Bases de la conception orientée objet
Diagrammes de séquence
ACDA 2.1

Luc Dartois

luc.dartois@u-pec.fr

Objectifs

Objectifs du cours d'aujourd'hui

- Savoir construire et interpréter des Synopsis
- Maîtriser les notions de communication et temps dans un diagramme séquence
- Savoir construire des diagrammes séquence pertinents

Décrire un modèle par l'exemple

Les besoins que doivent remplir un programme/une application s'expriment en terme d'usages et de fonctionnalités.
On peut donc décrire (partiellement) le fonctionnement d'un programme par les usages que l'on va en faire.

Décrire un modèle par l'exemple

Les besoins que doivent remplir un programme/une application s'expriment en terme d'usages et de fonctionnalités.
On peut donc décrire (partiellement) le fonctionnement d'un programme par les usages que l'on va en faire.
L'utilisateur effectue une action telle que :

- Je lance l'application
- Je clique sur tel bouton,
- J'entre telles données à tel endroit

Et une réponse précise est attendue.

Décrire un modèle par l'exemple

Les besoins que doivent remplir un programme/une application s'expriment en terme d'usages et de fonctionnalités.
On peut donc décrire (partiellement) le fonctionnement d'un programme par les usages que l'on va en faire.
L'utilisateur effectue une action telle que :

- Je lance l'application
- Je clique sur tel bouton,
- J'entre telles données à tel endroit

Et une réponse précise est attendue.
Il n'est généralement pas possible de décrire tout un système de cette façon, mais permet de donner des idées sur le fonctionnement, et donc l'implémentation à donner derrière.

Décrire un modèle par l'exemple

Les besoins que doivent remplir un programme/une application s'expriment en terme d'usages et de fonctionnalités.
On peut donc décrire (partiellement) le fonctionnement d'un programme par les usages que l'on va en faire.
L'utilisateur effectue une action telle que :

- Je lance l'application
- Je clique sur tel bouton,
- J'entre telles données à tel endroit

Et une réponse précise est attendue.
Il n'est généralement pas possible de décrire tout un système de cette façon, mais permet de donner des idées sur le fonctionnement, et donc l'implémentation à donner derrière.
$\rightarrow$ Aujourd'hui : 2 outils pour décrire de tels comportement !

Synopsis

Un *synopsis* est un texte court décrivant une utilisation de votre programme.

- Il est court, et concentré sur quelques (1-5) cas d'usages.

Synopsis

Un *synopsis* est un texte court décrivant une utilisation de votre programme.

- Il est court, et concentré sur quelques (1-5) cas d'usages.
- Il est écrit du point de vue d'un utilisateur.

L'utilisateur est un acteur avec un rôle précis qu'il convient de donner. On parle donc de ce que fait l'utilisateur et quelle est la réponse visible du système !
→ On ne décrit pas la machinerie interne du système invisible à l'utilisateur.

Synopsis

Un *synopsis* est un texte court décrivant une utilisation de votre programme.

- Il est court, et concentré sur quelques (1-5) cas d'usages.
- Il est écrit du point de vue d'un utilisateur.
- Il est précis.

L'utilisateur interagit de façon précise, il clique sur tel bouton, ou lance tel cas d'usage de telle façon.
→ On évite de mélanger les cas d'usages.

Synopsis

Un *synopsis* est un texte court décrivant une utilisation de votre programme.

- Il est court, et concentré sur quelques (1-5) cas d'usages.
- Il est écrit du point de vue d'un utilisateur.
- Il est précis.
- Il peut être écrit par un non informaticien.

Un non informaticien ne connait pas l'implémentation et les objets mis en jeu.
→ On reste naïf sur l'implémentation, qui n'est pas décrite.

Exemple

Système d'authentification sur un site internet

L'utilisateur arrive sur une page avec un bouton lui proposant de se connecter. En cliquant, il reçoit une invite pour entrer son login et son mot de passe. L'utilisateur entre ses informations, mais la procédure d'authentification échoue, le serveur le redirige alors vers une page avec un formulaire lui proposant de s'inscrire.

Exemple

Système d'authentification sur un site internet

L'utilisateur arrive sur une page avec un bouton lui proposant de se connecter. En cliquant, il reçoit une invite pour entrer son login et son mot de passe. L'utilisateur entre ses informations, mais la procédure d'authentification échoue, le serveur le redirige alors vers une page avec un formulaire lui proposant de s'inscrire.

- Quel est le système et quel(s) sont les acteurs ?
- Quels usages le texte suggère-t-il ?
- À quoi ressemble le diagramme de cas d'usage qu'on pourrait lui associer ?

Deuxième exemple

Système d'authentification sur in site internet

Une fois que l'utilisateur a entré ses données, le serveur encrypte le mot de passe et se connecte à la base de données pour vérifier les informations. Si l'authentification fonctionne, on créé un cookie pour stocker la connexion. Si elle échoue, on retourne vers la page de connection.

- En quoi ce synopsis est différent du précédent ?
- Quels sont les soucis avec celui-ci ?

