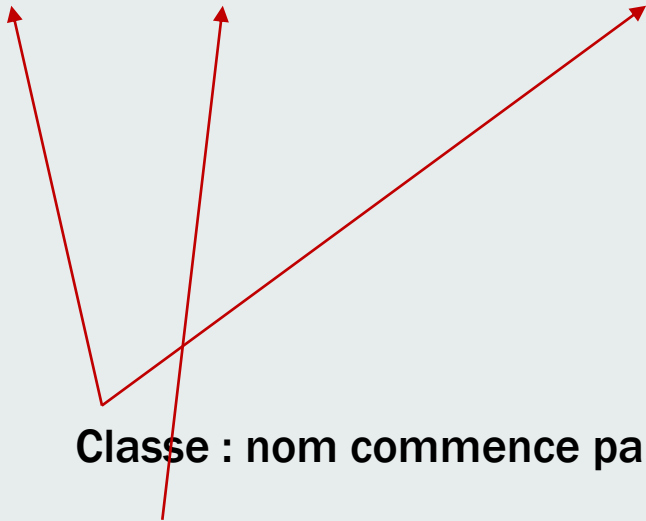
- Une classe contient au minimum un constructeur qui porte le nom de la classe
- Un constructeur a pour rôle d'initialiser les attributs lors de la création d'un objet
  - L'objet créé doit avoir un état cohérent

# CRÉATION D'UN OBJET

## ■ Syntaxe

TypeObjet nomObjet = intialisation grâce à un constructeur;

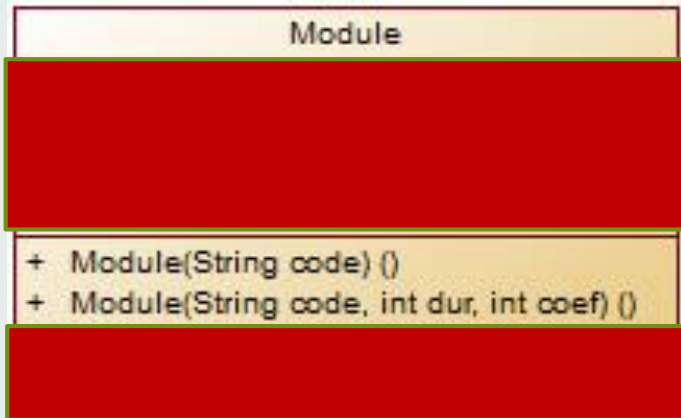
NomClasse nomObjet = new NomClasse();



**Classe** : nom commence par une majuscule

**Objet** : nom commence par une minuscule

# EXEMPLE



- `Module mod1 = new Module(«JAVA »);`
- `Module mod2 = new Module(« UML », 26, 1,5);`

Nom des objets choisis  
par le développeur

# RÉFÉRENCE NULL

- Référence ne désignant aucun objet en mémoire

```
Module mod = null;
```

```
if (mod==null)  
    mod = new Module(« nom »);
```

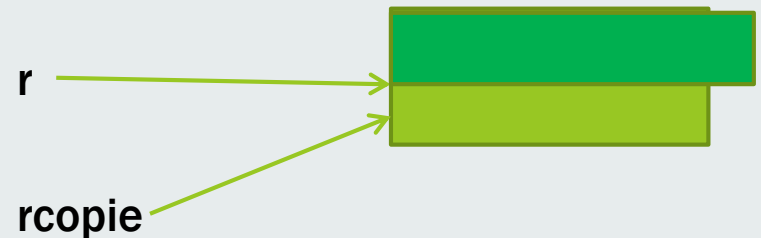
- Valeur par défaut des variables références d'objet non initialisées

# RÉFÉRENCE D'OBJETS

- L'affectation d'un objet consiste à créer une nouvelle référence sur le même objet

- Pour créer une copie d'un objet, on utilisera la méthode clone (à définir dans la classe)

- `Rectangle r = new Rectangle(8,6);`
- `Rectangle rcopie = r;`
- `rcopie.modifRectangle(10,3);`



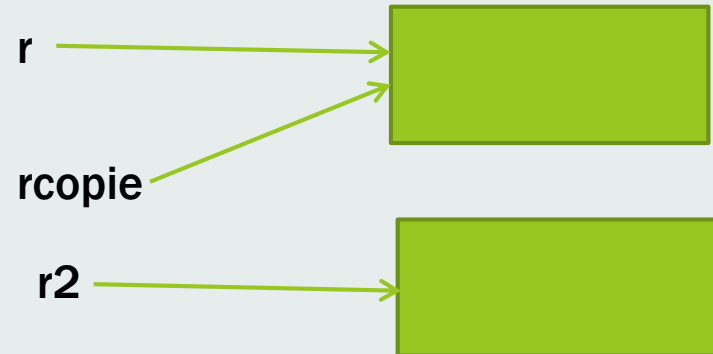


# EGALITÉ DES OBJETS

- L'opérateur `==` entre des objets teste l'égalité des références d'objet

- Pour tester si deux objets sont égaux, on utilisera la méthode `equals` (à définir dans la classe)

- `Rectangle r = new Rectangle(8,6);`
- `Rectangle rcopie = r;`
- `Rectangle r2 = new Rectangle(8,6);`

