



PARTIE 2 : CONCEPTION

Chapitre 4 :

Conception dynamique


Diagramme de séquences objet

Diagramme d'états-transition

3^{ème} année A

Année universitaire :
2020-2021

Henda SFAXI
henda.sfaxi@esprit.tn



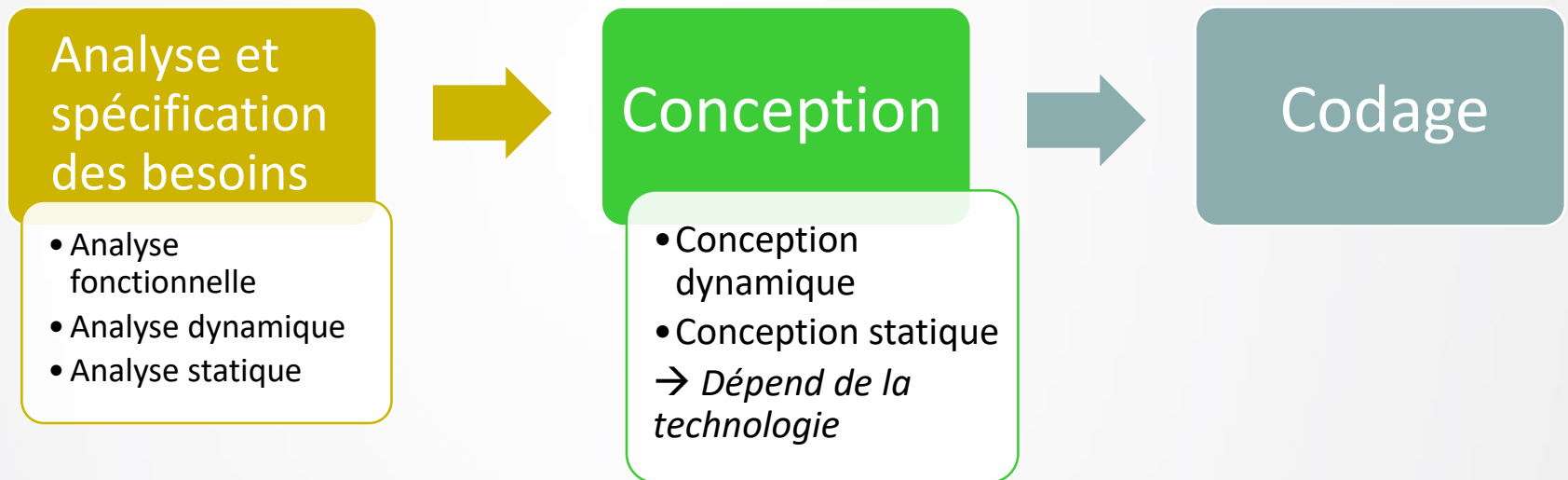
Plan du cours

- Mise en contexte
- Objectifs de la conception dynamique
- [Diagramme de séquences objet](#)
- [Diagramme d'états-transition](#)



Mise en contexte

► Où en sommes nous?





Diagrammes d'interaction



Objectifs

- Décrire *comment* les objets interagissent au sein du système
 - Montrer les interactions entre les objets dans le temps
 - Présenter la séquence (l'ordre) et la dépendance des messages échangés entre les objets
- Représenter certains aspects dynamiques pour la réalisation
 - d'un cas d'utilisation
 - d'une opération relative à un cas d'utilisation



Types

- Diagramme de séquence (système et objet)
 - Diagramme de séquence système → phase d'analyse
 - Diagramme de séquence objet → phase de conception
- Diagramme de communication
- Diagramme de temps



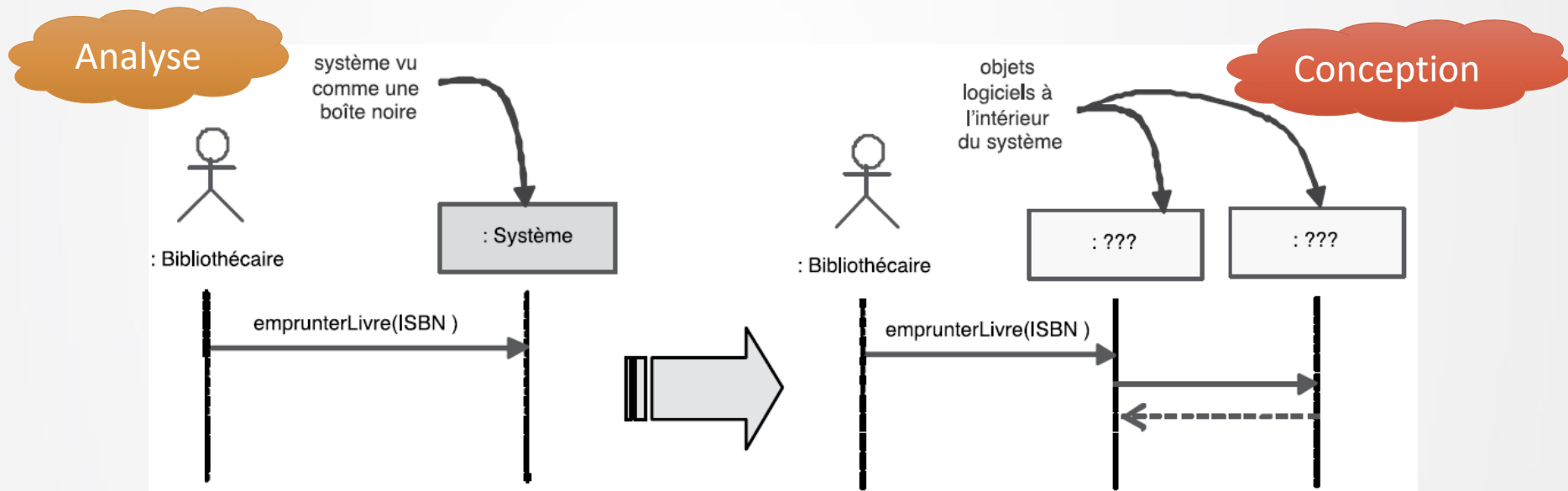
Diagramme de séquence objet



Présentation

- Un diagramme de séquence est à deux dimensions:
 - Dimension verticale : le temps
 - L'**ordre** (le **séquencement**) d'envoi d'un message est déterminé par la position du message sur l'axe vertical du diagramme
 - Le temps s'écoule "de haut en bas" de cet axe
 - Dimension horizontale : les objets (et les acteurs)

Diagramme de séquence système Vs. diagramme de séquence objet



Source : UML2 par la pratique



Concepts clés

- Objets
- Lignes de vie
- Zones d'activation
- Messages
- Structures de contrôle :
 - Fragments combinés

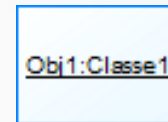
► Objet - Ligne de vie



- Objet :

- Représente une instance :

- d'un acteur
- d'une classe



- Syntaxe : NomObjet : NomClasse
NomPersonne : NomActeur

- Ligne de vie :

- Traduit l'existence d'un objet pendant une période de temps

► Messages



- Les objets communiquent entre eux via des messages

- Types des messages:

- Message synchrone 

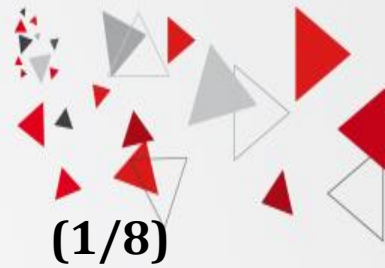
- Message asynchrone 


- Message de retour 

- Message de création 

- Message de destruction 

► Messages synchrones




- Un message **synchrone** : traduit l'invocation d'une opération:
 - Un objet demande à un objet appelé d'exécuter une opération
 - L'objet **émetteur** reste *bloqué* le temps de l'exécution de la méthode par l'objet récepteur
 - Cette durée est appelée : **durée d'activation**
- Syntaxe :

[résultat =] nomOpération([paramètres])

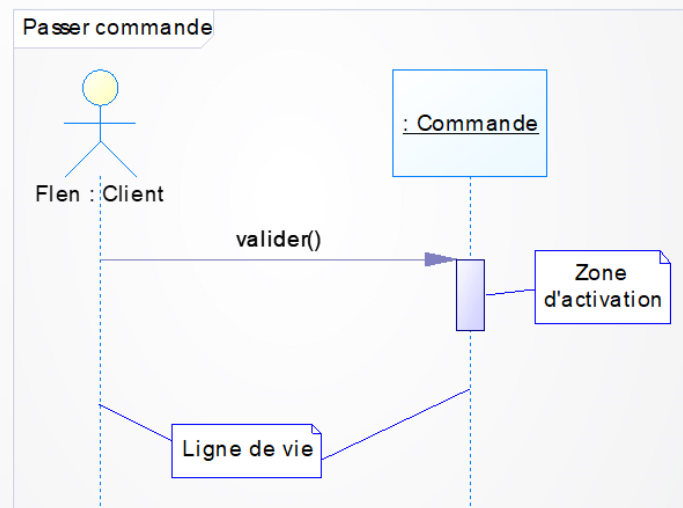
 Optionnel

► Messages synchrones

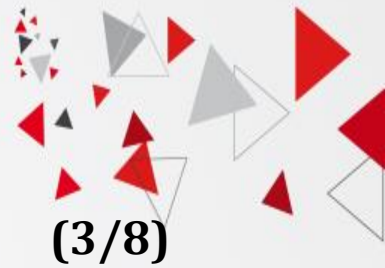
(2/8)


- Zone d'activation :
 - Rôle :
 - Représente le temps durant lequel un objet est actif : en train d'exécuter une opération (méthode)
 - N'est utilisée qu'avec les messages **synchrones**
 - Syntaxe : 

- Exemple :