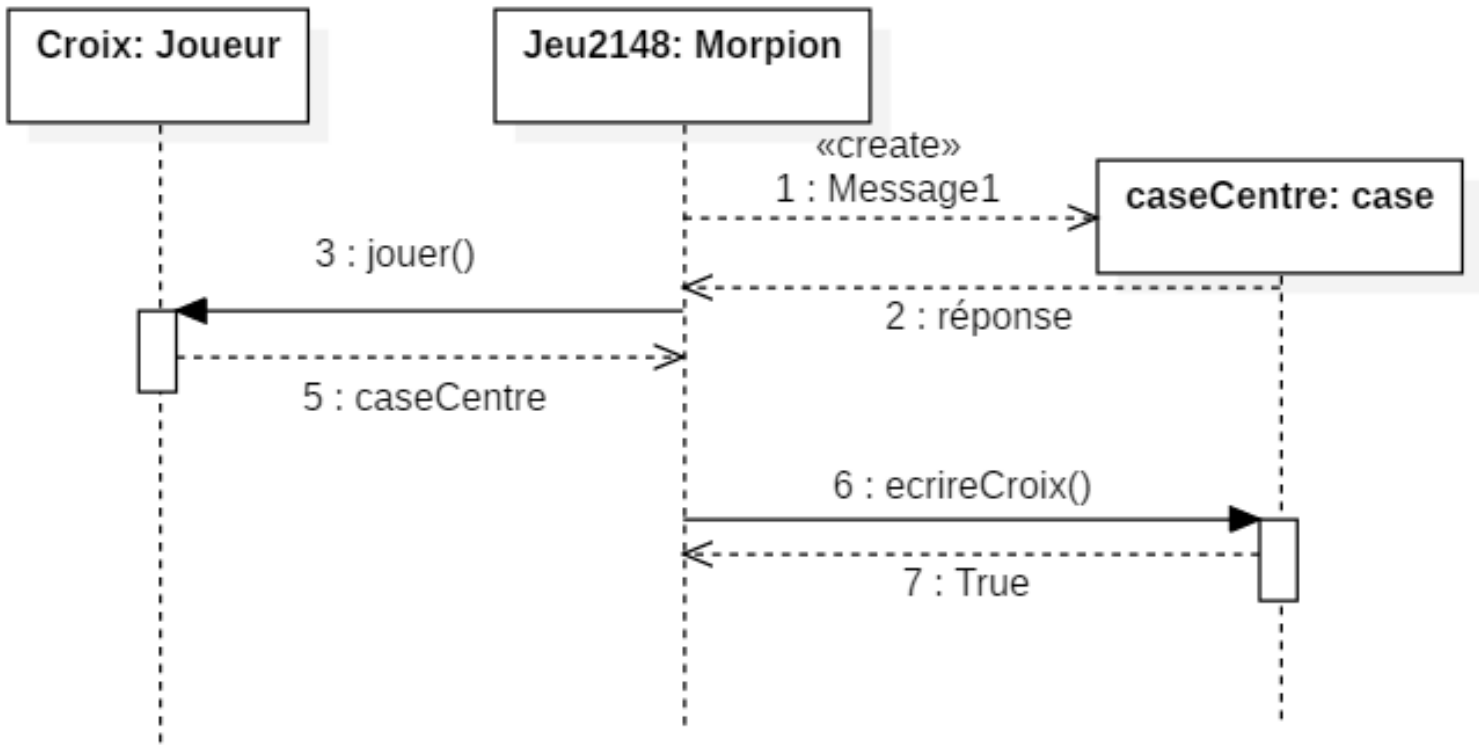
Diagrammes séquence

Les synopsis permettent à une personne extérieure au projet de décrire ses attentes, du point de vue d'un utilisateur.
Les diagrammes séquence permettent de raffiner cette vue, en décrivant les *messages* entre des objets de notre programme **lors d'une utilisation**.

Diagrammes séquence

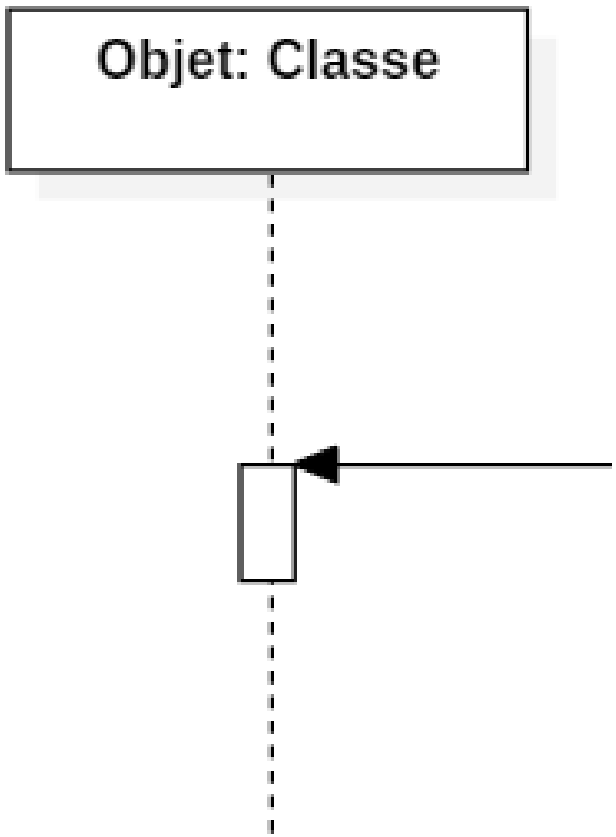
Les synopsis permettent à une personne extérieure au projet de décrire ses attentes, du point de vue d'un utilisateur.
Les diagrammes séquence permettent de raffiner cette vue, en décrivant les *messages* entre des objets de notre programme **lors d'une utilisation**.
Les *messages* prennent la forme de création/destruction d'objets, et d'appels d'opérations.

Example de diagramme séquence



Des interactions entre des *objets*

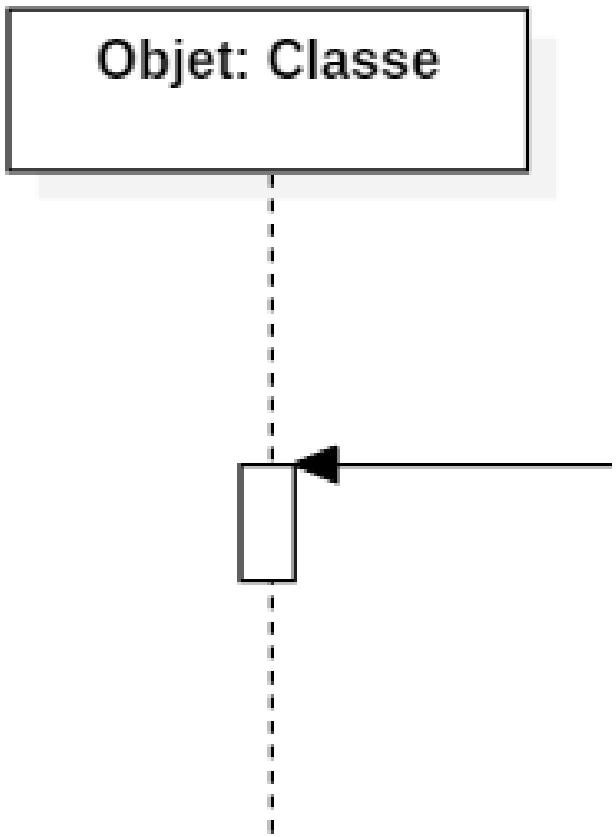
Les diagrammes séquence font intervenir des *objets*. Il s'agit de leurs collaborations dans une *instance* du programme.
On décrit des certains usages, et pas le programme en entier.



Un *objet* dans un diagramme séquence est un élément de notre système. Il peut s'agir d'une instance de classe du programme, ou d'une instance d'un *acteur* d'un diagramme de cas d'usage.

Des interactions entre des *objets*

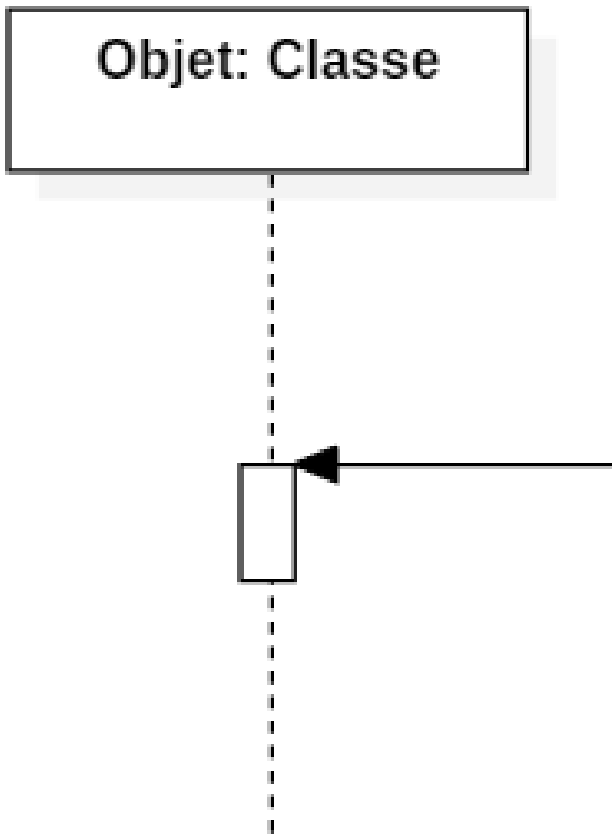
Les diagrammes séquence font intervenir des *objets*. Il s'agit de leurs collaborations dans une *instance* du programme.
On décrit des certains usages, et pas le programme en entier.



Un objet est caractérisé par un nom (ici Objet) et sa classe (Classe), séparés par deux points. C'est la même notation que dans un diagramme objet !