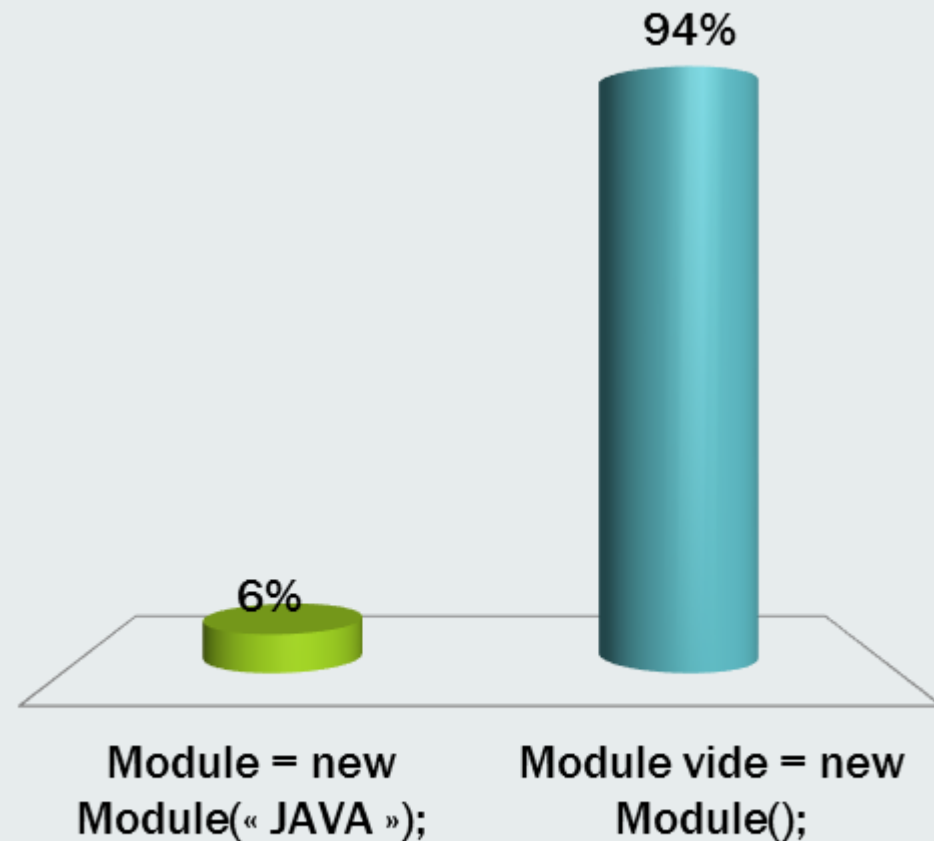


`r == rcopie` **true**

`r == r2` **false**

# QUELLE SYNTAXE EST CORRECTE ?

- A. `Module = new  
Module(« JAVA »);`
- B. `Module vide = new  
Module();`

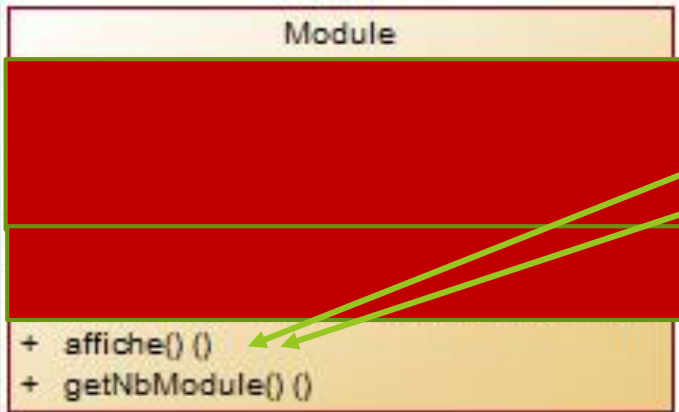


# LES MÉTHODES

# LES MÉTHODES

- Fonctions ou procédures créées dans une classe
- Sont appelées par l'intermédiaire d'un objet créé au préalable
- Syntaxe d'appel :  
`varResult = nomObjet.nomMethode(valeurParam1, valeurParam2...)`

# EXEMPLE



- `Module mod1 = new Module(«JAVA »);`
- `Module mod2 = new Module(« UML », 26, 1,5);`
- `mod1.affiche();`
- `mod2.affiche();`

**Module Java**

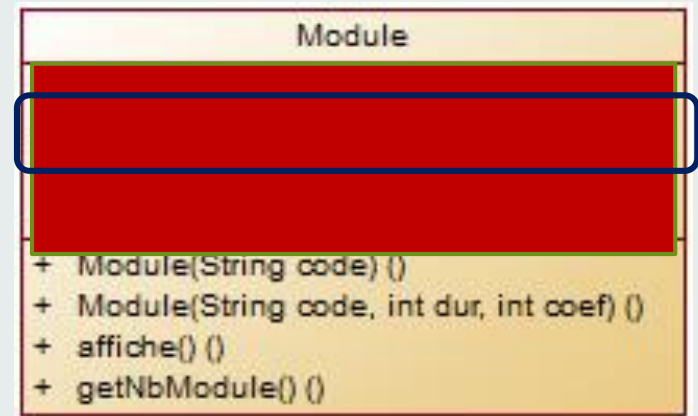
**0 heures, coefficient 0**

**Module UML**

**26 heures, coefficient 1,5**

# ACCESSEUR / MUTATEUR

- Permettent de donner accès aux attributs privés de la classe
  - Accesseur : en lecture
  - Mutateur : en écriture