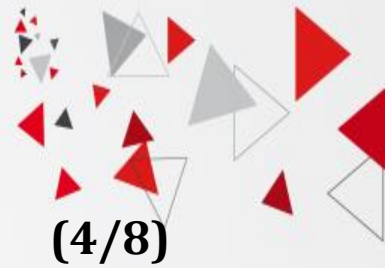


► Messages synchrones

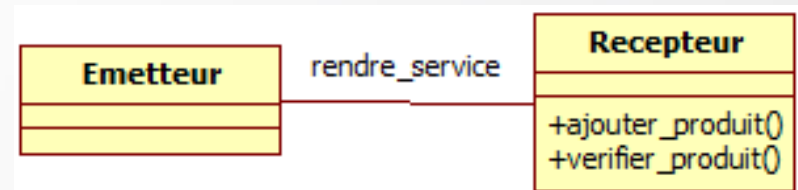
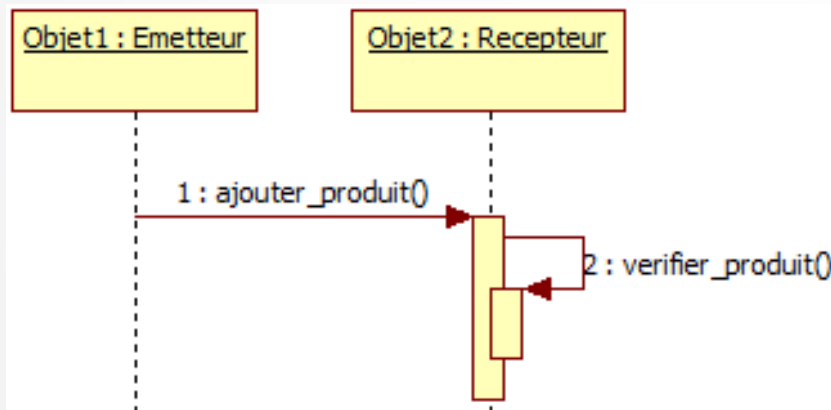


- Un message ***synchrone réflexif*** 
 - L'objet invoque une méthode locale : envoie d'un message à lui-même

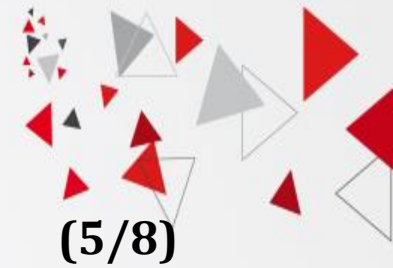
► Messages synchrones



- Remarque 1 :
 - L'opération invoquée doit être définie dans la classe relative à l'objet récepteur et visible dans la classe relative à l'objet émetteur
- Exemple :



► Messages synchrones

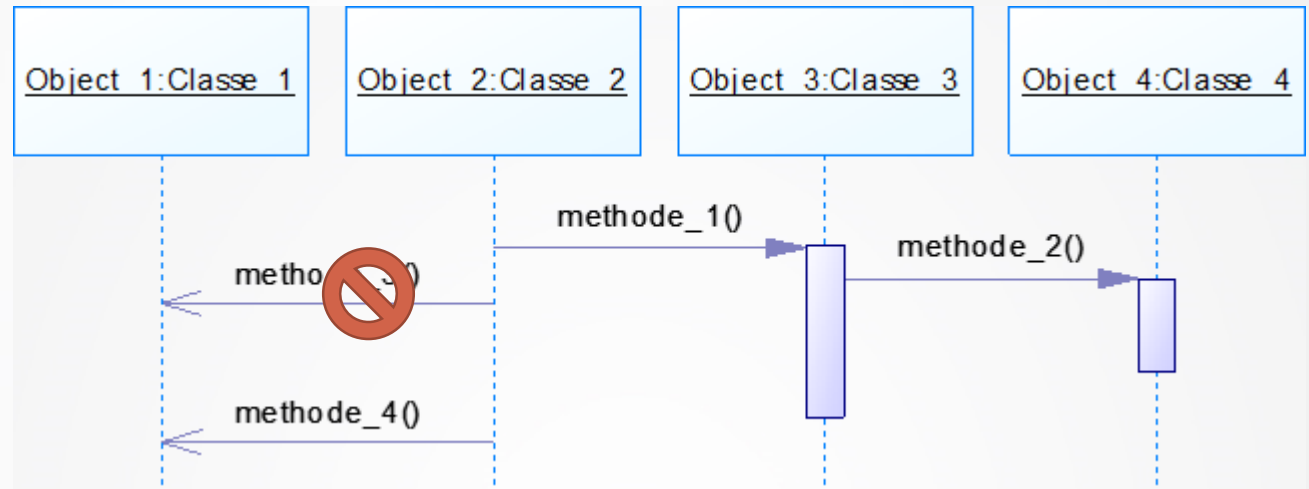


- Remarque 2 :
 - Le message synchrone est bloquant pour l'émetteur
 - L'objet qui demande l'exécution d'une méthode ne peut pas invoquer d'autres méthodes avant la fin de cette méthode
 - Le message synchrone n'est pas bloquant pour l'objet qui l'exécute
 - L'objet qui exécute une méthode peut invoquer d'autres méthodes imbriquées

► Messages synchrones

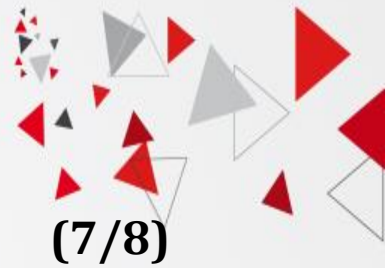
(6/8)

- Exemple :



- Objet_2 est bloqué en attendant la fin de methode_1()
 - Ne peut envoyer aucun message
- Objet_3 est en train d'exécuter methode_1()
- Objet_3 invoque la methode_2()
 - La methode_1() contient un appel imbriqué à methode_2()

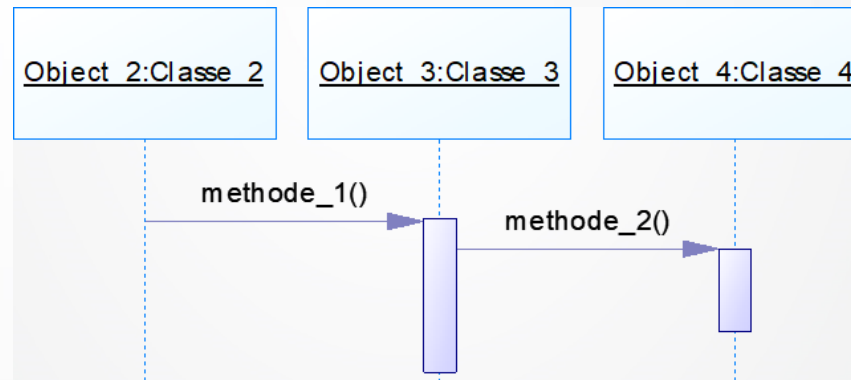
► Messages synchrones



(7/8)

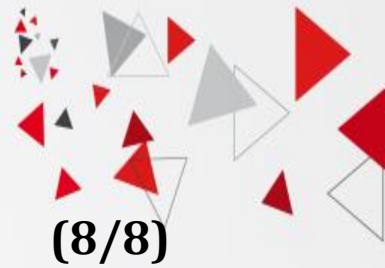
- Remarque 3 :
 - Une méthode peut invoquer une autre méthode
 - ➔ Appel imbriqué de méthodes (Exemple: D17)
 - La méthode appelante ne peut finir qu'après la fin de la méthode appelée

- Exemple :



- La méthode `methode_1()` effectue un appel imbriqué à `methode_2()`
 - La durée d'activation de `methode_1()` ne prends fin qu'après la durée d'activation de `methode_2()`

► Messages synchrones



- Message de **création** d'instance 

- Invoquer le constructeur de la classe


- Message de **destruction** d'instance 

- Invoquer le destructeur de la classe

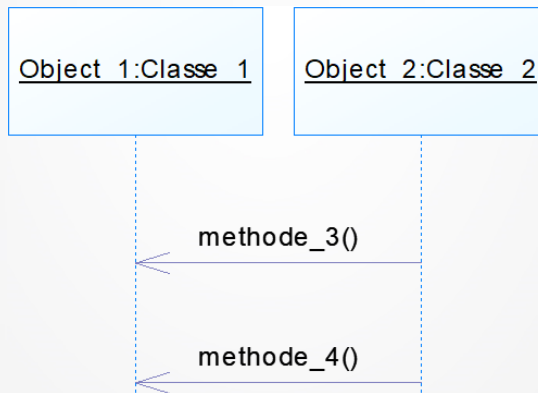
- Arrêt de la ligne de vie de l'objet

► Messages asynchrones



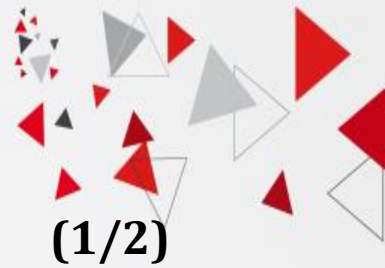
- Rôle : 
 - Envoyer un signal ou une notification à un objet
 - Exemple : Envoyer un email
 - Invoquer une méthode sans bloquer l'émetteur
 - L'émetteur n'attend pas la fin de l'exécution de la méthode pour poursuivre le traitement

- Exemple :






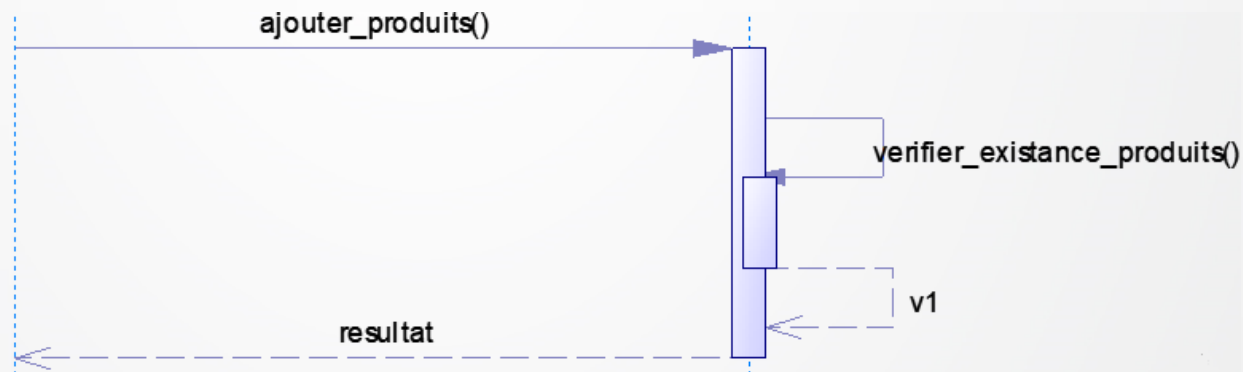
- *Object_2* ne reste pas bloqué durant l'exécution de *methode_3*
- *methode_3* n'a pas de durée d'activation

► Messages de retour

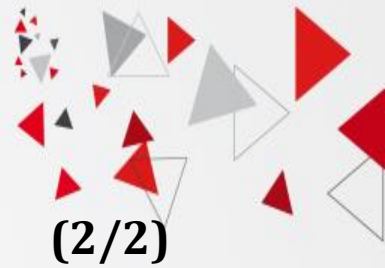


(1/2)

