Outils de développement, programmation événementielle et IHM

Cyrille Bertelle - UFRST Le Havre
25, rue Ph. Lebon - 76058 Le Havre Cedex
Cyrille.Bertelle@univ-lehavre.fr

March 18, 2004

Plan général du cours

1. Introduction
   - objectifs
   - la nécessité d'une méthodologie
   - modèles, langage et outils retenus
2. Le modèle objet avec UML
   - Les raisons d'une méthodologie objet
   - Les différentes descriptions par diagrammes d'UML
   - présentation rapide de BlueJ
3. Java, langage de développement objet
   - Caractéristiques de Java

- Classes et objets
- Types et structures de contrôle
- Exceptions
- Héritage
- Entrée/Sorties
- Les threads

4. Java, graphisme de base : Applets et AWT
5. Les concepts et les modèles généraux pour la construction d'IHM
6. Swing : une implantation du modèle MVC en Java

Chap. 1 : Introduction

1.1 Objectif du cours

- Conception et développement d'applications et d'interfaces homme-machine (IHM).

Pourquoi et comment développer des IHM ?

- Développement de l'informatique et de son utilisation
  - Augmentation du nombre d'utilisateurs aussi bien en utilisation domestique qu'en utilisation professionnelle
  - Avant : utilisation réservée à des spécialistes ... une interface minimale
  - Aujourd'hui : large gamme d'utilisateurs ... nécessite des interfaces ergonomiques
  - Aujourd'hui : multiples échanges de données (par exemple, via Internet) nécessitant des méthodes de manipulation. Un moteur de recherche est une interface.

- Développement du hardware
  - Permet l’utilisation abondante de graphismes et d’interactivités qui deviennent incontournables aux systèmes d’interfaçage.

- Des modèles et des méthodes sont nécessaires.
  - Le graphisme et l'interactivité à outrance ne fait pas tout !
  - On vise à augmenter la facilité et l'efficacité de l'utilisation et NON la distraction (sauf si c'est le but visé) !
  - Besoin de développer des méthodologies
  - Modèle cognitif des utilisateurs
  - Interface "intelligente"
  - Interface multi-modale

Démarche de conception d'une interface :

- Inspirée des techniques utilisées en génie logiciel
- Implémentation dans le cadre de la programmation objet : UML et Java

1.2 La nécessité d'une méthodologie

- Pb : Comment gérer le développement et la maintenance/évolution d'applications informatiques industrielles ?
  - Outil de conception unique pour le développeur, pour le client, pour le spécialiste ou le non-informaticien.
  - Fournir des spécifications des applications à restituer à des équipes de développement
  - Outils de dialogue avec les clients
  - Faciliter la validation
  - Favoriser la réutilisabilté