# EXEMPLE : ATTRIBUT DUREE

## Accesseur

```
public int getDuree()  
{  
    return duree;  
}
```

## Mutateur

```
public void setDuree(int  
newDuree)  
{  
    duree=newDuree;  
}
```

# EXEMPLE C#

- L'attribut privé est encapsulé dans un attribut public disposant de getter et setter

C#

```
class Person
{
    private string name; // the name field
    public string Name    // the Name property
    {
        get
        {
            return name;
        }
        set
        {
            name = value;
        }
    }
}
```

Name = « Durand » est équivalent à setName(« Durand »)

L'affichage de Name est équivalent à l'affichage de getName()



# ATTRIBUTS/METHODES DE CLASSE

# ATTRIBUT/MÉTHODE DE CLASSE

## Attribut de classe

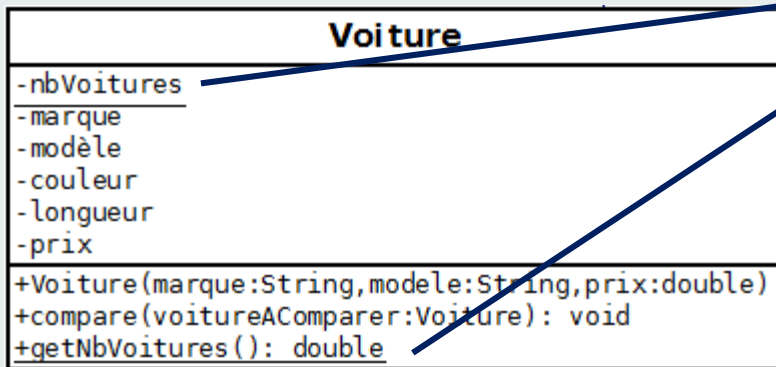
- Attribut commun à tous les objets de la classe
- Valeur accessible par tous les objets de la classe créés par l'application

## Méthode de classe

- Méthode de la classe qui est indépendante des objets de la classe
  - N'a pas accès aux attributs standards de la classe
  - A accès aux attributs de classe

# ATTRIBUT/MÉTHODE DE CLASSE

UML



JAVA

- Ajout du mot clé static
- Appel d'une méthode de classe à travers le nom de la classe (ne nécessite pas la création d'un objet)