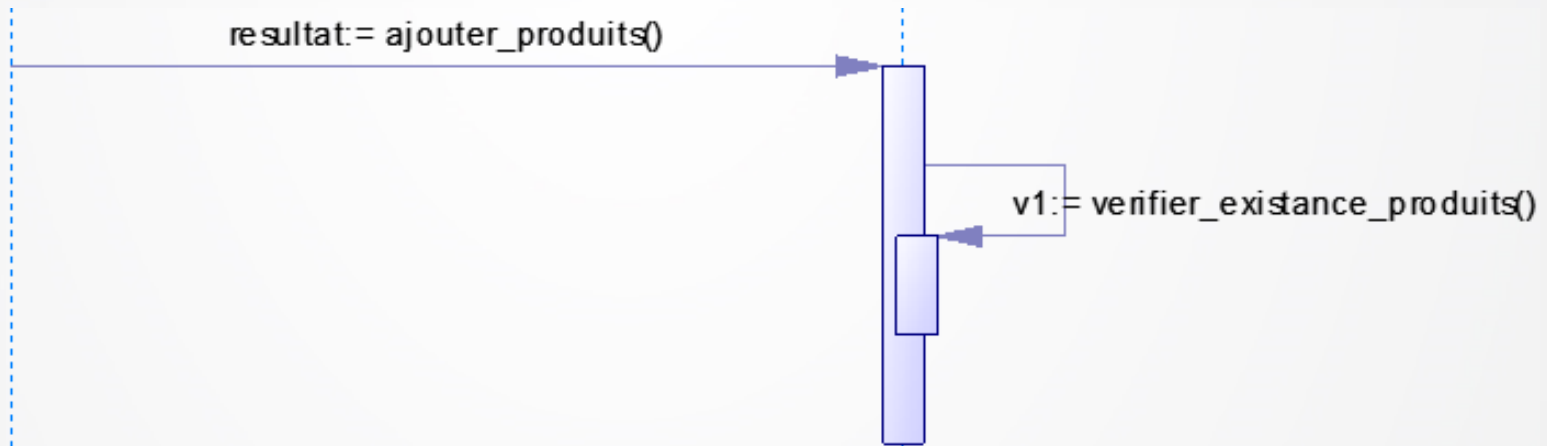
- Rôle : 
 - Renvoyer le retour d'une méthode d'un message ***synchrone***
 - La variable de retour peut être de n'importe quel type
 - Peut ne rien contenir → *Optionnel*
- Syntaxe :  
- Exemple :



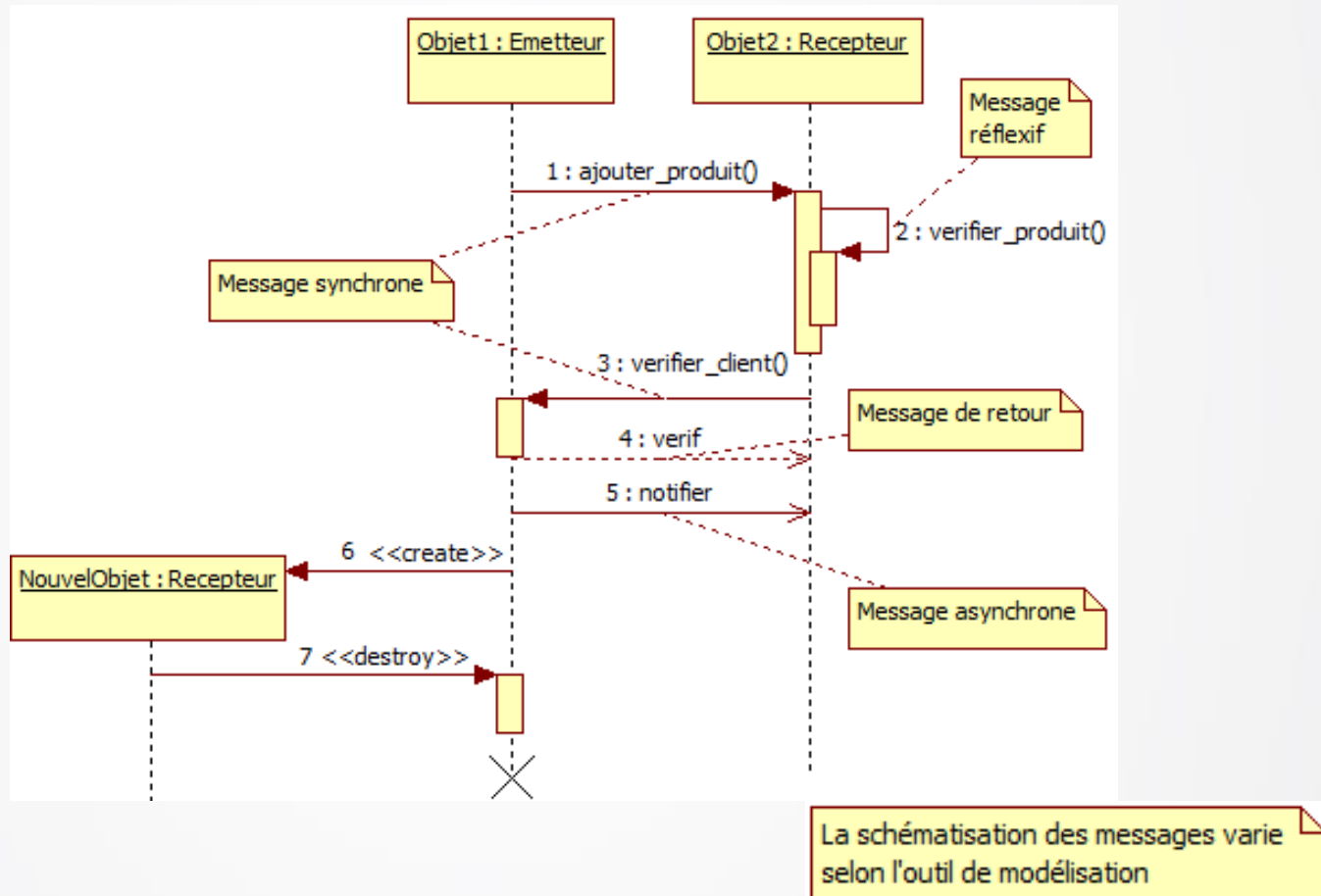
► Messages de retour



- Autre représentation :



► Représentation graphique



► Exemple

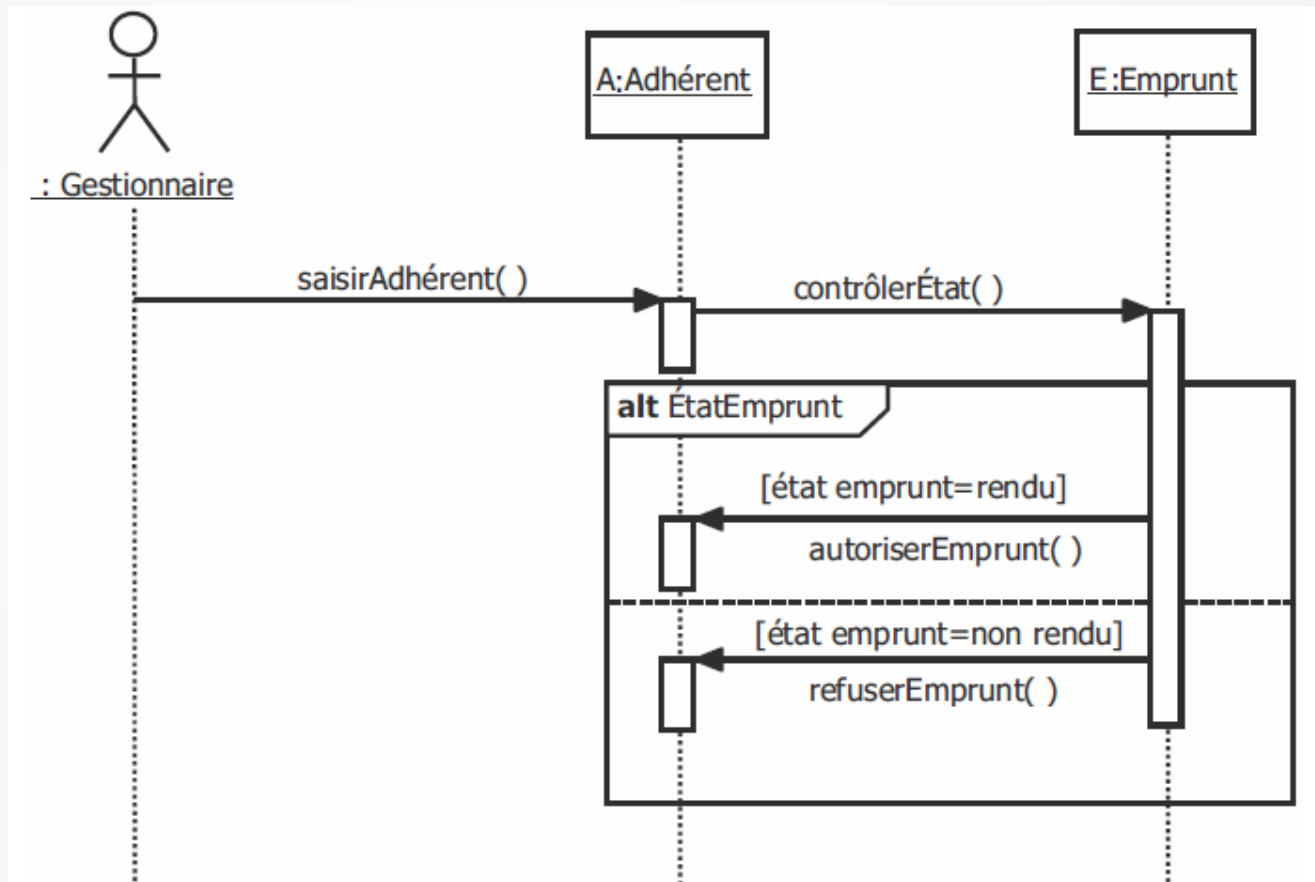
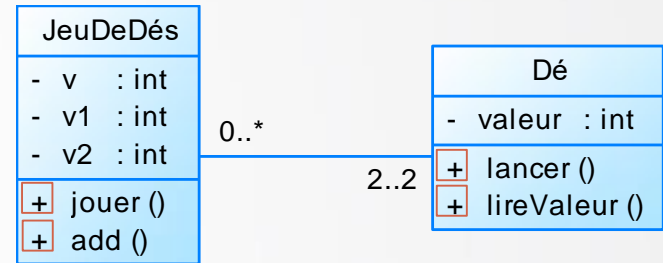
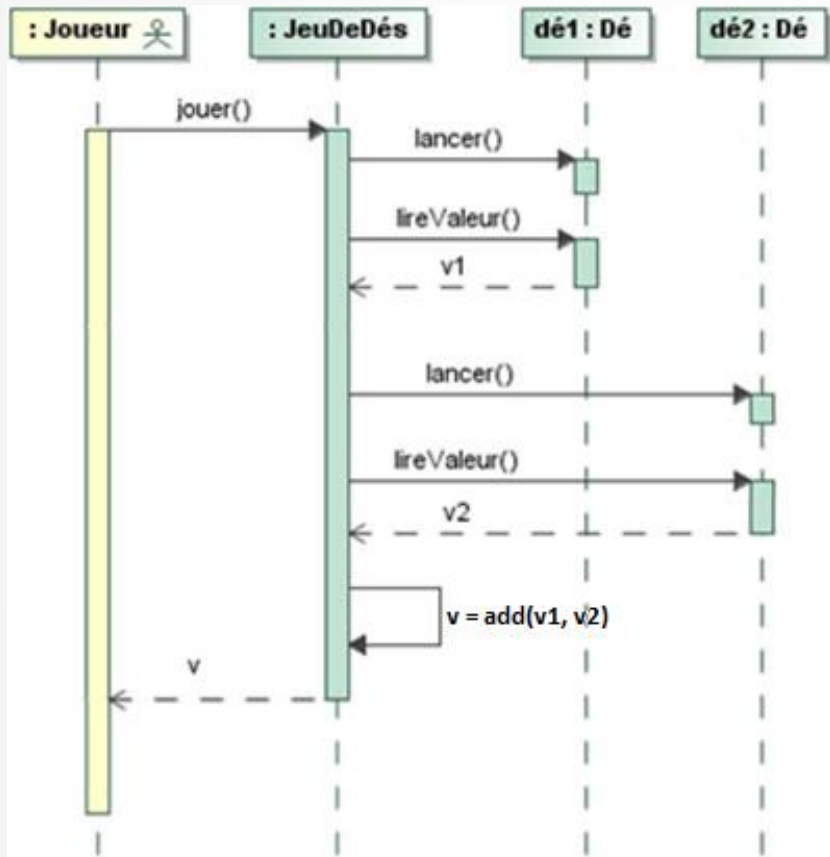