Des interactions entre des *objets*

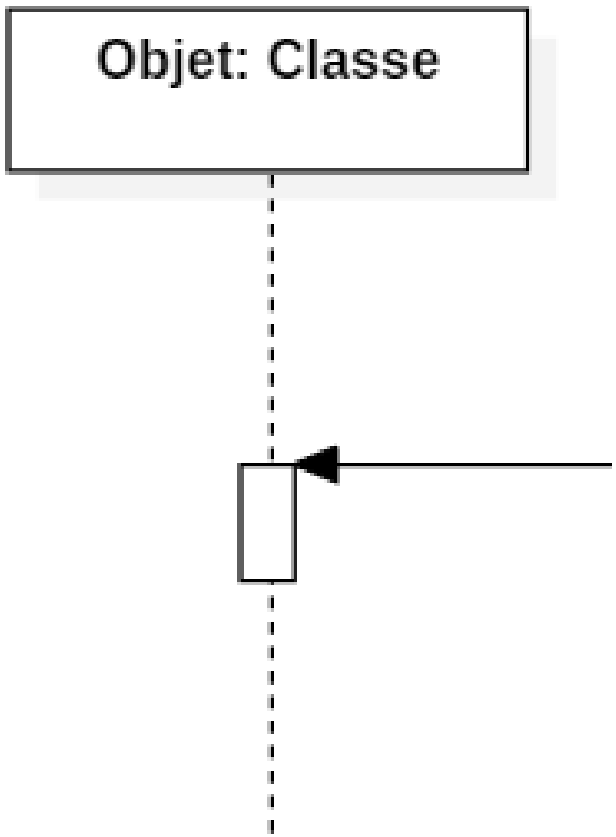
Les diagrammes séquence font intervenir des *objets*. Il s'agit de leurs collaborations dans une *instance* du programme.
On décrit des certains usages, et pas le programme en entier.



L'objet peut ne pas être typé !
→ Si l'objet est un acteur au sens des Diagramme de cas d'usage, càd un élément extérieur au système.

Des interactions entre des *objets*

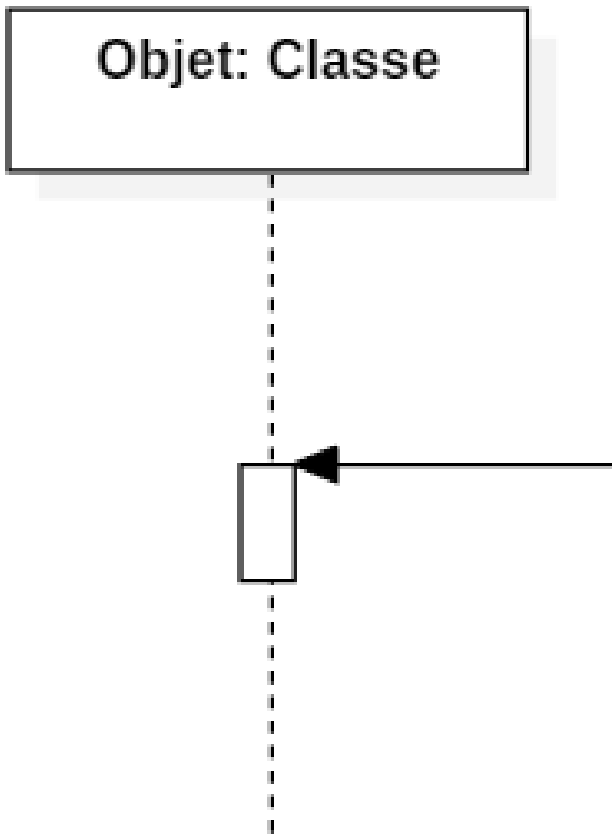
Les diagrammes séquence font intervenir des *objets*. Il s'agit de leurs collaborations dans une *instance* du programme.
On décrit des certains usages, et pas le programme en entier.



L'objet peut ne pas avoir de nom !
$\rightarrow$ Si l'objet n'est utilisé qu'une fois et n'est donc pas référencé plus tard.

Des interactions entre des *objets*

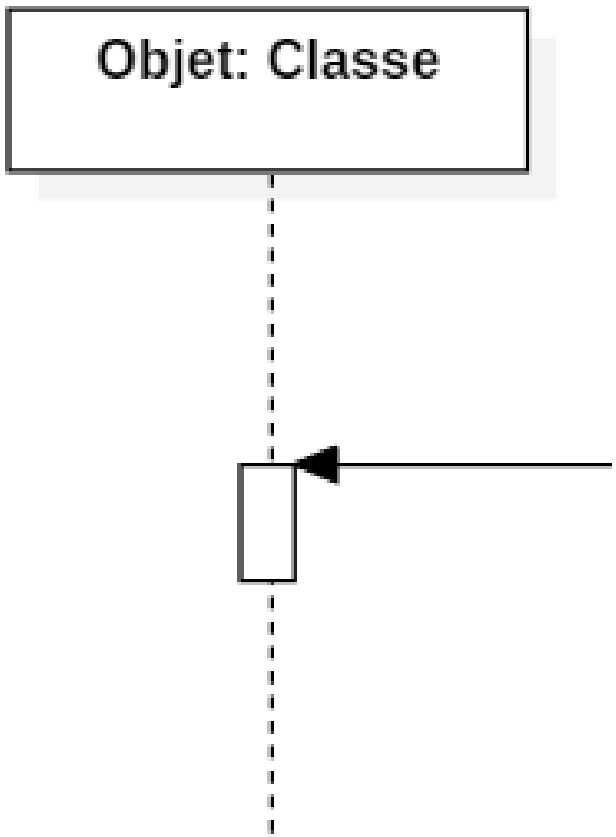
Les diagrammes séquence font intervenir des *objets*. Il s'agit de leurs collaborations dans une *instance* du programme.
On décrit des certains usages, et pas le programme en entier.



La ligne de pointillé s'appelle la *ligne de vie*, et représente verticalement l'axe du temps pour l'objet.

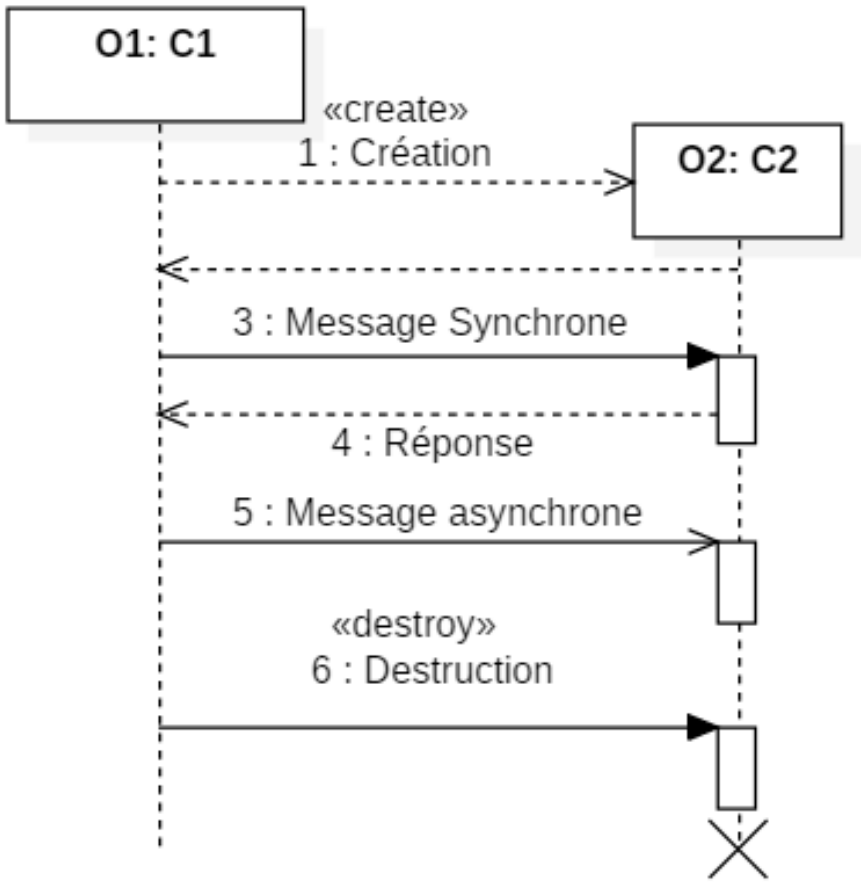
Des interactions entre des *objets*

Les diagrammes séquence font intervenir des *objets*. Il s'agit de leurs collaborations dans une *instance* du programme.
On décrit des certains usages, et pas le programme en entier.