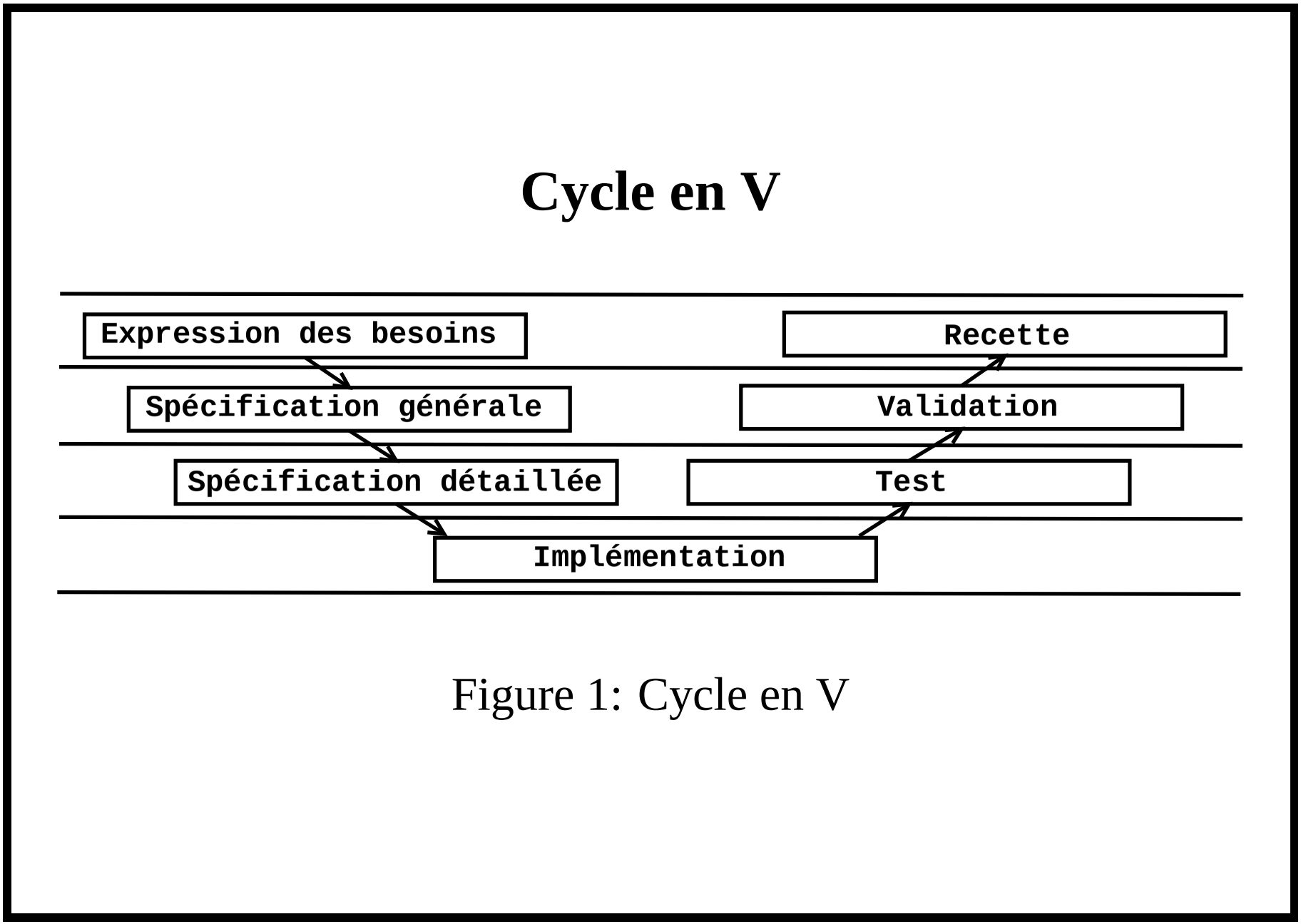


Figure 1: Cycle en V

La nécessité d'une méthodologie (2)

- Grande différence d'approche entre l'activité d'un développeur passionné isolé et le développement industriel d'une application de grande taille (centaines de programmeurs, plusieurs années).
  - Travail isolé : habitudes et méthodes personnelles.
  - Travail industriel
    - Applications de plus en plus complexes et capables d'évoluer
    - Communication nécessaire et efficace entre les participants
    - Cahier des charges

Modèle de développement en spirale

On doit pouvoir disposer de prototypes puis de versions qui s'améliorent progressivement en respectant un cahier des charges. Il faut donc savoir faire évoluer le développement de manière efficace.

1.3 Les modèles, langages et outils retenus

- UML
- Java
- BlueJ
- Swing

UML : Unified Modeling Language

- S’est construit suite à l’identification des différentes phases de conception d’un logiciel et des contraintes associées
- S’est construit suite au développement de recherches méthodologiques depuis 10 ans autour de la conception orientée objets (OO)
- Début 90 :
  - Plus de 50 méthodes OO
  - 3 méthodes majeures fusionnent sous l’impulsion de leurs concepteurs : James Rumbaugh, Gary Booch et Ivar Jacobson.

  - Il en résulte un langage unificateur de description graphique. Ce n’est pas une méthode.

- Novembre 97 : standard UML adopté par l’OMG (Object Management Group)

Java

- Langage de programmation objet multi-plateforme, intégrant des bibliothèques graphiques et de la programmation événementielle.
- Inclus des sytèmes d’interfaçage avec le Web.
- Extension aux développements d’applications distribuées ... réseaux d’entreprises.

Swing

- Bibliothèque graphique Java basée sur MVC (Modèle Vue Controleur) qui définit des concepts pour le développement d'IHM.

BlueJ

- Environnement de développement dédié à Java de l’Université de Southern Denmark ... C’est une belle interface adaptée à l’enseignement de Java (Gestion interactive des objets, visualisatiuon des diagrammes UML, ...).