Diagramme de séquence objet du CU « Ajouter emprunt »



Exercise

▶ Traduction d'un diagramme de séquence objet en code java

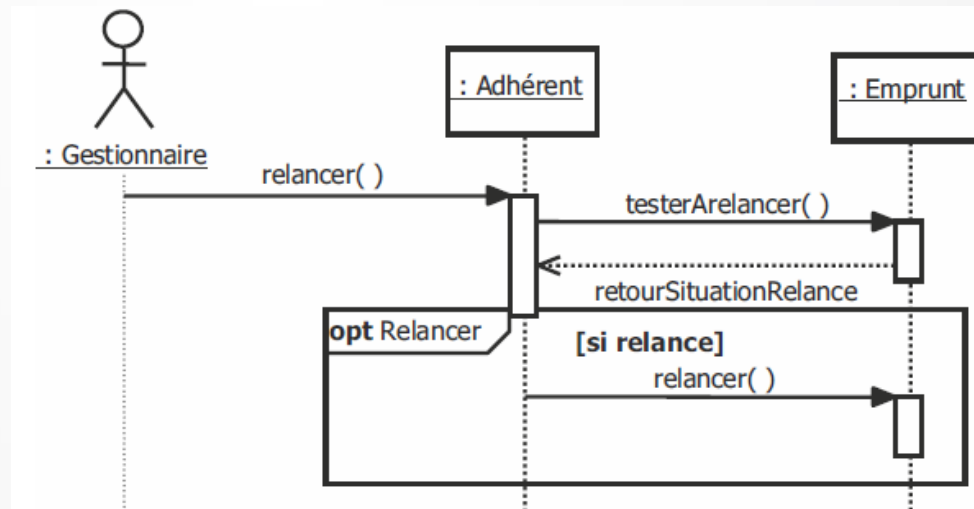


```
public int jouer()
{
    int v=0, v1, v2;
    de1.lancer();
    v1 = de1.lireValeur();
    de2.lancer();
    v2 = de2.lireValeur();
    v = add(v1, v2);
    return v;
}
```

► Structures de contrôle



- Fragments combinés
 - Fragment d'interaction comportant un opérateur d'interaction (exemple : loop, opt, alt, ref...)
 - Voir chapitre 2 : Analyse dynamique
- Exemple :



Source : Livre UML2 analyse et conception



Modéliser en utilisant l'architecture logique



Architecture logique : Bref aperçu

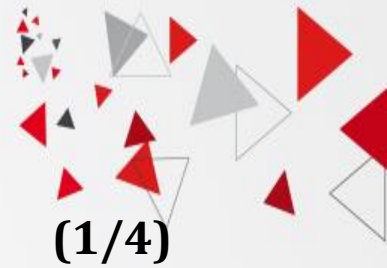


- Architecture logique ou Architecture logicielle :
 - Modèle utilisé pour :
 - Organiser le logiciel
 - Déterminer le rôle de chaque classe
 - Déterminer le type de liens et de communication entre chacun de ces rôles
 - Plusieurs types de modèles selon le framework utilisé :
 - Modèle en 2 couches (obsolète), en 3 couches, en n-couches



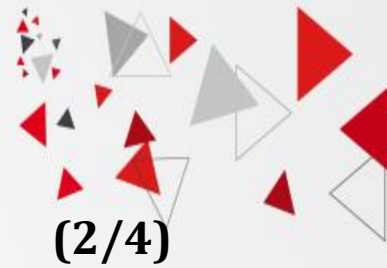
Exemple : Le modèle en 3 couches

Modèle en 3 couches



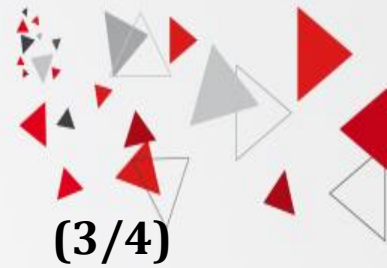
- 3 couches :
 - Couche présentation
 - Couche métier
 - Couche persistance

► Modèle en 3 couches



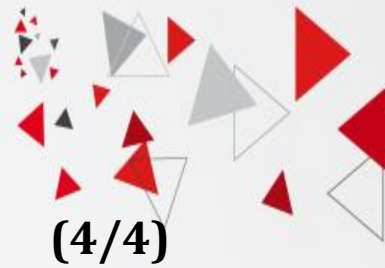
- Couche présentation
 - Représente les ***interfaces graphiques*** du logiciel dites:
 - IHM : Interface Homme-Machine
 - Ou
 - UI : User Interface
- « *Vitrine* » du logiciel :
 - Permet aux utilisateurs d'interagir avec le logiciel
 - Permet à l'acteur de visualiser et manipuler les données
- Ces interfaces graphiques peuvent avoir diverses extensions
 - Exemples : .html, .xhtml, .js, .java

► Modèle en 3 couches



- Couche métier
 - Couche intermédiaire entre les interfaces graphiques et les données
 - Regroupe les classes dites ***gestionnaires*** :
 - Contiennent les différentes méthodes offertes par le logiciel comme les *CRUD*
 - Avantages :
 - Permet la réutilisation des méthodes
 - Facilite la maintenance des méthodes

► Modèle en 3 couches



- Couche persistance ou couche des modèles
 - Regroupe les classes **entités**
 - *Remarque* : Classe entité = attributs + accesseurs
 - Permet la récupération et la manipulation des données
 - Les données sont indépendantes des traitements
 - Attention:
 - Ne pas confondre la couche persistance avec la base de données, il s'agit de deux notions différentes



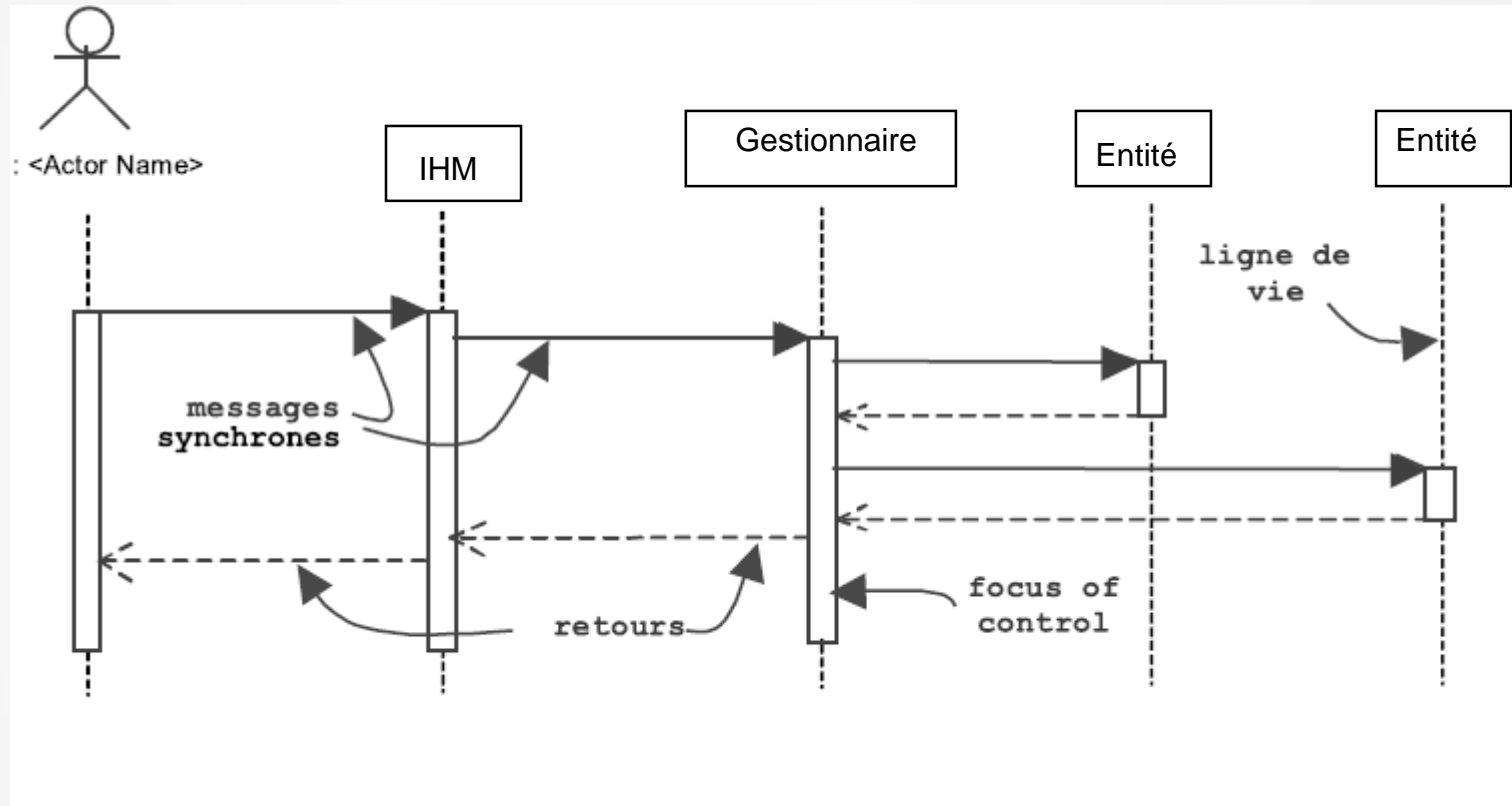
► Communication entre les couches (1/2)