# ATTRIBUT DE CLASSE

nbVoitures

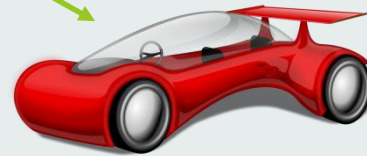
3



Marque:Audi  
Modele:A3  
Couleur:vert  
Longueur:3,8  
Prix:25600

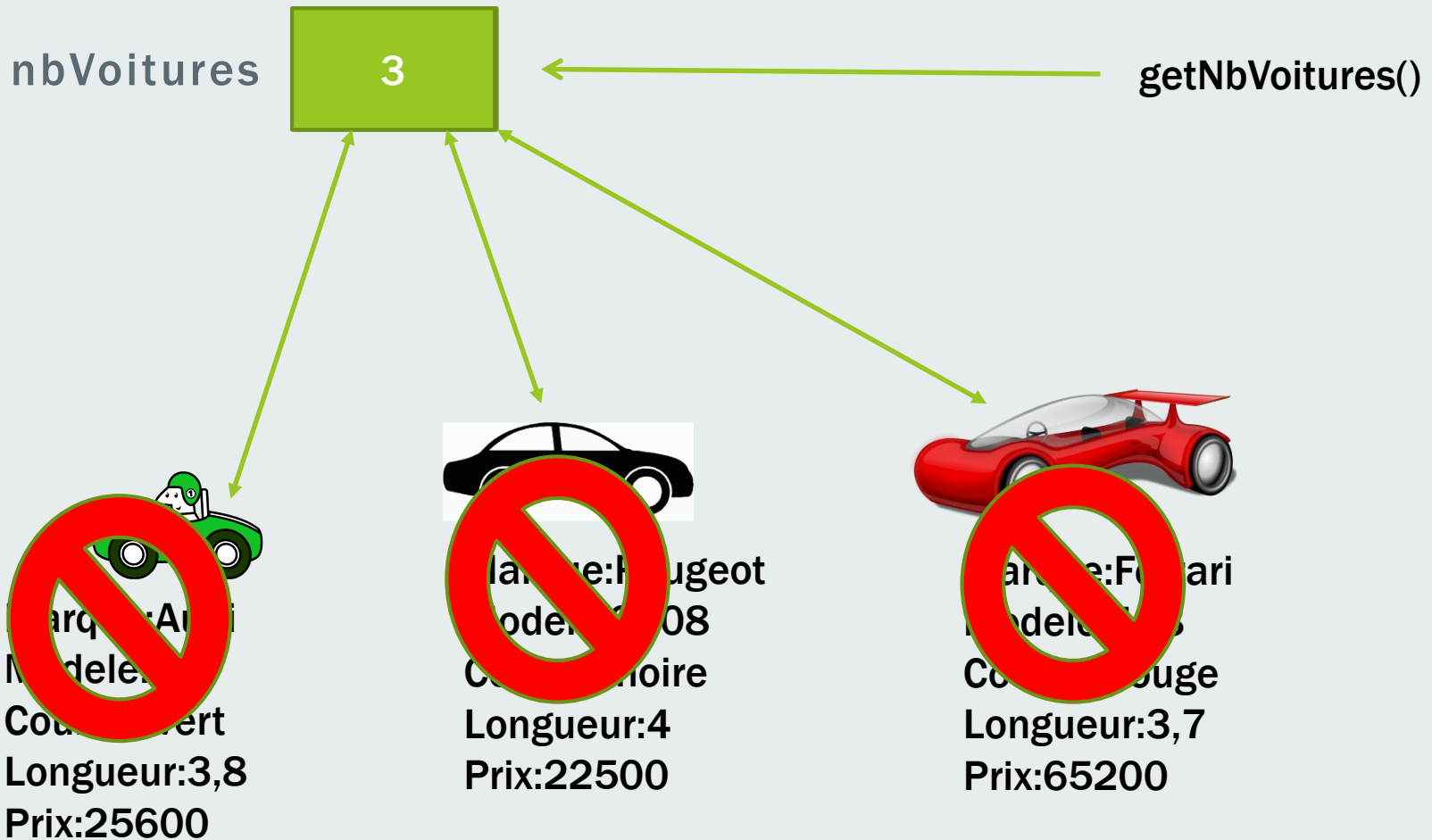


Marque:Peugeot  
Modele:3008  
Couleur:noire  
Longueur:4  
Prix:22500



Marque:Ferrari  
Modele:488  
Couleur:rouge  
Longueur:3,7  
Prix:65200

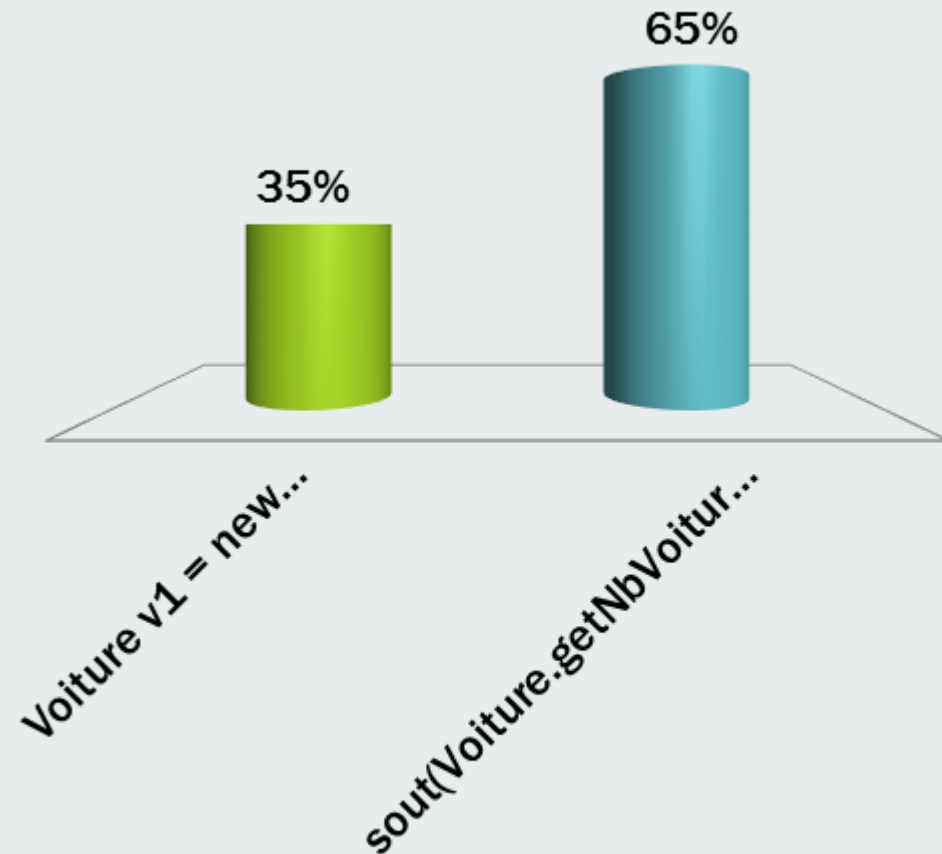
# METHODE DE CLASSE



# QUEL EST LE BON APPEL DE LA MÉTHODE STATIC ?

- A. `Voiture v1 = new Voiture(...);`  
`sout(v1.getNbVoitures());`
- B. `sout(Voiture.getNbVoitures());`

Voiture
-nbVoitures
-marque
-modèle
-couleur
-longueur
-prix
+Voiture(marque:String,modele:String,prix:double)
+compare(voitureAComparer:Voiture): void
+getNbVoitures(): double