Chap. 2 : Le modèle objet avec UML

2.1 Les raisons d’une méthodologie objet

L’objet, un élément de programme dynamique opératoire qui simplifie la complexité d’un problème

- Objet : modules cohérants regroupant des données et des opérations associées
- Ils sont créés dynamiquement au cours d’un programme (instanciation) à partir d’une classe, abstraction statique qui définit les opérations réalisables par ses objets.

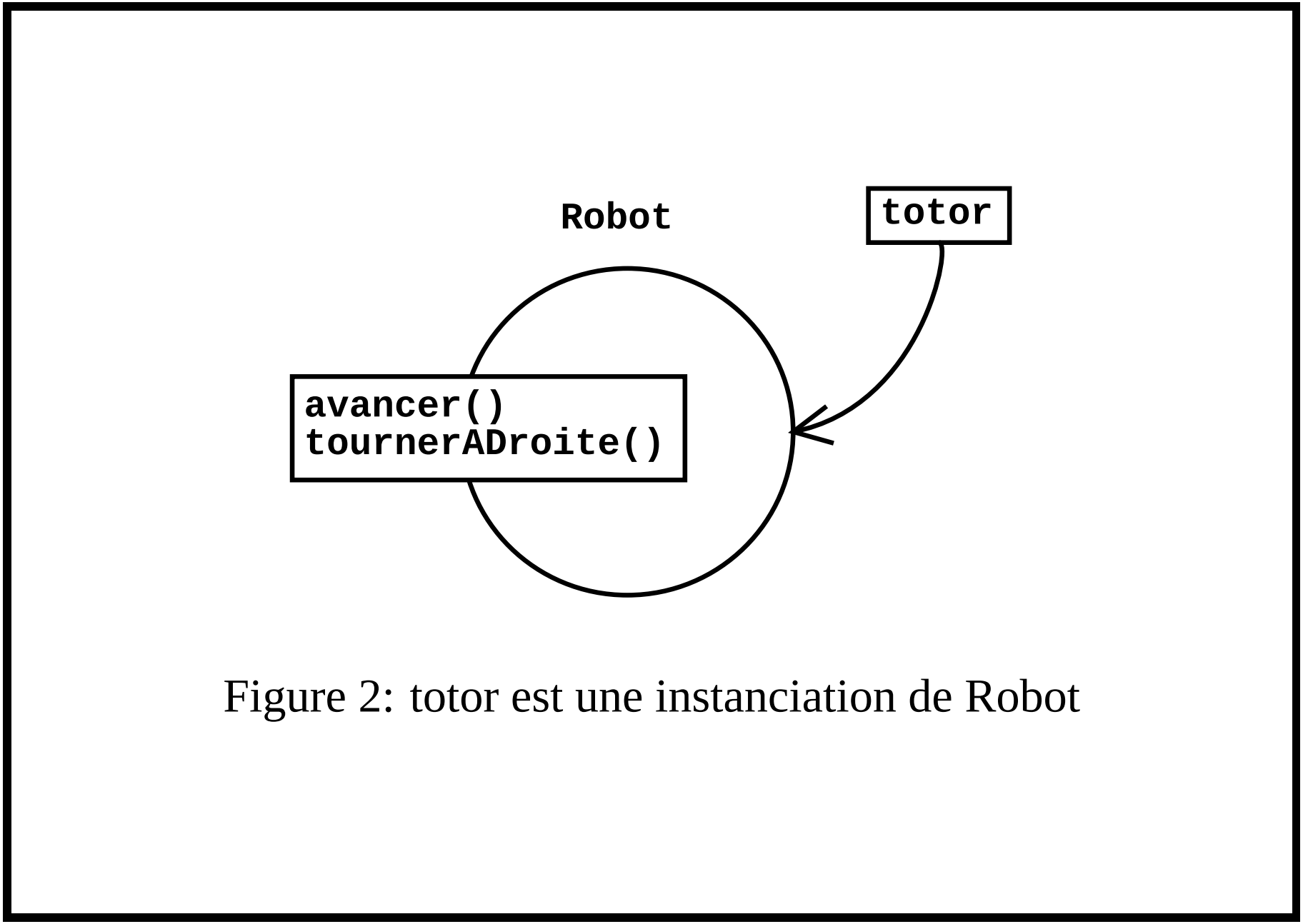


Figure 2: totor est une instanciation de Robot

- Approche décentralisée d'un développement logiciel : des unités élémentaires, objets, qui interagissent
  - simplifie la complexité d'une description globale parfois non souhaitable ou non accessible
  - pas nécessaire de connaître le système complet pour en développer une partie

L’encapsulation

- Un objet est une coquille cachant à l’utilisateur son contenu (données ou opérations)
- Interface publique : elle décrit les données ou opérations accessibles à l’extérieur de l’objet (par les autres objets).