- Le modèle en 3 couches peut être implémentés de différentes manières
 - Exemple : **MVC**
 - **M** : Model → représente la couche persistance
 - **V** : View → représente la couche présentation
 - **C** : Controller → représente la couche métier
- Chaque implémentation impose un genre de communication entre les couches

► Communication entre les couches (2/2)

- Pour nos exemples, nous allons considérer les sens de communication suivants :
 - Acteur \leftrightarrow Présentation
 - Présentation \leftrightarrow Présentation
 - Présentation \leftrightarrow Métier
 - Métier \leftrightarrow Métier
 - Métier \rightarrow Persistance
 - Persistance \leftrightarrow Persistance
- Les classes d'une même couche peuvent communiquer ensemble
- Les classes de la couche **métier** peuvent communiquer avec les classes de la couche **présentation** (et inversement)
- Les classes de la couche **métier** peuvent communiquer avec les classes de la couche **persistance**

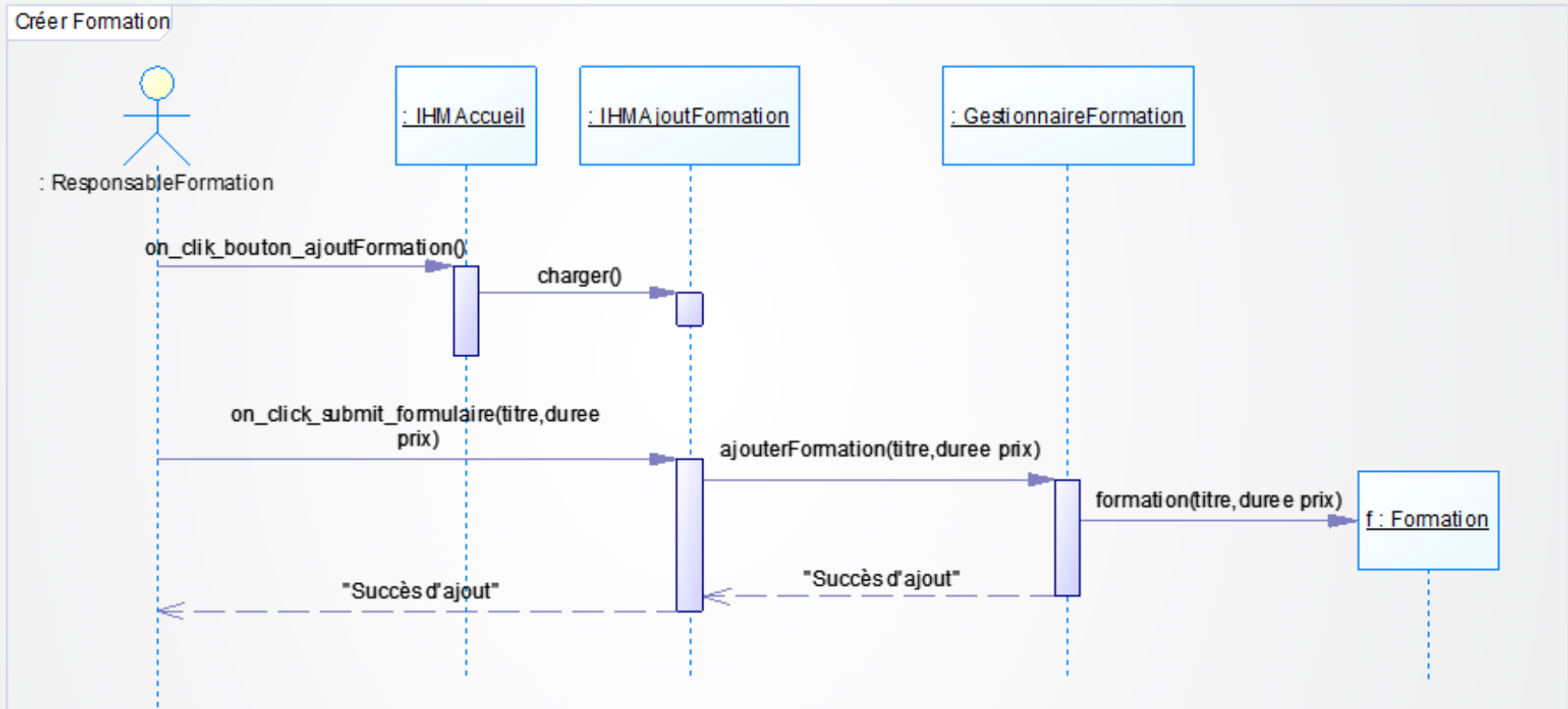
► Représentation



Source : Livre UML2 par la pratique

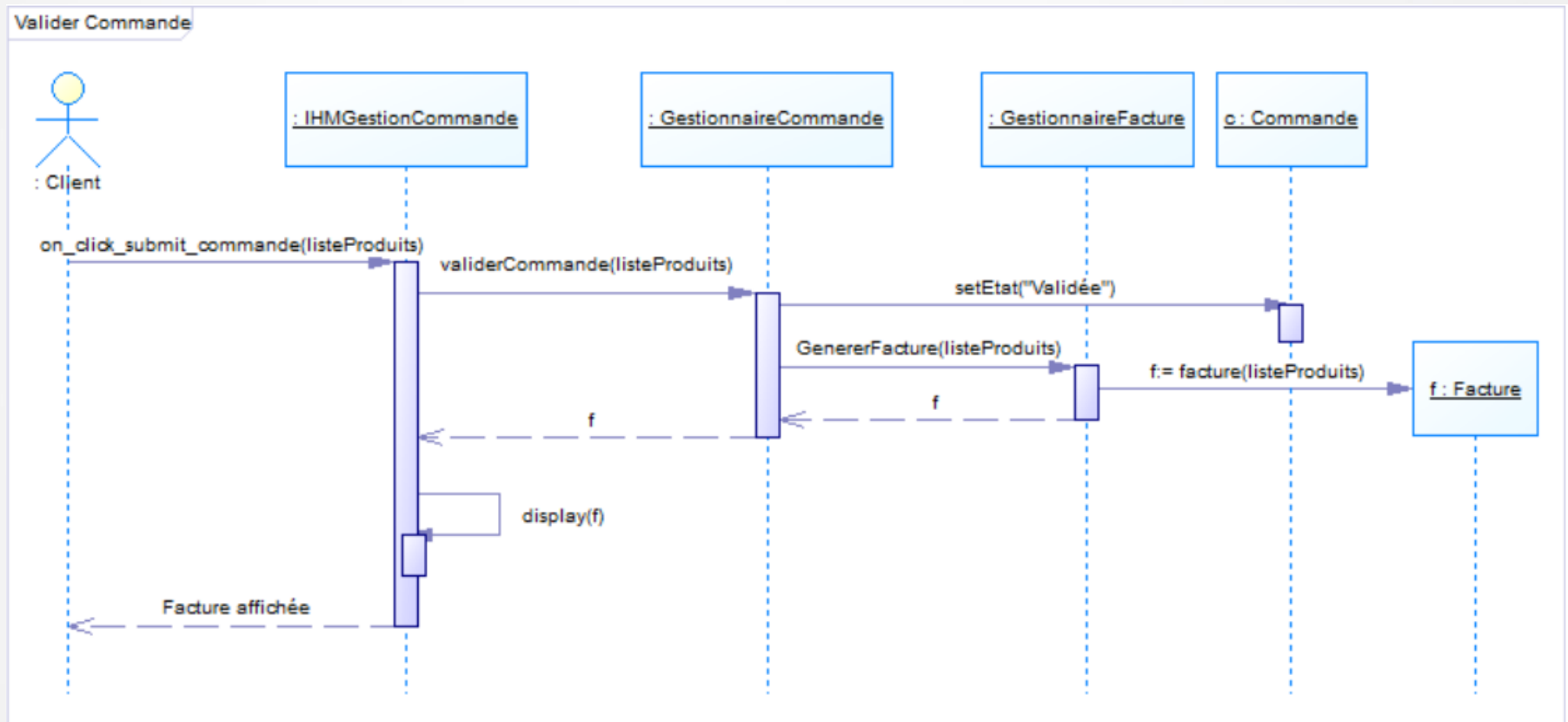


Exemple 1 : Un seul gestionnaire



Exemple 2 :