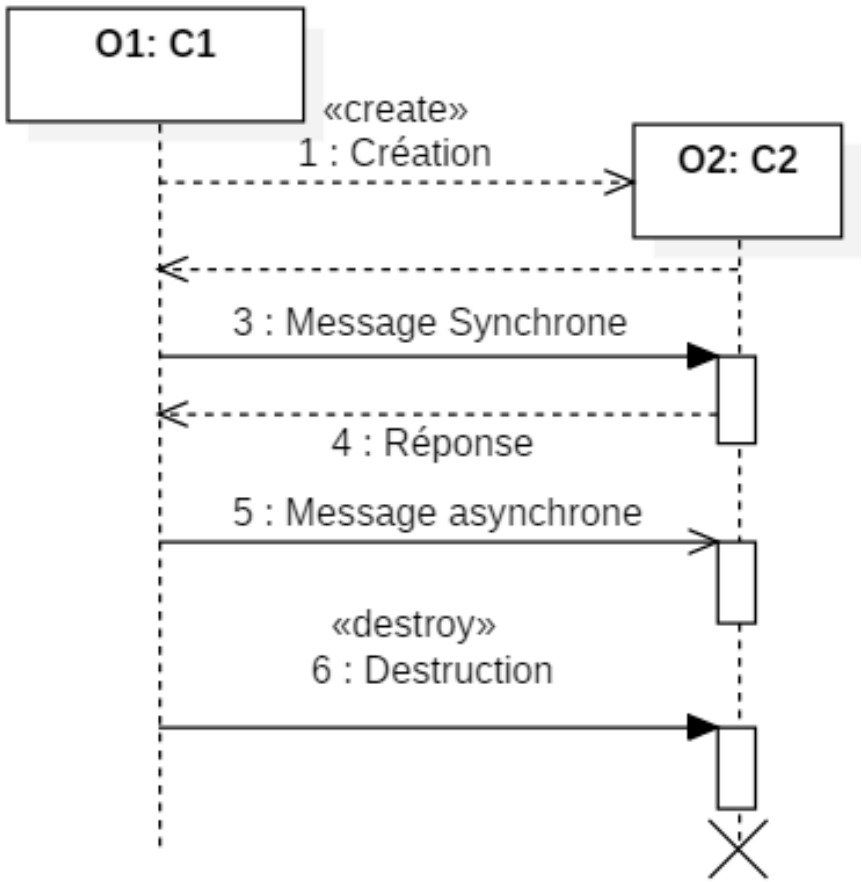
Les rectangles sont des *périodes d'activités*, il s'agit du temps pendant lesquels les objets effectuent des actions demandées par d'autres objets.

Des messages entre objets



Les interactions entre objets sont représentées par des *messages*, càd des flèches dans les lignes de vie des objets. Il s'agit le plus souvent d'appels d'opérations.

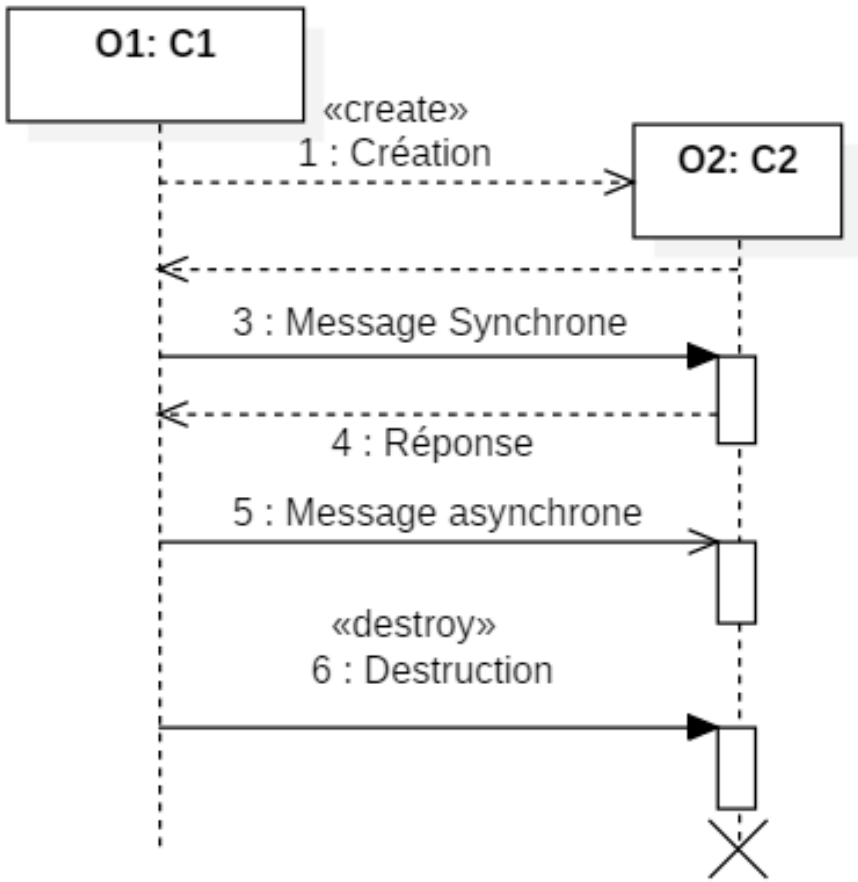
Des messages entre objets



Chaque message représente **deux** événements !

- l'*envoi* qui est le début de la flèche,
- la *réception* qui est l'extrémité de la flèche.

Des messages entre objets



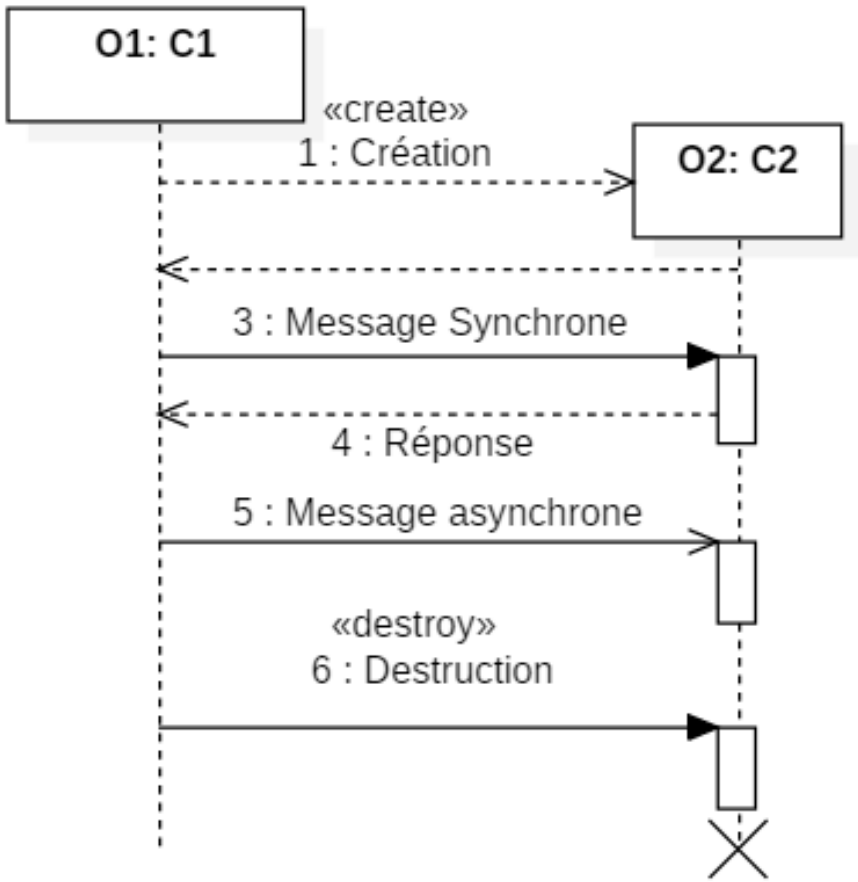
Les messages de création

Un objet appelle le créateur d'une classe pour générer un nouvel objet. En java, cela donne :

```
C2 O2= new C2();
```

dans le code d'une opération lancée par O1. Elle attend ici une réponse, qui sera un pointeur vers l'objet.