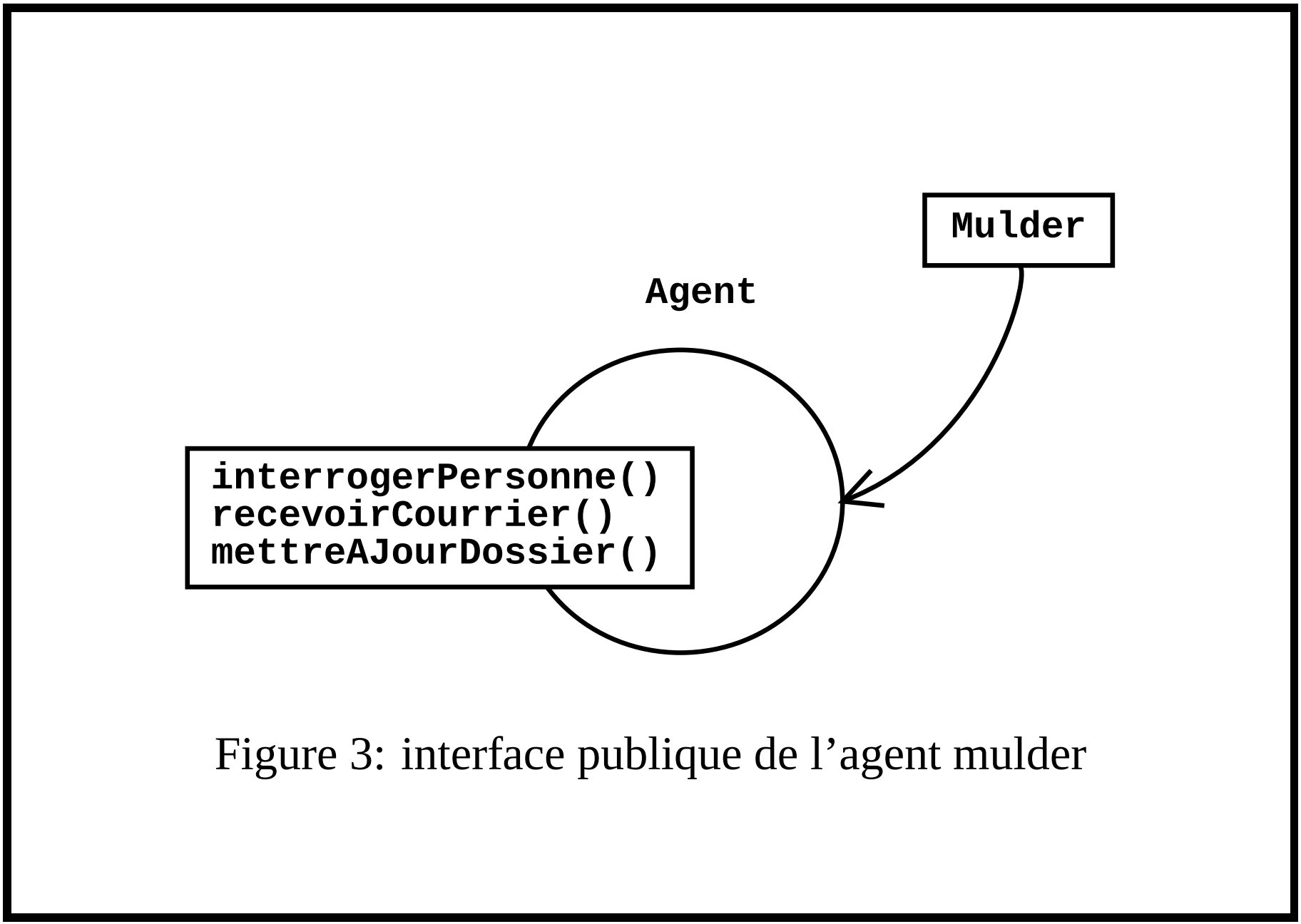


Figure 3: interface publique de l’agent mulder

- Interface privée : elle décrit les données ou méthodes non visibles hors de l'objet.