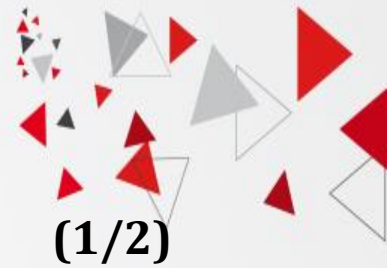
Un gestionnaire par classe entité





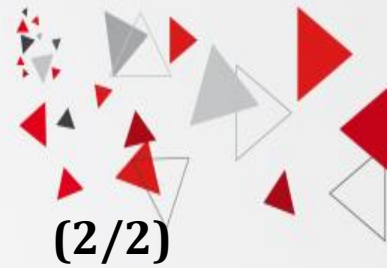
Exemple : Le modèle en 2 couches

► Modèle en 2 couches



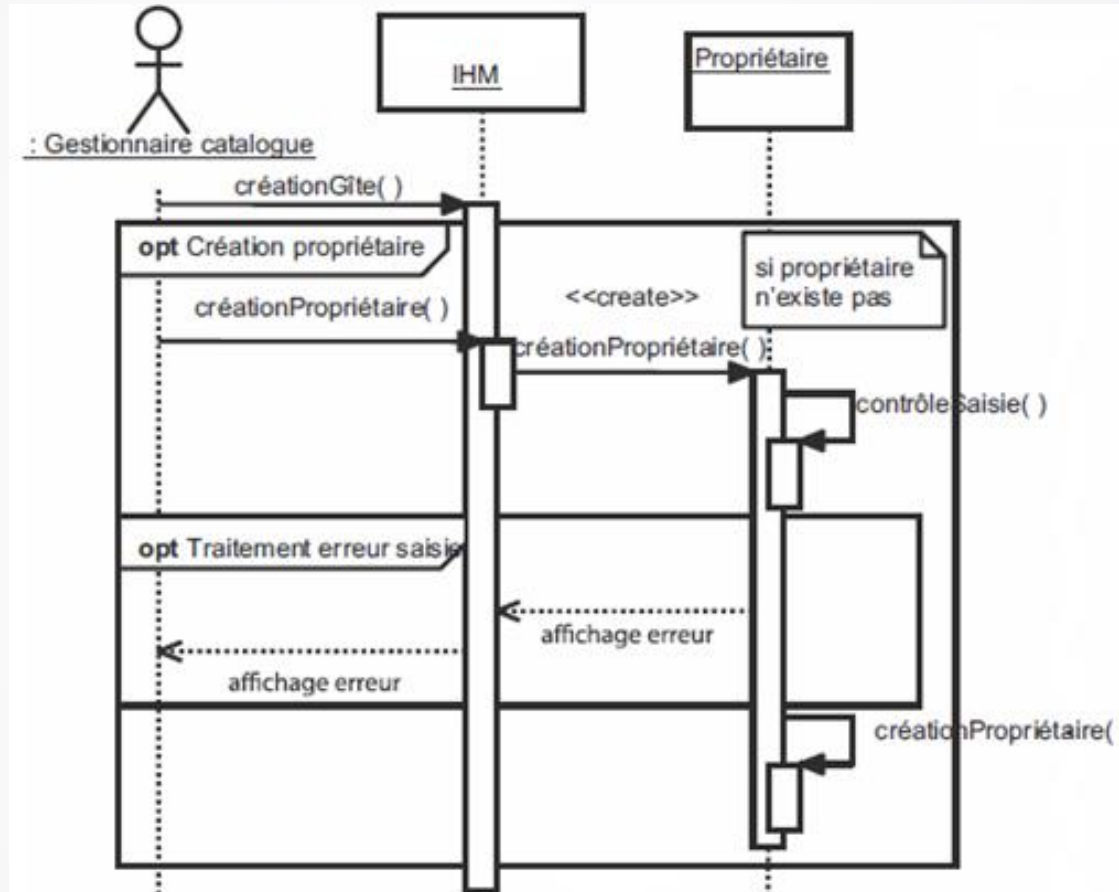
- Différentes possibilités
 - 2 couches :
 - Couche présentation + métier
 - Couche persistance
 - 2 couches :
 - Couche présentation
 - Couche persistance + métier
 - Exemple : les classes entités en plus des accesseurs contiennent les méthodes CRUD
 - 2 couches :
 - Couche présentation + métier
 - Couche persistance + métier

► Modèle en 2 couches



- Inconvénients :
 - Réutilisation des méthodes impossibles :
 - Il faut faire des copier-coller du même code dans plusieurs interfaces
 - Maintenance difficile :
 - Tout le code métier est situé au niveau des interfaces graphiques → pas de lisibilité
 - Mise à jour de toutes les copies d'une méthode

► Exemple



Source : Livre UML2 analyse et conception

- Exemple implémentant le modèle en 2 couches :
 - La couche présentation
 - La couche persistance et métier



Diagramme d'états-transitions

Statechart diagram (en)



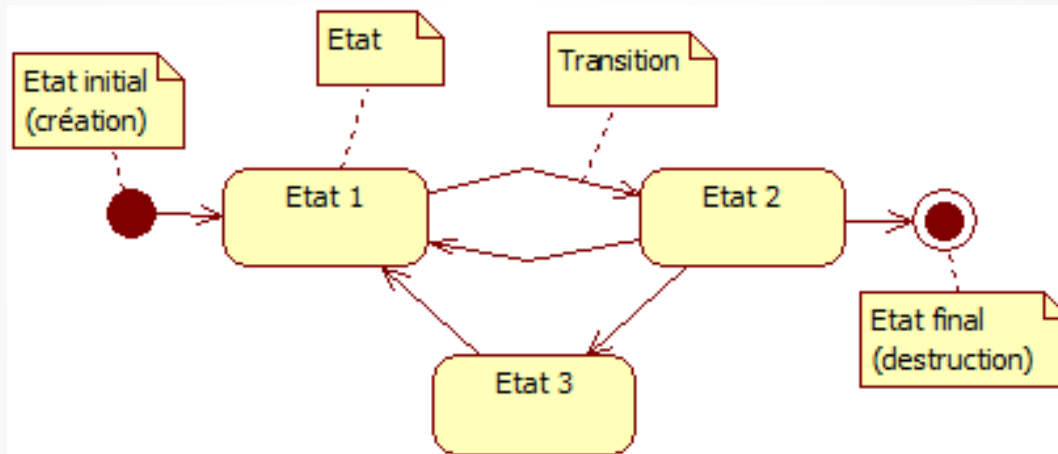
Présentation

- Diagramme comportemental
- Utilisé pour les classes qui ont un comportement complexe
- Est associé à une instance d'une classe (objet) possédant *un* ou *plusieurs* états
- Décrit comment un objet réagit à des événements en fonction de son état courant et comment il passe à un nouvel état

► Représentation



- Graphe orienté d'états et de transitions représentant un automate à états finis





Concepts clés

- État
 - Activité
- Transition
 - Évènement
 - Condition de garde
 - Action
- État composite



État

► État



- Rôle :
 - Représente l'état de l'objet sous certaines conditions
- Représentation graphique: Etat
- Exemples :
 - États d'un objet de la candidature :

Envoyée

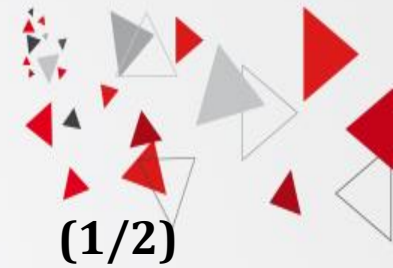
Acceptée

Refusée



Activité

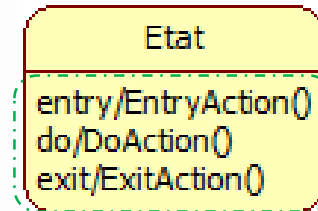
▶ Activité



(1/2)

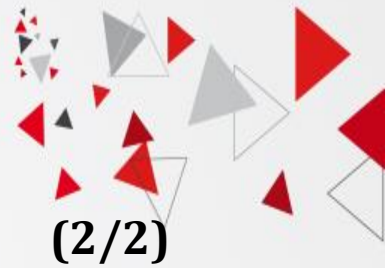
- Rôle :
 - Spécifie un comportement optionnel de l'objet lorsqu'il atteint un nouvel état (après le déclenchement d'une transition)

- Représentation graphique :

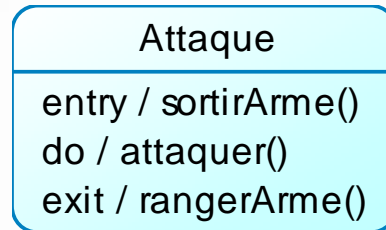


- Types d'activités :
 - **entry** : activités d'entrée
 - Actions effectuées au moment de l'entrée dans un état
 - **do** : activités durable
 - Indique un travail effectué tant que l'objet est dans l'état
 - **exit** : activités de sortie
 - Actions effectuées au moment de la sortie d'un état

► Activité



- Exemple:
 - Un des états de la classe « *Personnage* » d'un jeu vidéo

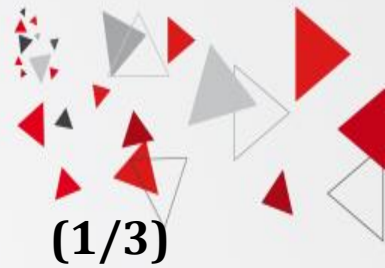


- Lorsque le personnage **atteint** l'état **Attaque** il va :
 1. Exécuter une seule fois **sortirArme()**
 2. Exécuter en boucle **attaquer()**
- Lorsque le personnage **quitte** l'état **Attaque** il y a :
 1. Arrêt de l'exécution en boucle de la fonction **attaquer()**
 2. Exécution une seule fois **rangerArme()**



Transition

Transition

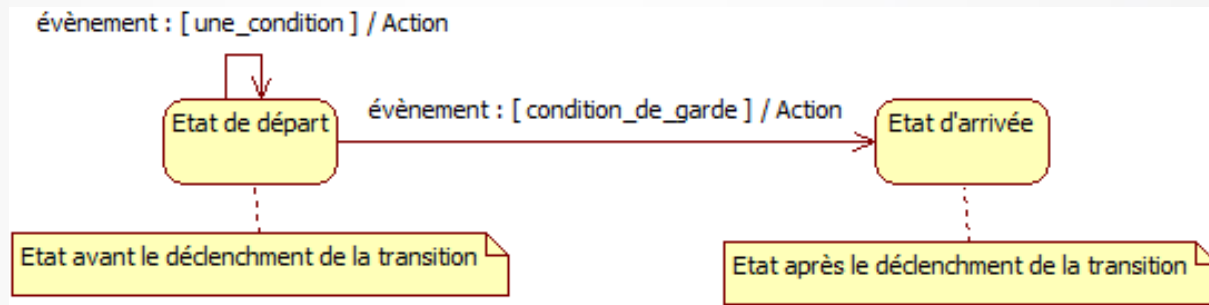


- Permet le passage d'un état à un autre
 - Peut être réflexive
- Décrit la *réaction* d'un objet lorsqu'un événement se produit
- Une transition peut avoir :
 - Un événement déclencheur
 - Une condition de garde :
 - A mettre après l'évènement et entre **[]**
 - Une action (méthode associée à la transaction) :
 - A mettre après l'évènement et/ou la condition et précédée d'un **/**
 - Un état cible

Transition

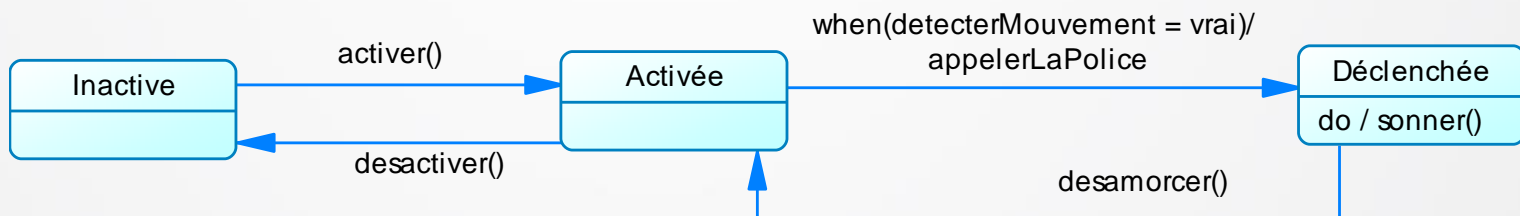
(2/3)