# RELATIONS ENTRE CLASSE

# RELATION DE DÉPENDANCE

- Un module a un enseignant responsable





# RELATION DE DÉPENDANCE

## UML

- Représenté par

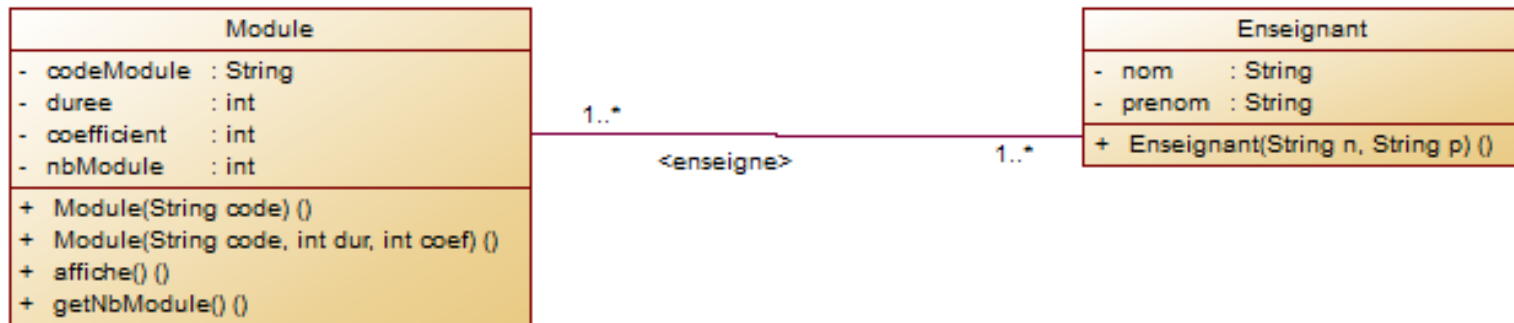


## Java

- La classe d'où part la relation de dépendance : Module
- Contient dans ses attributs un objet de la classe pointée par la flèche : Enseignant

# RELATION BIDIRECTIONNELLE

- Un module est enseigné par 1 à plusieurs enseignants
- Un enseignant enseigne 1 à plusieurs modules



# RELATION BIDIRECTIONNELLE

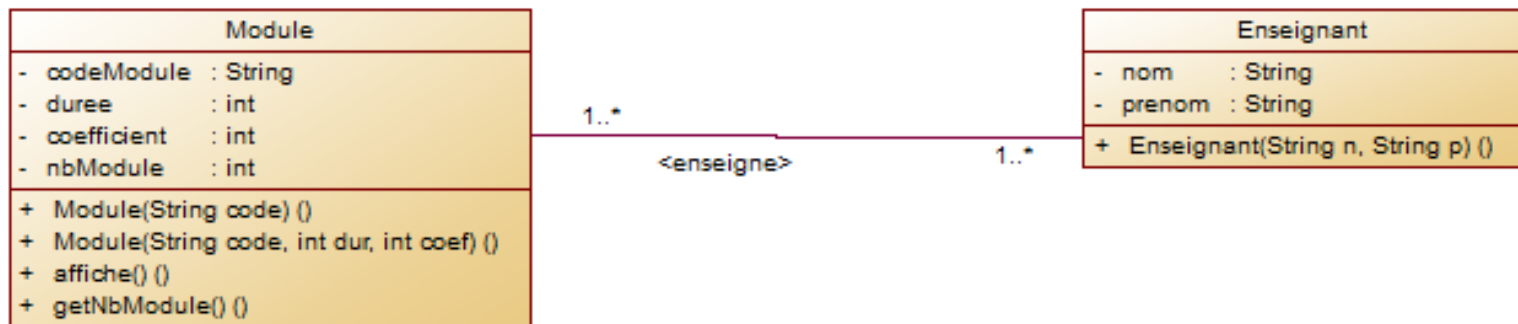
- Représentée par

1 \*

- Dans les deux classes reliées, on trouvera un ou plusieurs objets de la classe en lien
- 1 objet pour 1
- Une liste d'objets pour \*

# EXEMPLE

- Dans la classe Module, on trouve une liste d'objets Enseignant
- Dans la classe Enseignant, on trouve une liste d'objets Module
- La cohérence des listes est gérée par l'application



# RELATION D'AGRÉGATION

- Notion d'appartenance
- Un immeuble appartient à plusieurs propriétaires
- Un propriétaire peut posséder plusieurs immeubles



# RELATION DE COMPOSITION

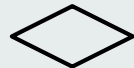
- Cas particulier de l'agrégation, implique des contraintes supplémentaires
- Multiplicité de 1 maximum obligatoire pour l'agrégat
- Un immeuble comprend plusieurs appartements
- Un appartement est situé dans 1 et 1 seul immeuble
  - L'appartement ne peut pas exister sans immeuble
  - Si l'immeuble est détruit, l'appartement l'est aussi