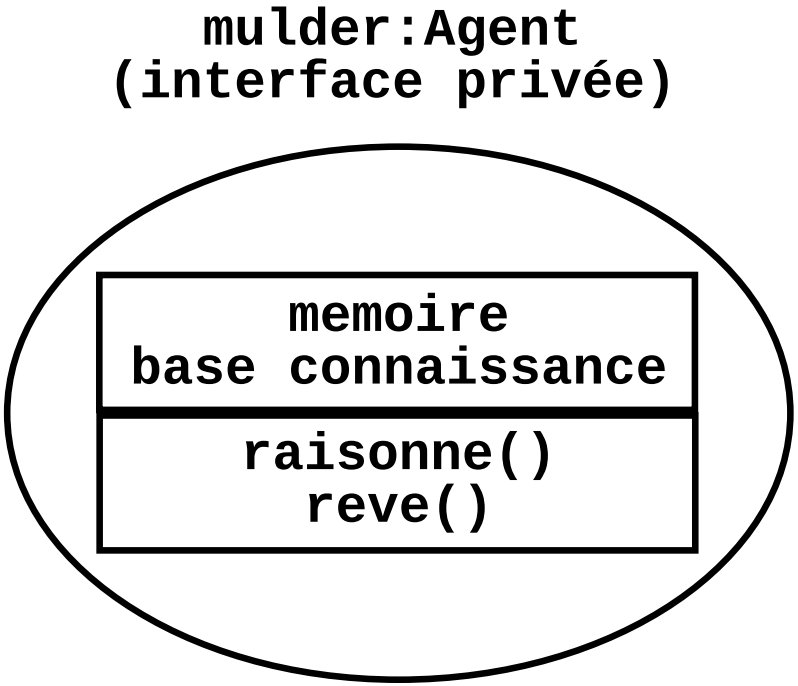


Figure 4: interface privée de l'agent mulder

L’héritage

- Une classe A peut se définir comme spécialisation d’une autre classe B. On dit que A hérite de B. A hérite de toutes les données de B et peut en définir d’autres ou en adapter certaines à ses caractéristiques.
- Permet la réutilisabilité
- Permet de “factoriser” des parties communes
- Facilite les évolutions du programme

Bilan : réutilisabilité, adaptabilité et sureté

- Réutilisabilité : grâce à l'héritage et à la définition de classes génériques pouvant être dérivée.
- Sureté : grâce à l'encapsulation. Le développeur ne possède que des contextes locaux délimités et plus faciles à gérer.

2.2 Décrire les utilisateurs et les cas d'utilisation : diagramme de cas

- Objectif : conceptualiser le problème du point de vue du client/utilisateur
- Représentation orientée utilisateur
- Découper le système en grandes tâches à répartir entre les équipes de développement
- Outil privilégié de communication entre les équipes et avec les clients

Les utilisateurs On recense les différents types d'utilisateurs en interaction avec le système.

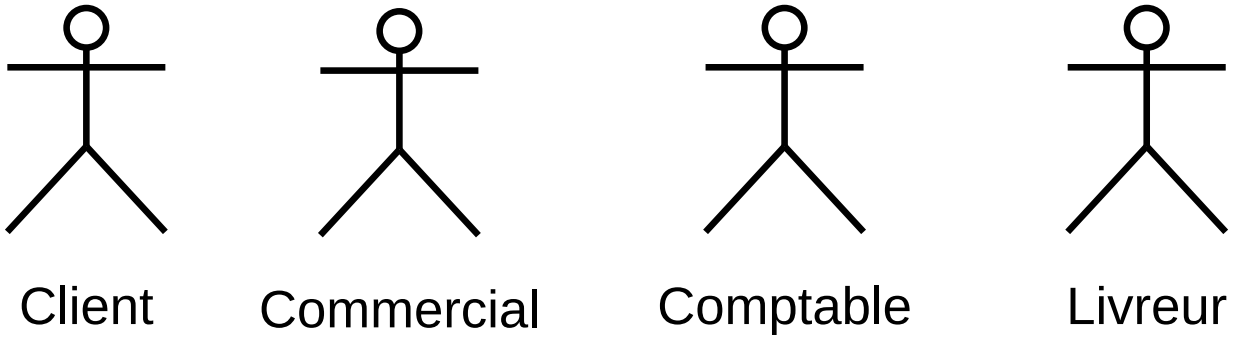


Figure 5: exemple d'utilisateurs

Les cas d'utilisation On recense les actions ou évènements demandés ou réalisés par les utilisateurs autour du système.