- Représentation :

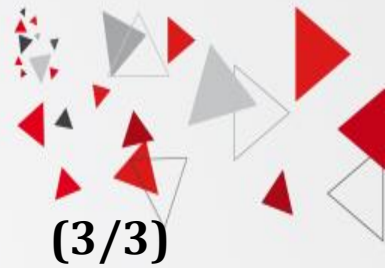


- Exemple 1 :

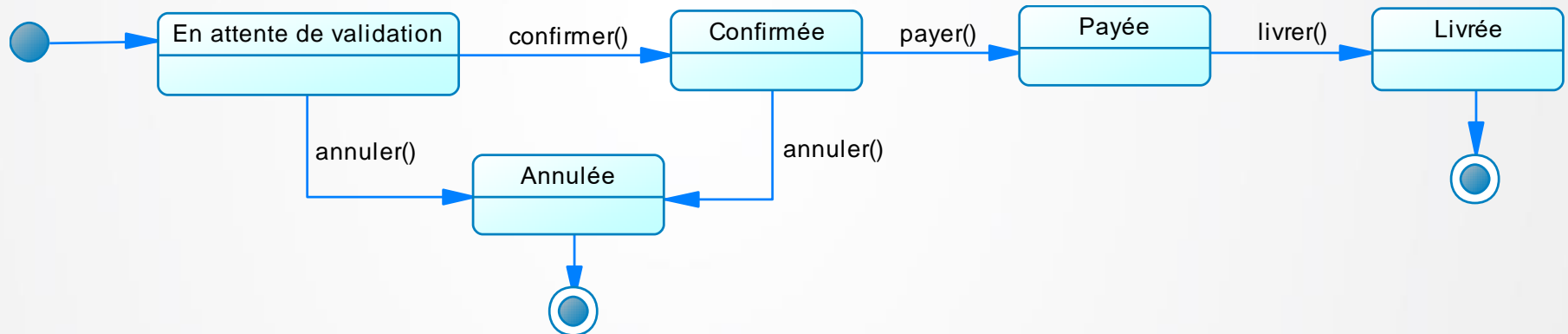
- États de la classe « *Alarme* »



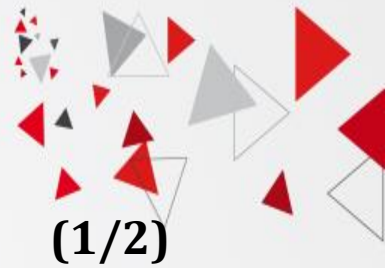
► Transition



- Exemple 2 :
 - États de la classe « *Commande* »

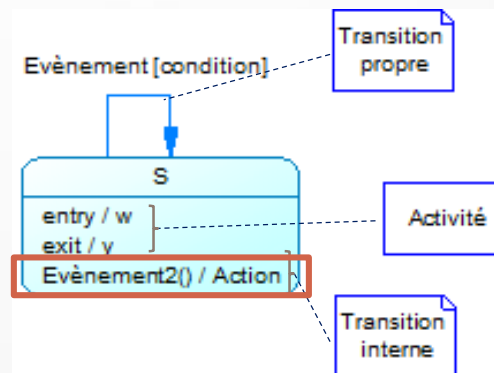


► Transition interne

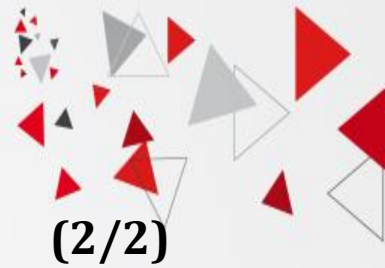


(1/2)

- Rôle:
 - Transition qui se déclenche dans un état courant
 - La transition a un état source mais pas d'état cible
 - L'objet ne quitte pas l'état courant
 - Est inscrite dans l'état
- Représentation graphique :



► Transition interne



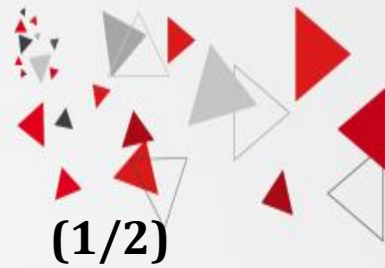
(2/2)

- Exemple :

Attaque
entry / sortirArme() do / attaquer() exit / rangerArme() recevoirCoup()[nbVieRestant>=1]() / esquiver()

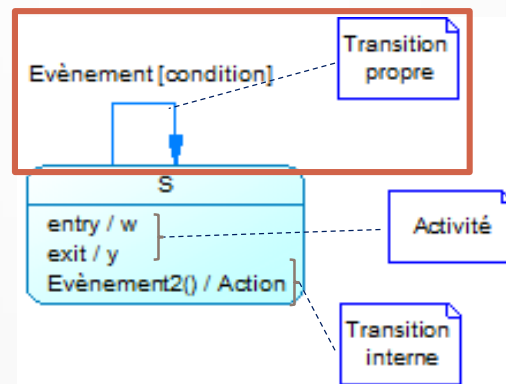
- Si on est dans l'état *Attaque* et qu'il y a *réception d'un coup* alors :
 1. Interruption de la méthode ***attaquer()***
 2. Exécution de la méthode ***recevoirCoup()***
 3. Invocation de la méthode ***esquiver()***
 4. Reprise de la méthode ***attaquer()***

► Transition réflexive

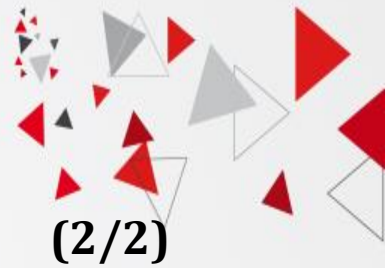


(1/2)

- *Transition réflexive* ou *transition propre*
 - Rôle :
 - L'état de départ et l'état cible sont identiques
 - L'objet quitte un état pour y revenir
 - Représentation graphique :

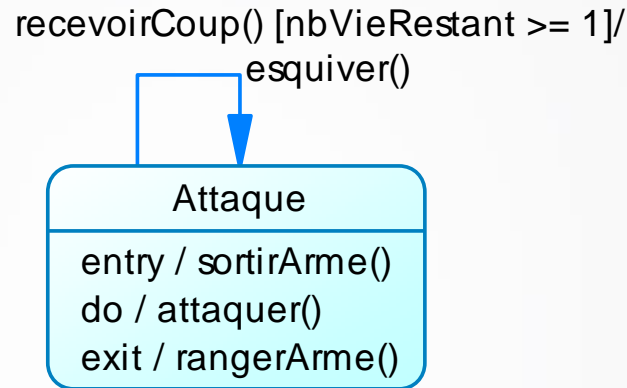


► Transition réflexive



(2/2)

- Exemple :

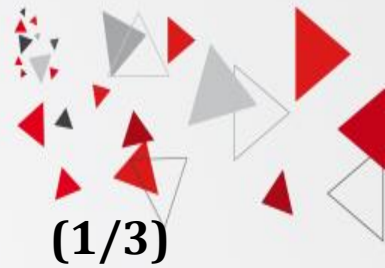


- Si on est dans l'état *Attaque* et qu'il y a et qu'il y a *réception d'un coup* alors il y a:
 1. Interruption de la méthode **attaquer()**
 2. Appel à **rangerArme()** : sortie de l'état *Attaque* → exécution du *exit*
 3. Exécution de la méthode **recevoirCoup()**
 4. Appel à **esquiver()** : l'action de la transition
 5. Appel à **sortirArme()** : entrée dans l'état *Attaque* → exécution du *entry*
 6. Appel à **Attaquer()** : activité durable de l'état *Attaque*



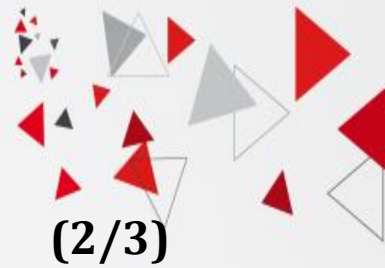
Évènement

Evènement



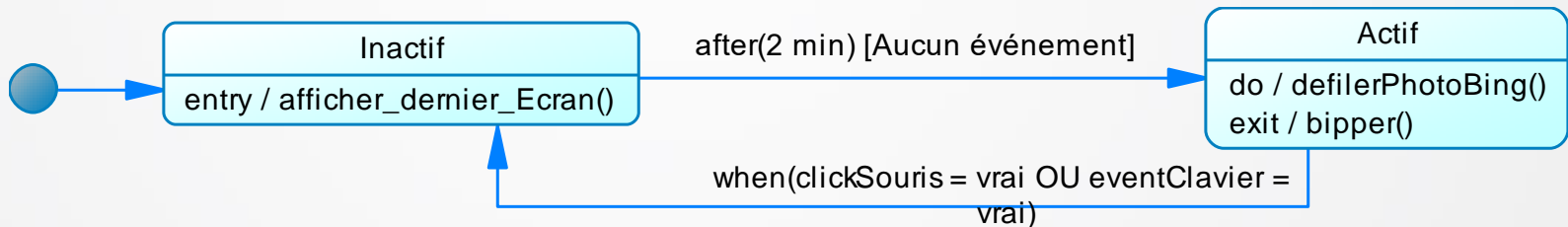
- Se produit à un instant donné et est instantané
- Déclenche une transition
- Types d'événements :
 - Type appel de méthode (call)
 - Type signal
 - Exemple : clic de souris, interruption d'entrées-sorties
 - Type changement de valeur (vrai/faux) :
 - Evaluation d'une expression booléenne :
 - **when** (condition_booléenne)

▶ Evènement



(2/3)

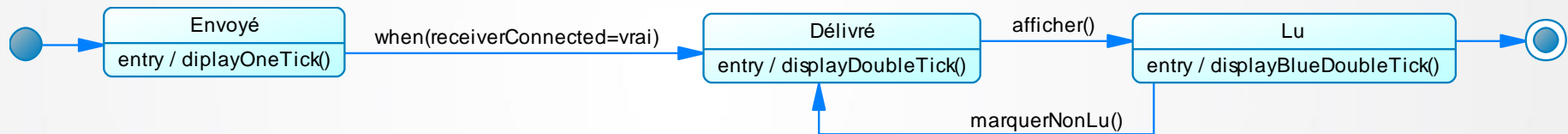
- Types d'événements (suite) :
 - Type temporel : événement lié à l'écoulement du temps
 - Après une durée précise:
 - **after** (durée)
 - A une date précise:
 - **when** (date)
- Exemple 1 :
 - États de la classe « *EcranVeille* » : Après deux minutes d'inactivité, l'écran de veille sera activé



► Evènement

(3/3)