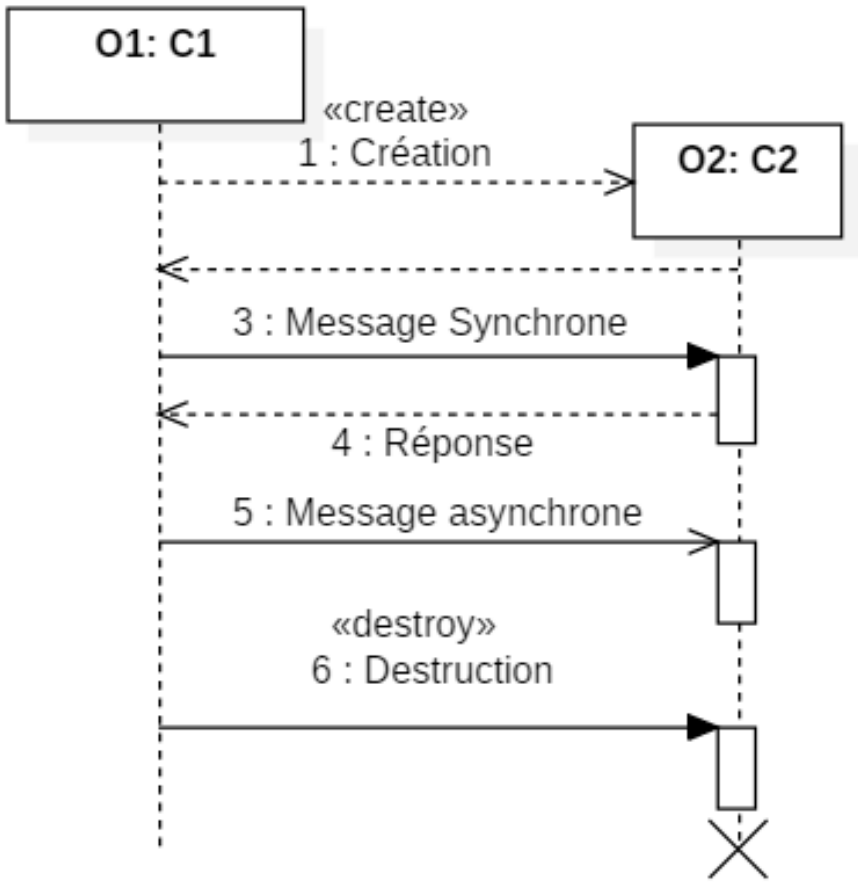
Des messages entre objets



Les messages *synchrones*

L'objet O1 appelle une opération de C2. Elle attend une réponse, càd un return de la part de l'opération appelée.
$\rightarrow$ O1 attend la réponse avant de continuer son exécution !

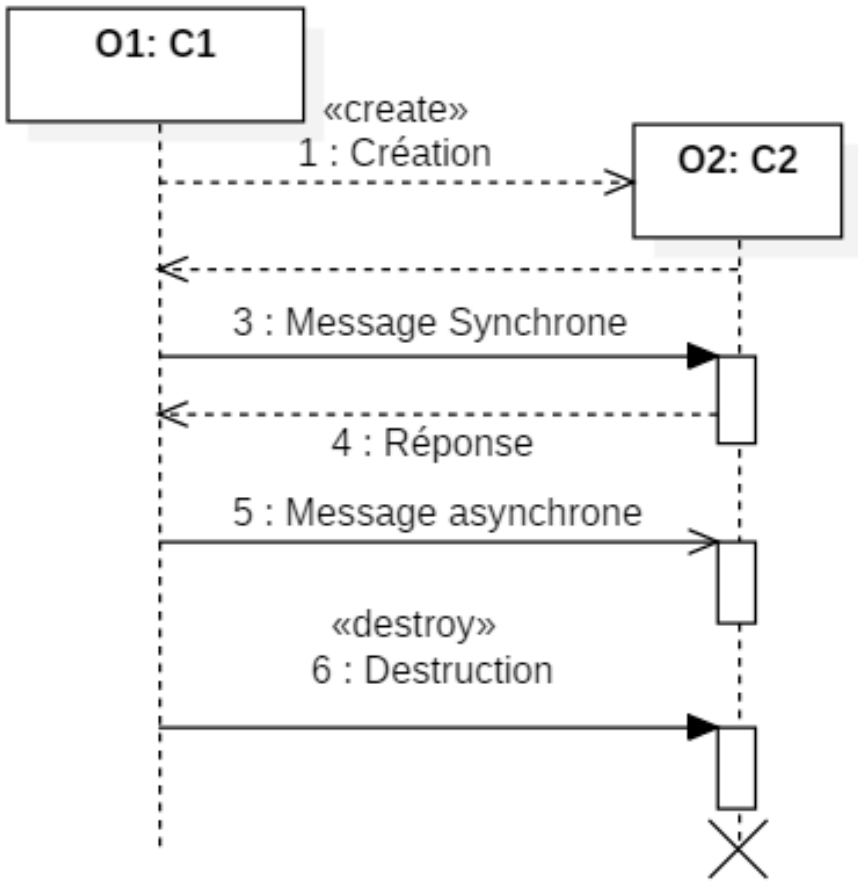
Des messages entre objets



Les messages *asynchrones*

L'objet O1 lance une opération chez O2, mais n'attend pas la réponse pour continuer son exécution.
→ Il peut s'agir du lancement d'un thread par exemple, tel que l'écoute sur un port.

Des messages entre objets



Les messages de *destruction*

L'objet O1 supprime l'objet O2, et libère l'espace mémoire utilisé.
→ Dans le cas de Java, cela est fait par le *garbage collector* et donc non géré par le programmeur. Nous l'utiliserons donc comme outil de modélisation signalant qu'un objet n'est plus utile.

Le temps et ses règles

La ligne de vie représente le passage du temps pour chaque objet. Plus précisément, il s'agit de l'ordre d'exécution des périodes d'activités.

Le temps et ses règles

La ligne de vie représente le passage du temps pour chaque objet. Plus précisément, il s'agit de l'ordre d'exécution des périodes d'activités.
Le temps se lit selon les deux règles suivantes :

- Pour un objet donné, le temps s'écoule du haut vers le bas,
- Pour un message donné, l'envoi s'effectue *avant* la réception.

Le temps et ses règles

La ligne de vie représente le passage du temps pour chaque objet. Plus précisément, il s'agit de l'ordre d'exécution des périodes d'activités.
Le temps se lit selon les deux règles suivantes :

- Pour un objet donné, le temps s'écoule du haut vers le bas,
- Pour un message donné, l'envoi s'effectue *avant* la réception.