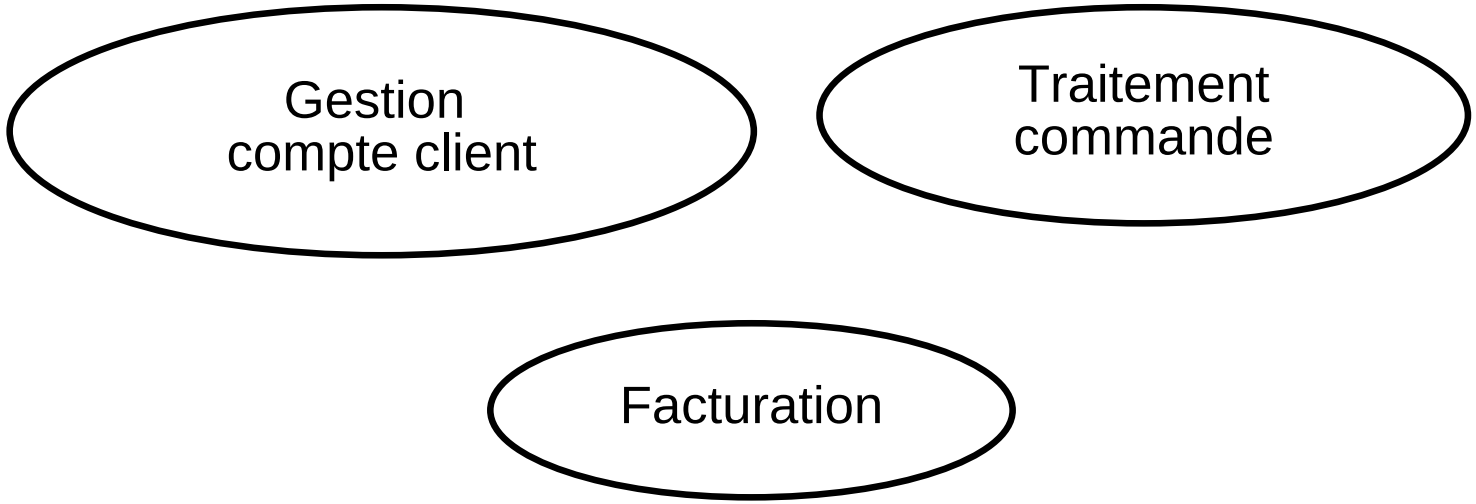


Figure 6: exemple de cas d'utilisation

Les diagrammes de cas

- Liaisons entre utilisateurs et cas
- Liaisons entre cas d'utilisation ("uses" ou "includes"). Sur l'exemple, la gestion de compte du client "utilise" le traitement de commande qui "utilise" la facturation.
- On peut aussi introduire une liaison "extends" entre des cas : c'est analogue à de l'héritage.