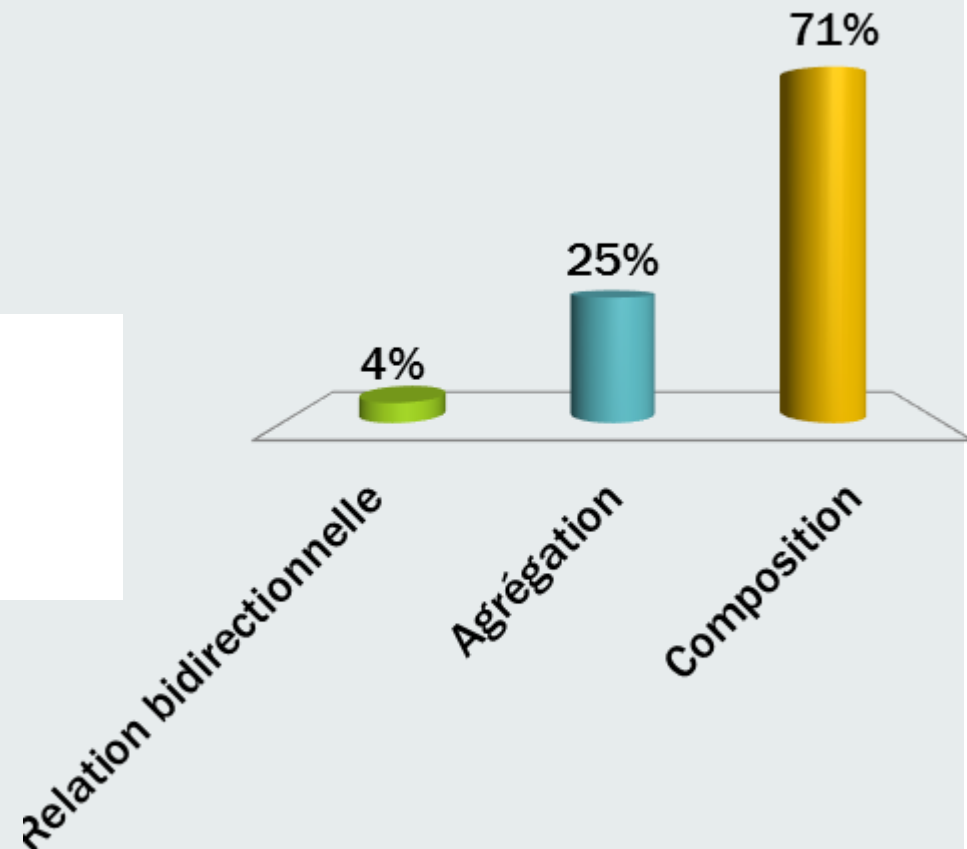
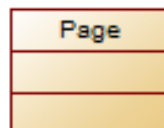
# QUEL TYPE DE RELATION DOIT-ON DÉFINIR ENTRE CES CLASSES ?

A. Relation bidirectionnelle

B. Agrégation



C. Composition



# COMPOSITION

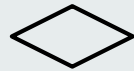




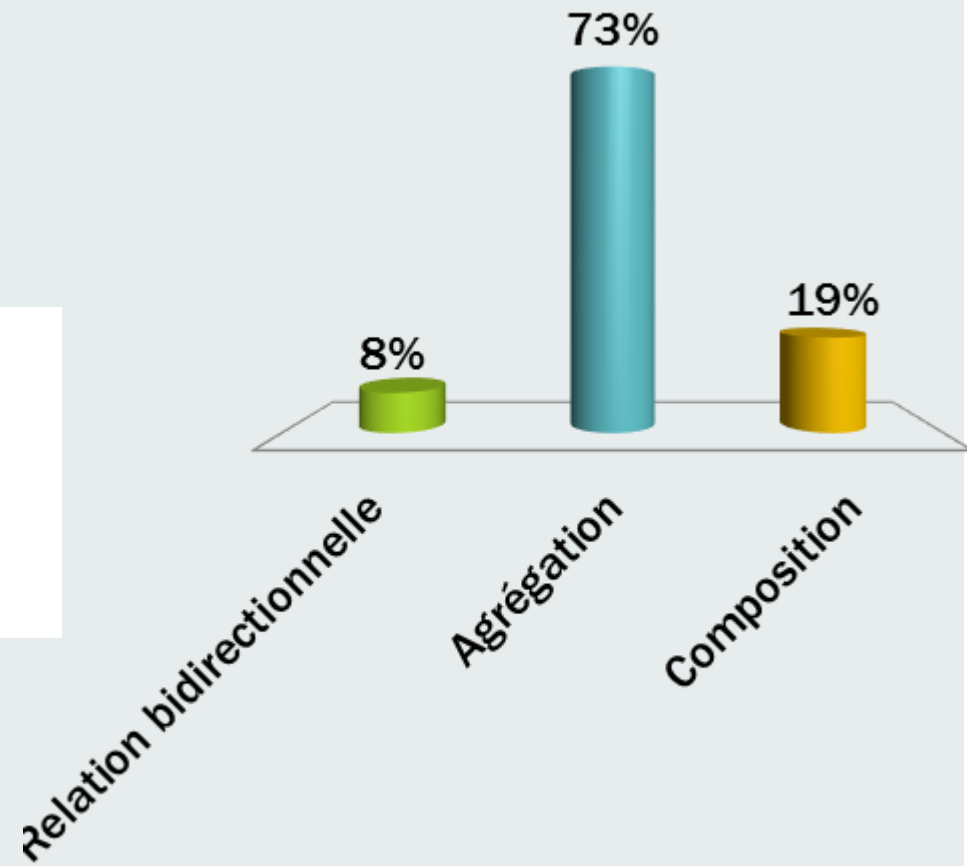
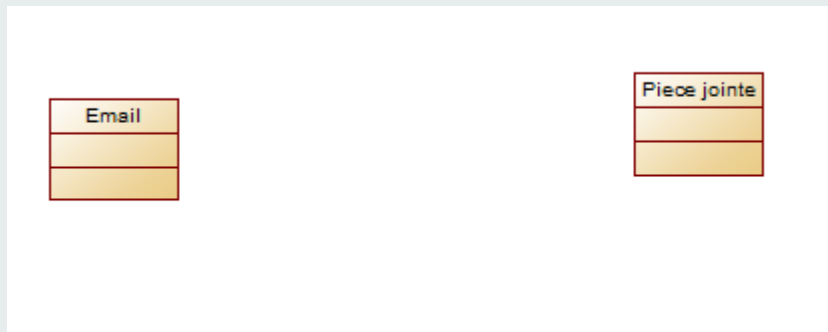
# QUEL TYPE DE RELATION DOIT-ON DÉFINIR ENTRE CES CLASSES ?