- Exemple 2 :
 - États de la classe « *Message* » de l'application *whatNew*



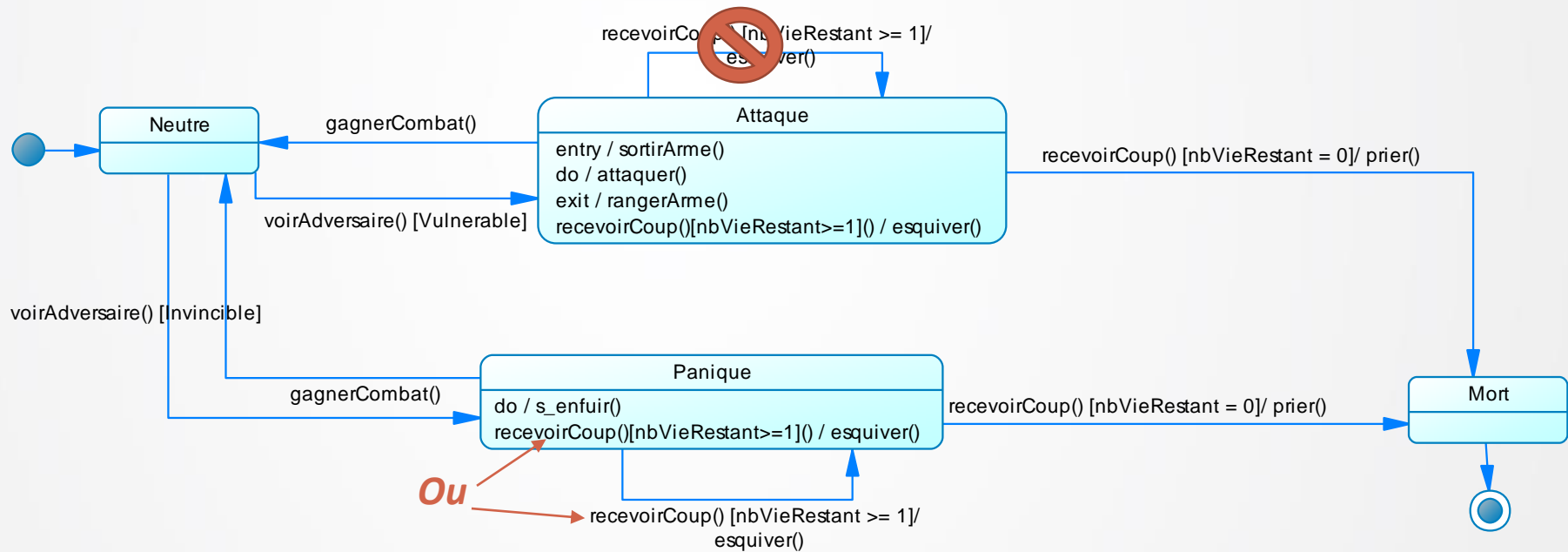


Exemple récapitulatif

► Exemple récapitulatif



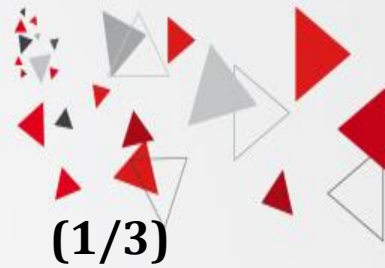
- Modélisation des états d'un personnage de jeu vidéo





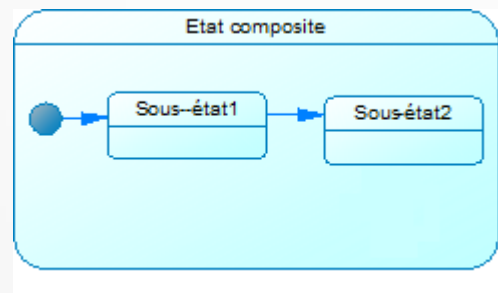
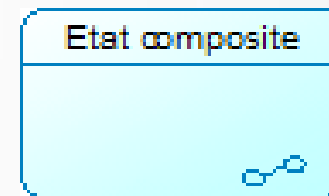
État composite (Super-état)

► Etat composite ou Super-état



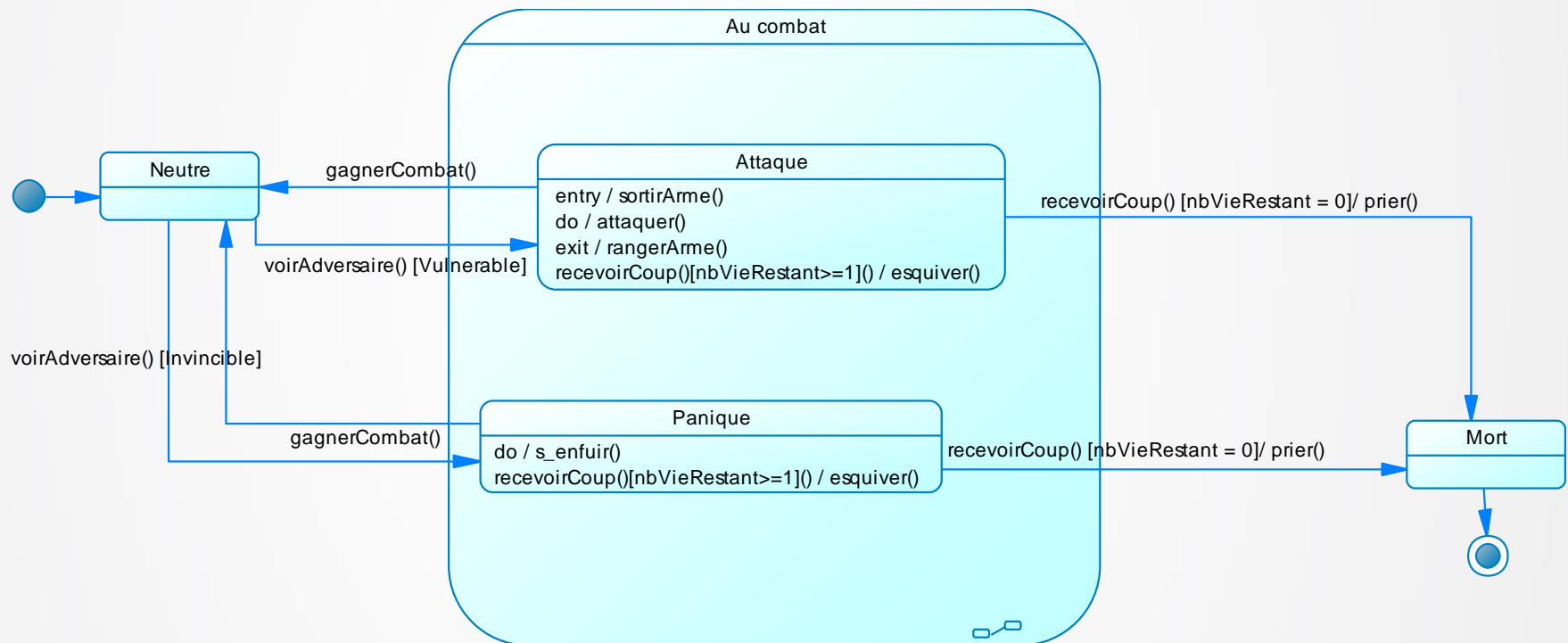
(1/3)

- Rôle :
 - Super-état qui regroupe des sous-états
 - Optionnel
- Représentation graphique :
 - Forme 1 : Décomposition cachée
 - Forme 2 : Décomposition explicite



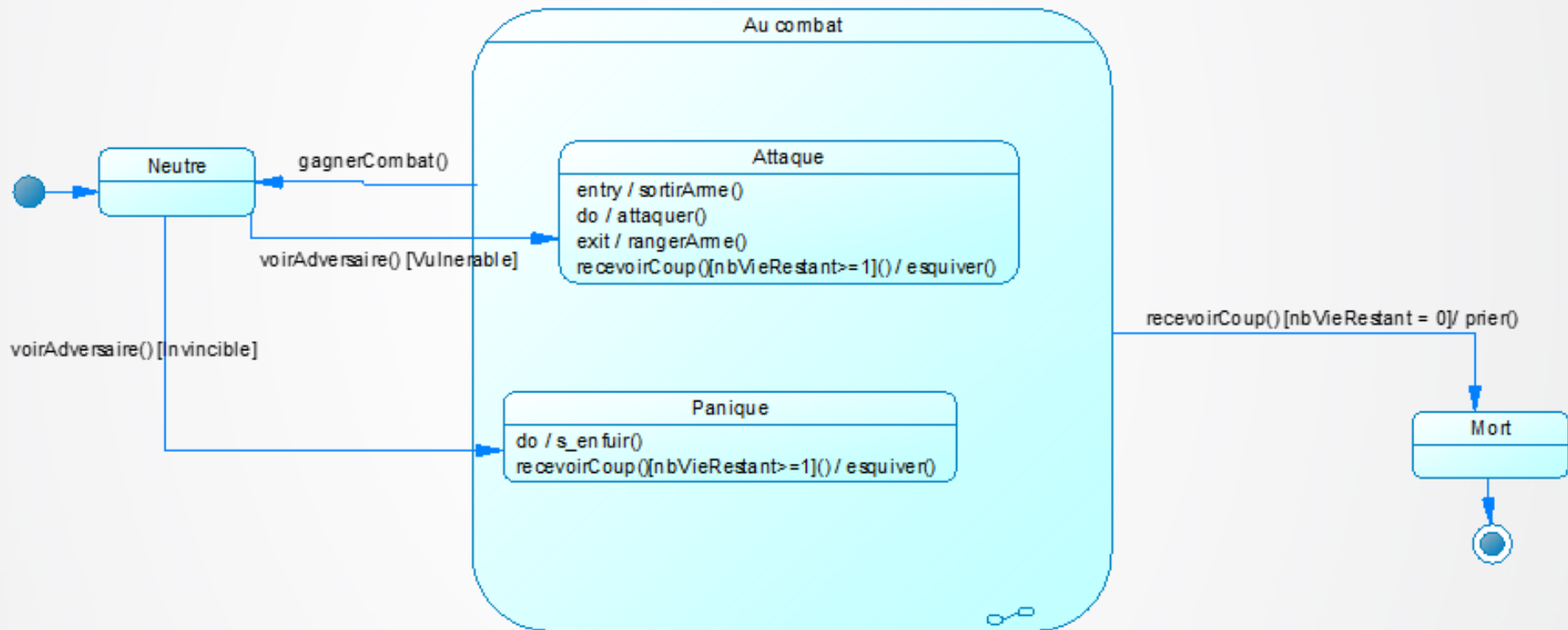
Exemple 1

(2/3)



► Exemple 1 modifié

(3/3)



► Des questions?