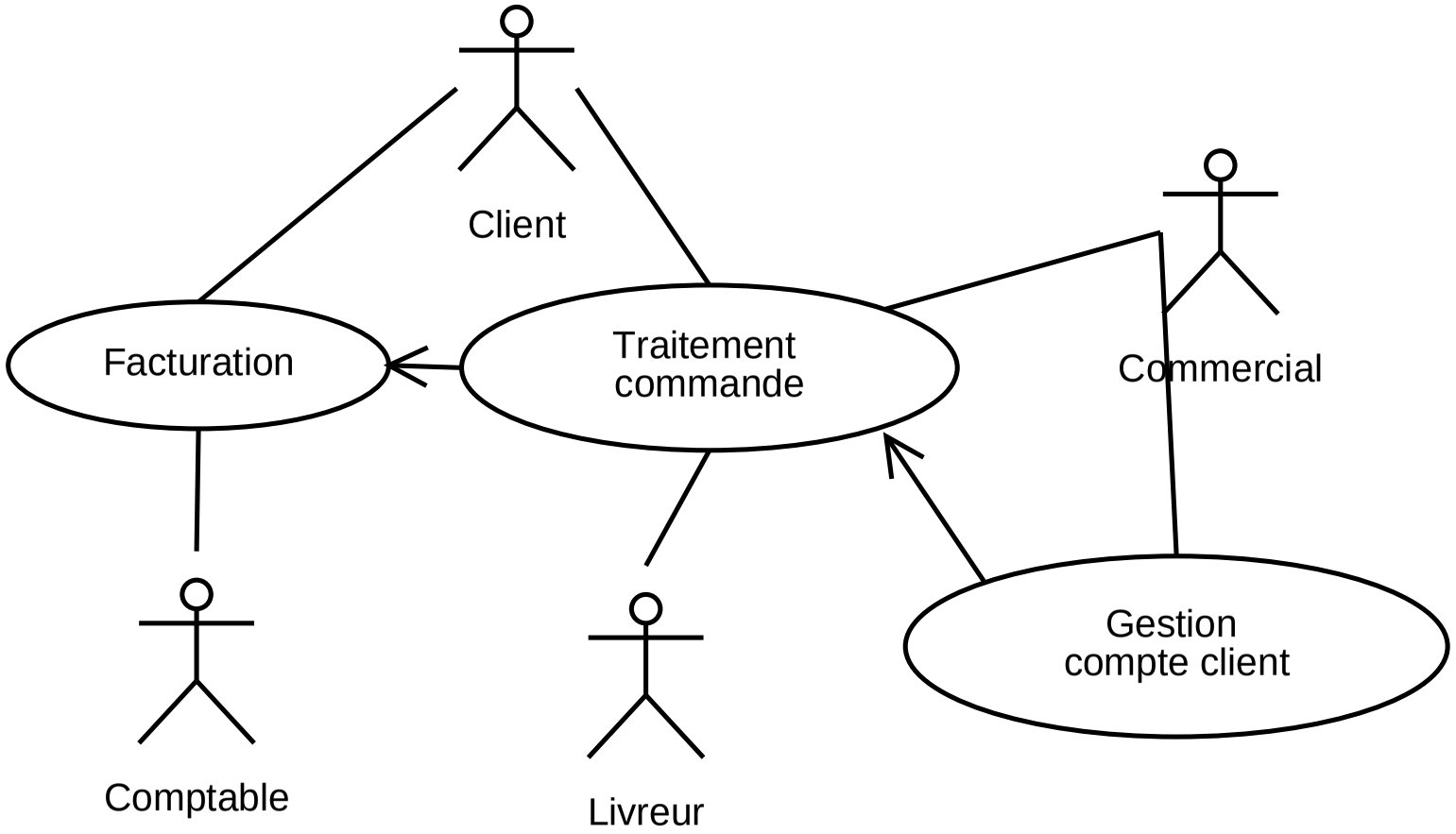


Figure 7: exemple de diagramme de cas

2.3 Décrire des objets : diagramme de classe

Une classe est un modèle à partir duquel on construire des objets. Elle est représentée par un rectangle découpé en 3 parties :

- Identifiant de la classe
- Liste des attributs
- Liste des méthodes

Les attributs ou méthodes sont précédés d'un symbole désignant sa visibilité :

- \- l'attribut ou la méthode est privé (visible que dans la classe)
- \+ l'attribut ou la méthode est public (visible partout)
- \# l'attribut ou la méthode est protégé (visible dans la classe et ses sous-classes)

- Les méthodes sont suivie par “:” et le type de retour, s’il y en a un (non indiqué si de type “void”).
- Les attributs peuvent être suivis de “=” et d’une valeur par défaut.
- Attention : Au niveau de la description UML, les attributs ne peuvent être que de type élémentaire (int, double, boolean, string, ...) et il ne peuvent être de type objet. Pour définir une telle dépendance, on utilise les associations.