Un ordre d'exécution des messages peut être dérivé de ces règles, pour tout diagramme de séquence.

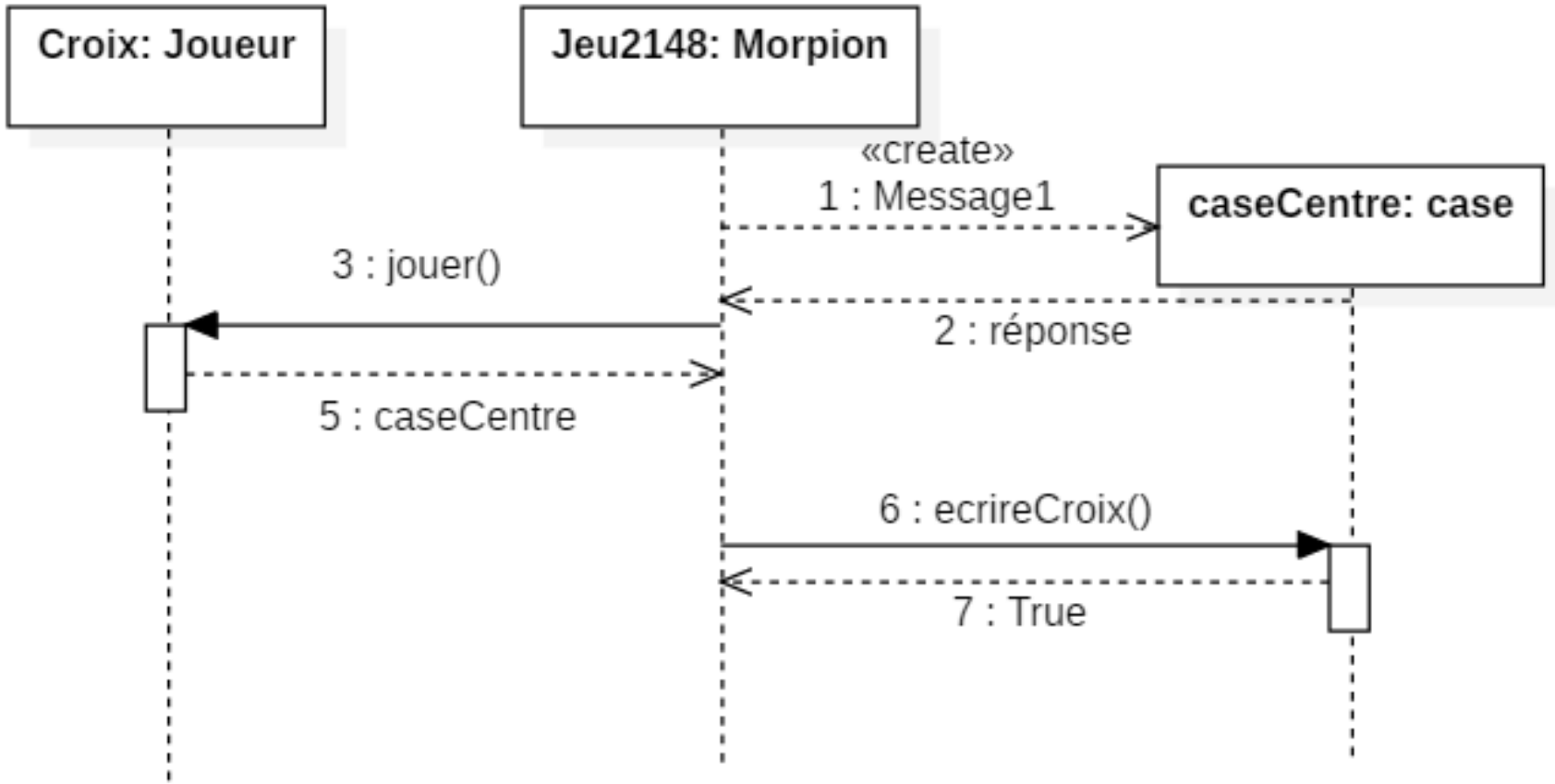
Le temps et ses règles

La ligne de vie représente le passage du temps pour chaque objet. Plus précisément, il s'agit de l'ordre d'exécution des périodes d'activités.
Le temps se lit selon les deux règles suivantes :

- Pour un objet donné, le temps s'écoule du haut vers le bas,
- Pour un message donné, l'envoi s'effectue *avant* la réception.

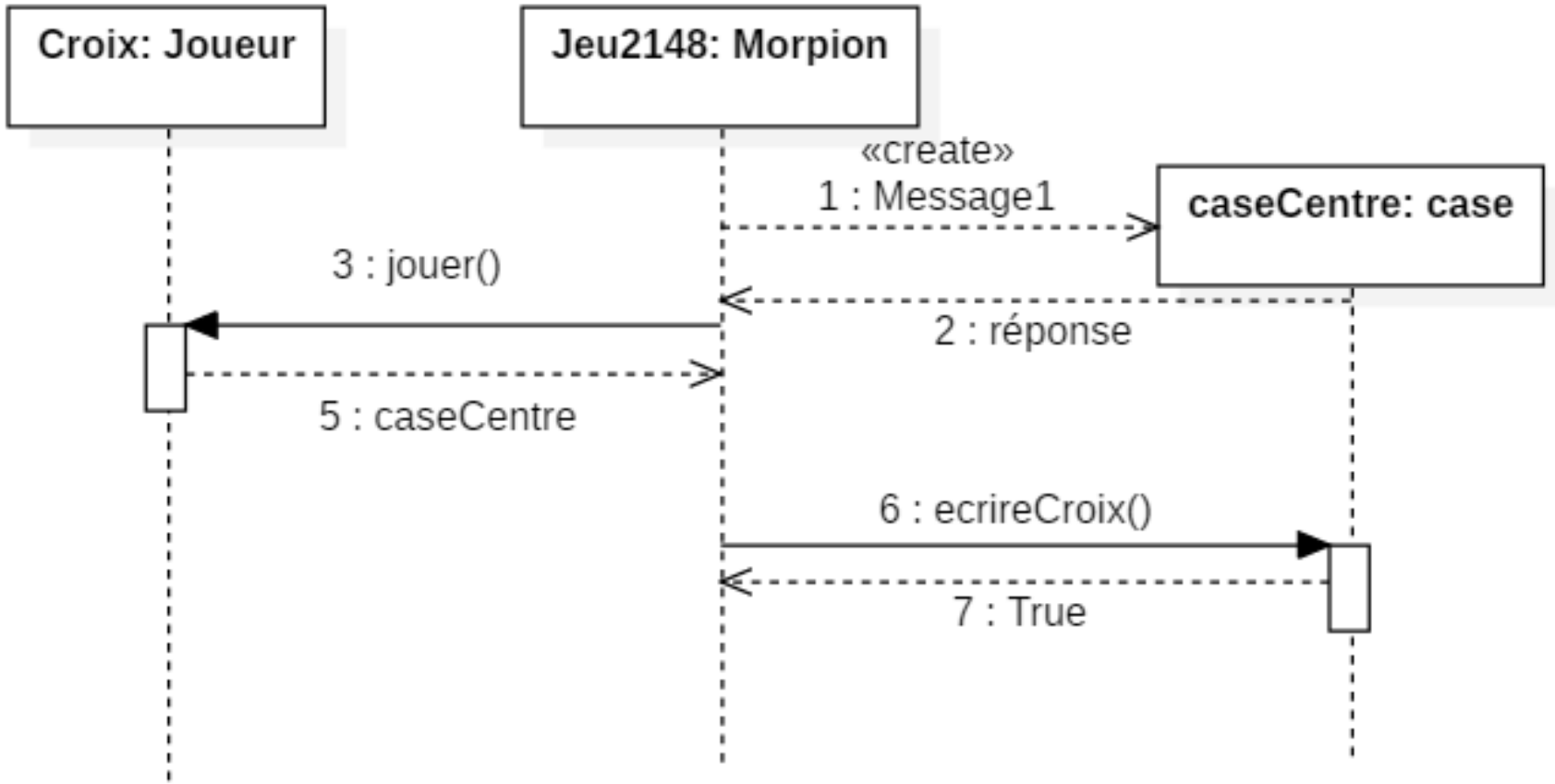
Un ordre d'exécution des messages peut être dérivé de ces règles, pour tout diagramme de séquence.
Question : Pourquoi *un* ordre ?