A. Relation bidirectionnelle

B. Agrégation



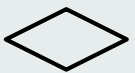

C. Composition

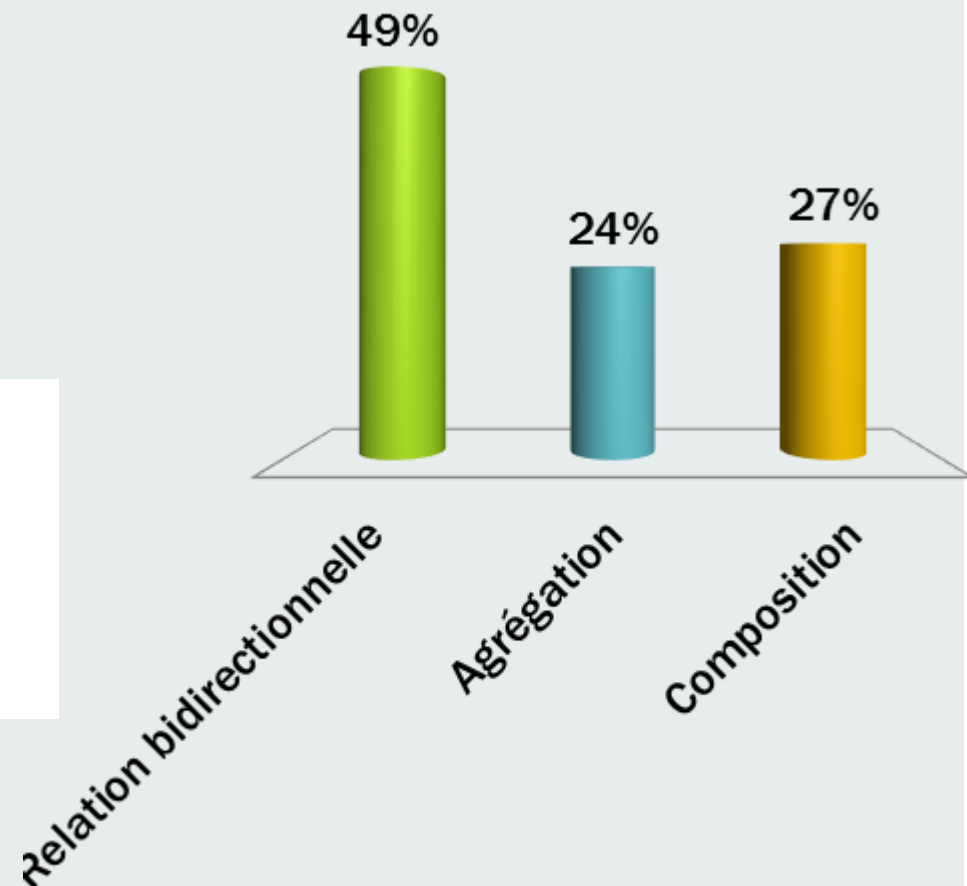
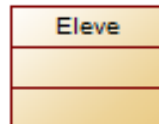
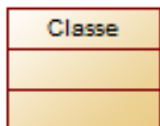


# AGREGATION



# QUEL TYPE DE RELATION DOIT-ON DÉFINIR ENTRE CES CLASSES ?



- A. Relation bidirectionnelle
- B. Agrégation 
- C. Composition 

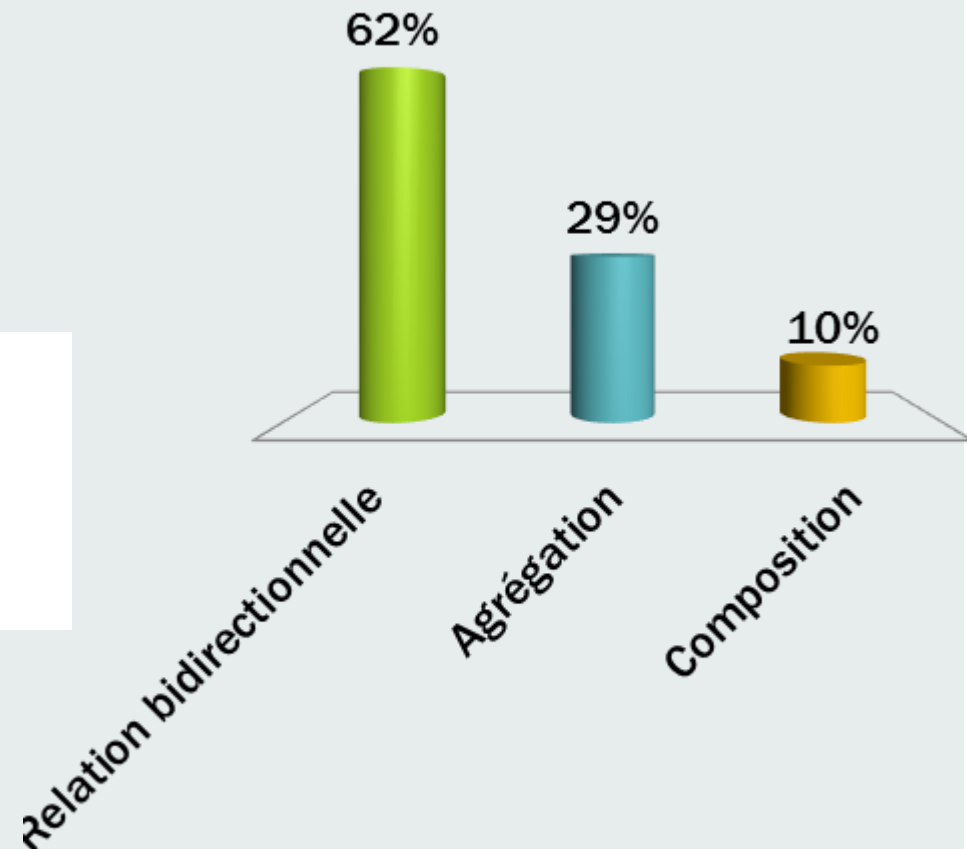
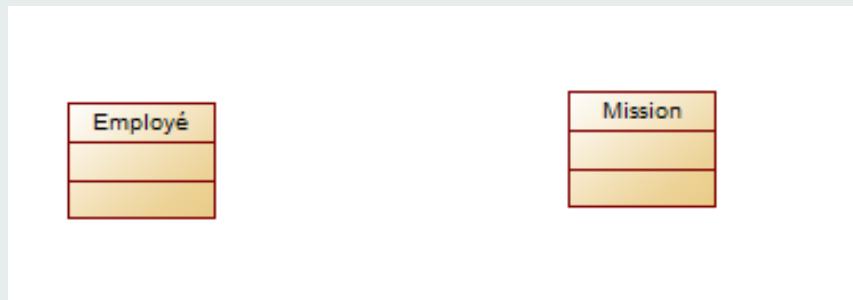