Des événements dépendants et indépendants



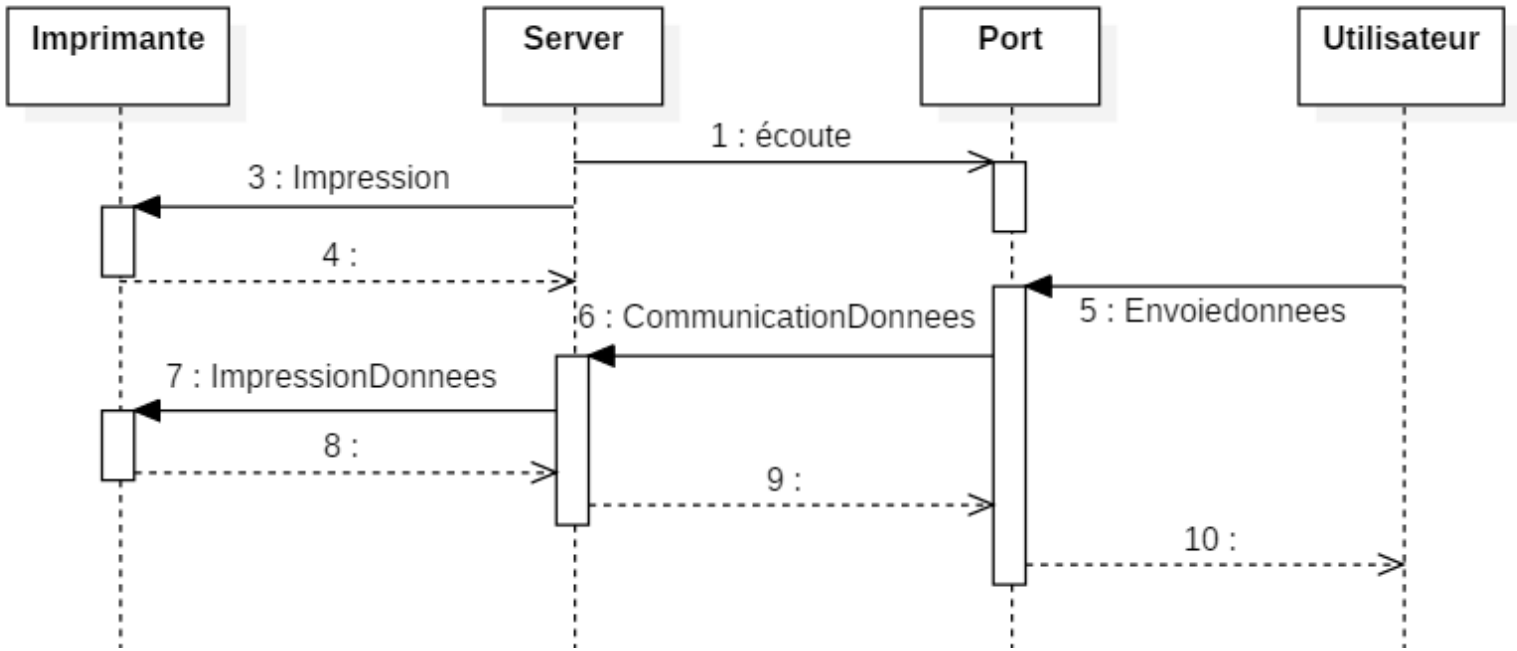
Ici, la ligne de vie du Jeu2148 nous indique que l'appel à jouer() se fait avant l'appel ecrireCroix() (règle 1 sur le temps vertical des objets).

Des événements dépendants et indépendants



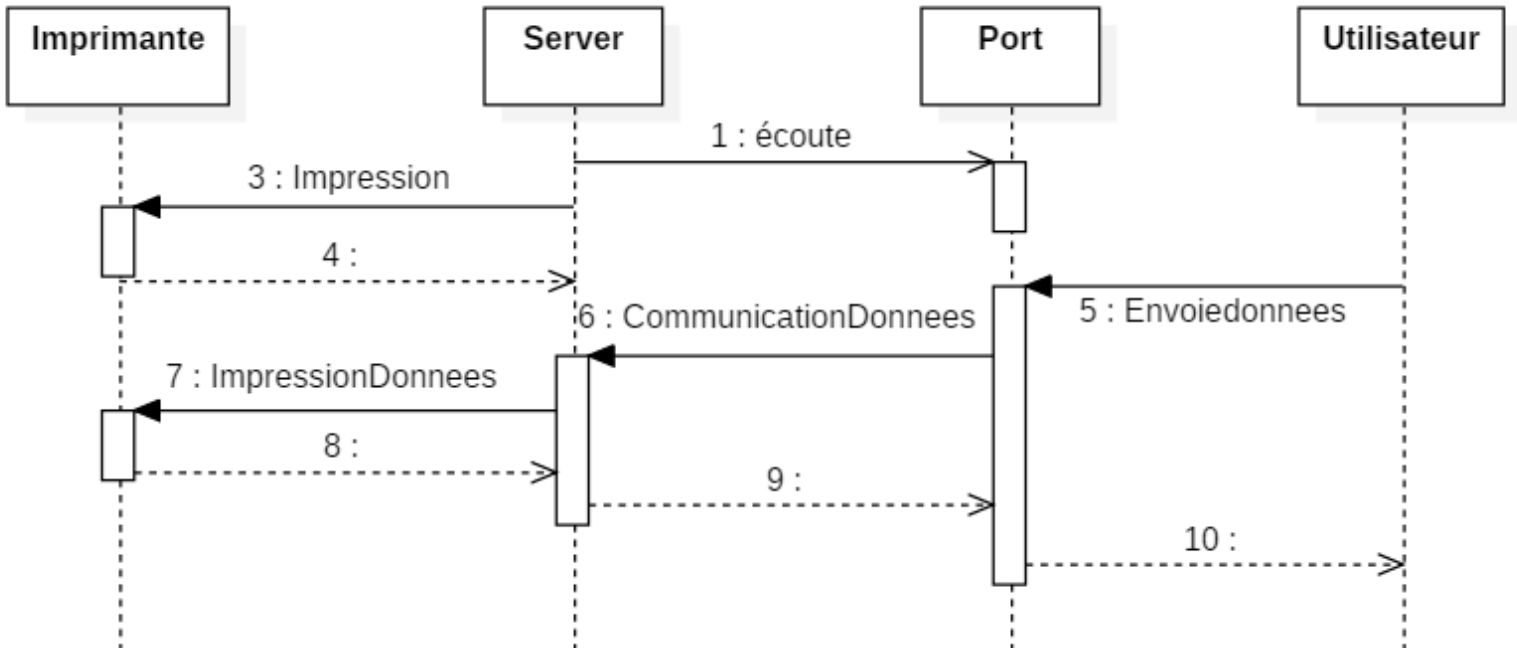
Ici, la ligne de vie du Jeu2148 nous indique que l'appel à jouer() se fait avant l'appel ecrireCroix() (règle 1 sur le temps vertical des objets). De même, l'activité jouer() du Joueur Croix se fait *après* la création de la case caseCentre !

Des événements dépendants et indépendants



- Quel événement arrive en premier entre EnvoiDonnees (5) et ImpressionDonnees (7) ? Pourquoi ?

Des événements dépendants et indépendants



- Quel événement arrive en premier entre EnvoiDonnees (5) et ImpressionDonnees (7) ? Pourquoi ?
- Et entre EnvoiDonnees (5) et Impression (3) ?

Cohérence avec les diagrammes déjà vus

Les différents types de diagrammes que l'on défini sont complémentaires !
→ Ils sont à utiliser ensemble pour décrire différents aspects du système que l'on modélise !

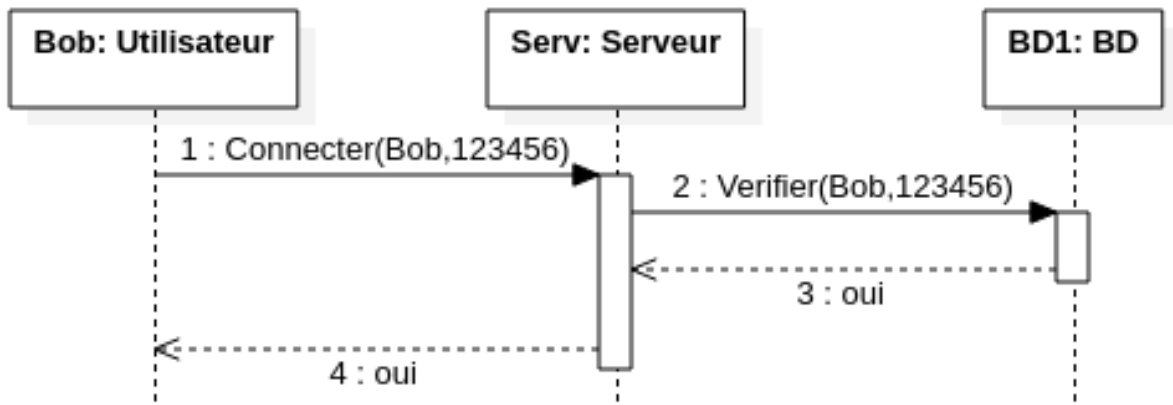
Cohérence avec les diagrammes déjà vus

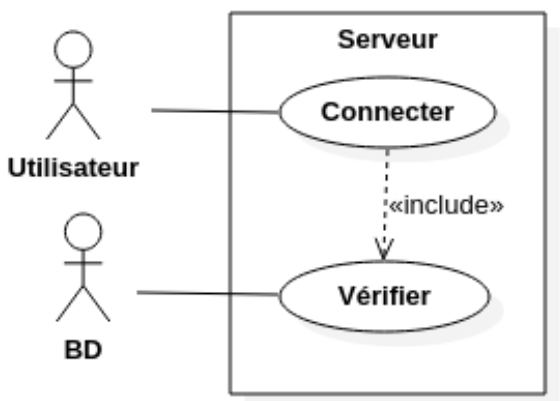
Les différents types de diagrammes que l'on défini sont complémentaires !
→ Ils sont à utiliser ensemble pour décrire différents aspects du système que l'on modélise !
→ Il faut donc veiller à garder une cohérence entre eux !

Cohérence avec les diagrammes déjà vus

Cohérence avec les DCU

- Les messages d'un DS sont étiquetés par des usages du DCU.
- Les messages en eux-même (DS) sont des relations (DCU).
- L'inclusion A vers B (DCU) implique un appel de B pendant A.

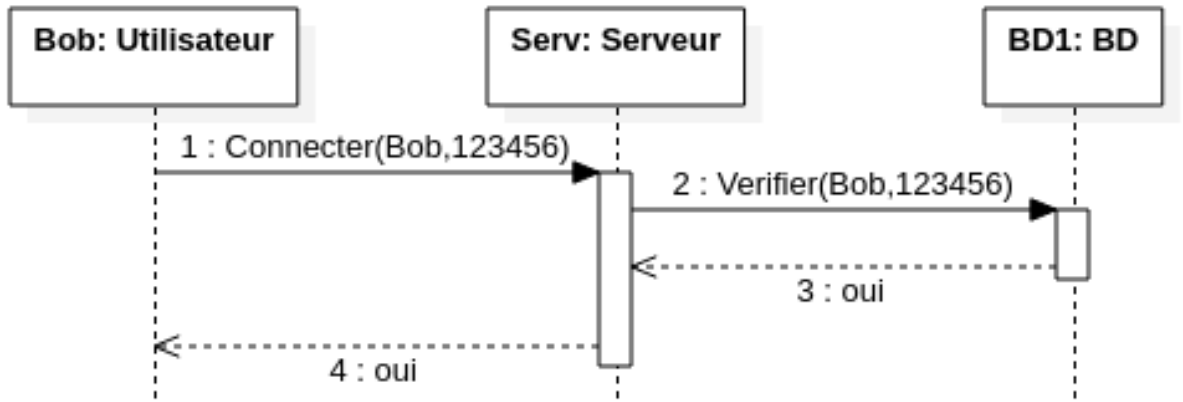


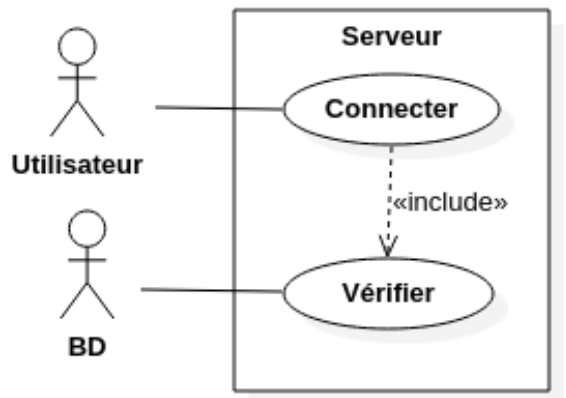


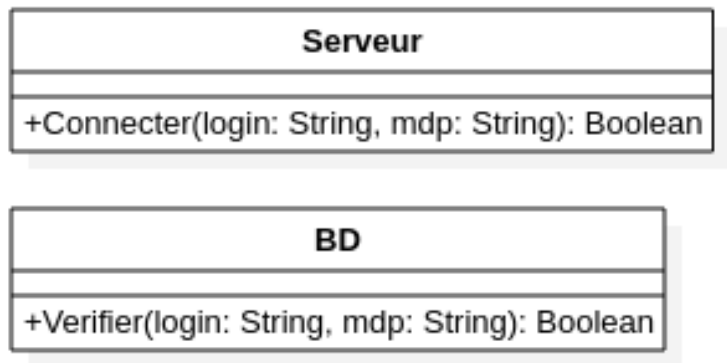
Cohérence avec les diagrammes déjà vus

Cohérence avec les diagrammes de classe

- Un message de A vers B doit être une opération existante chez B
- La signature du message doit correspondre à celle de l'opération déclarée dans le diagramme de classe.







Exercice

On modélise l'utilisation d'une application de messagerie. Les messages envoyés entre deux utilisateurs passent par le serveur centrale de la messagerie. Il est aussi possible d'envoyer des photos qui sont stockées par la messagerie.
Modélisez par un diagramme séquence Bob envoyant un message à Alice, qui lui répond par une photo.

- On identifie les objets interagissant, ainsi que leur type.
- On détermine les interactions entre ces objets (qui, vers où, quel opération)
- On détermine également l'ordre entre ces requêtes.