# AGREGATION

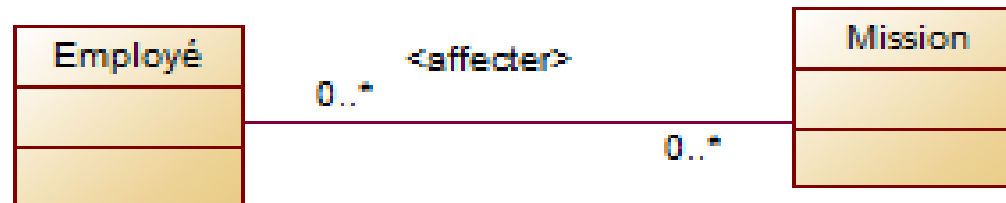


# QUEL TYPE DE RELATION DOIT-ON DÉFINIR ENTRE CES CLASSES ?

- A. Relation bidirectionnelle
- B. Agrégation 
- C. Composition 



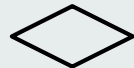
# RELATION BIDIRECTIONNELLE



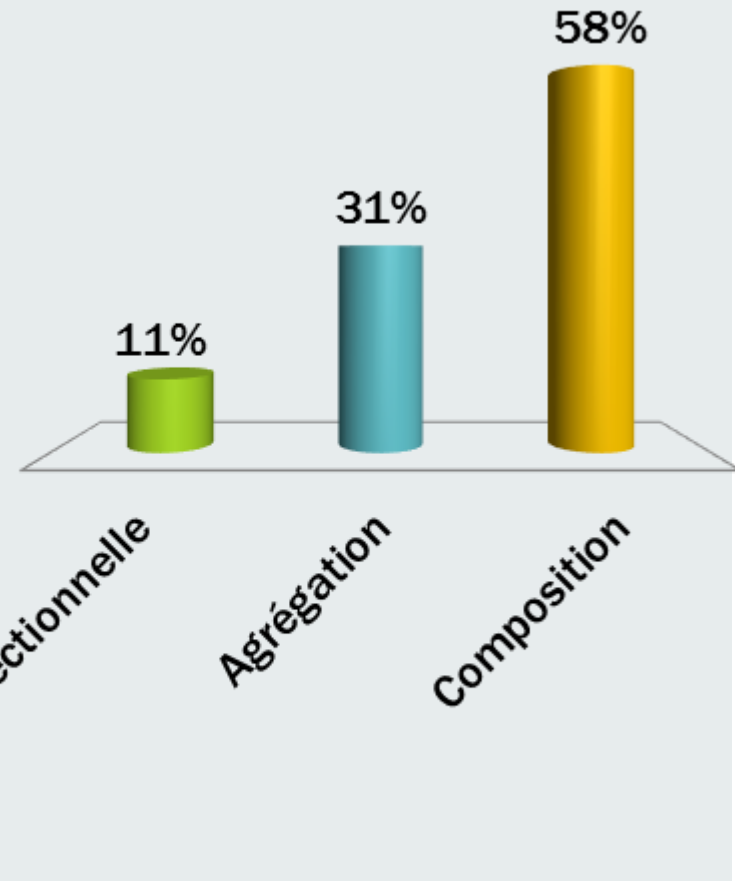
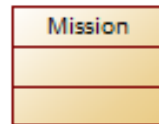
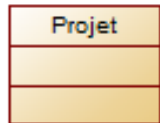
# QUEL TYPE DE RELATION DOIT-ON DÉFINIR ENTRE CES CLASSES ?

A. Relation bidirectionnelle

B. Agrégation



C. Composition



# COMPOSITION