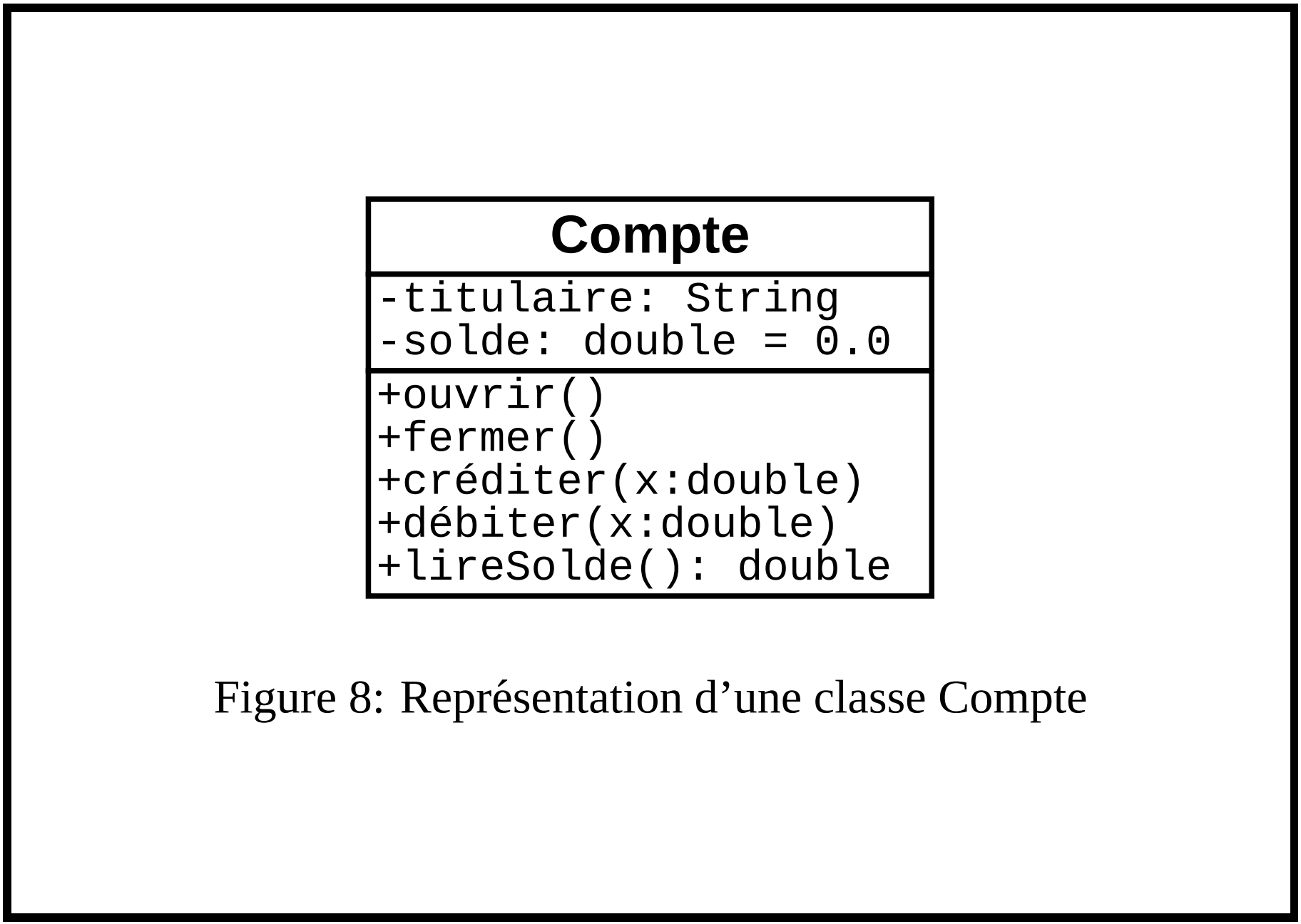


Figure 8: Représentation d'une classe Compte

La traduction en Java de cette classe, sans son implémentation, est la suivante :

```
class Compte {
   string titulaire;
   double solde = new Somme(0.0);

   public void ouvrir() { ... };
   public void fermer() { ... };
   public void crediter(double x) { ... };
   public void debiter(double x) { ... };
   public double lireSolde() { ... };
}
```

Pour désigner un objet d'une classe, on le spécifiera de la manière suivante :

Figure 9: Représentation d'un objet

Les associations

- Les associations correspondent à des relations entre objets.
- Description alternative des attributs-classes, interdits en UML
- Elles sont nommées avec un sens de lecture, si nécessaire
- Un rôle est attribué à chaque classe associée. Il est indiqué aux extrémités des relations.

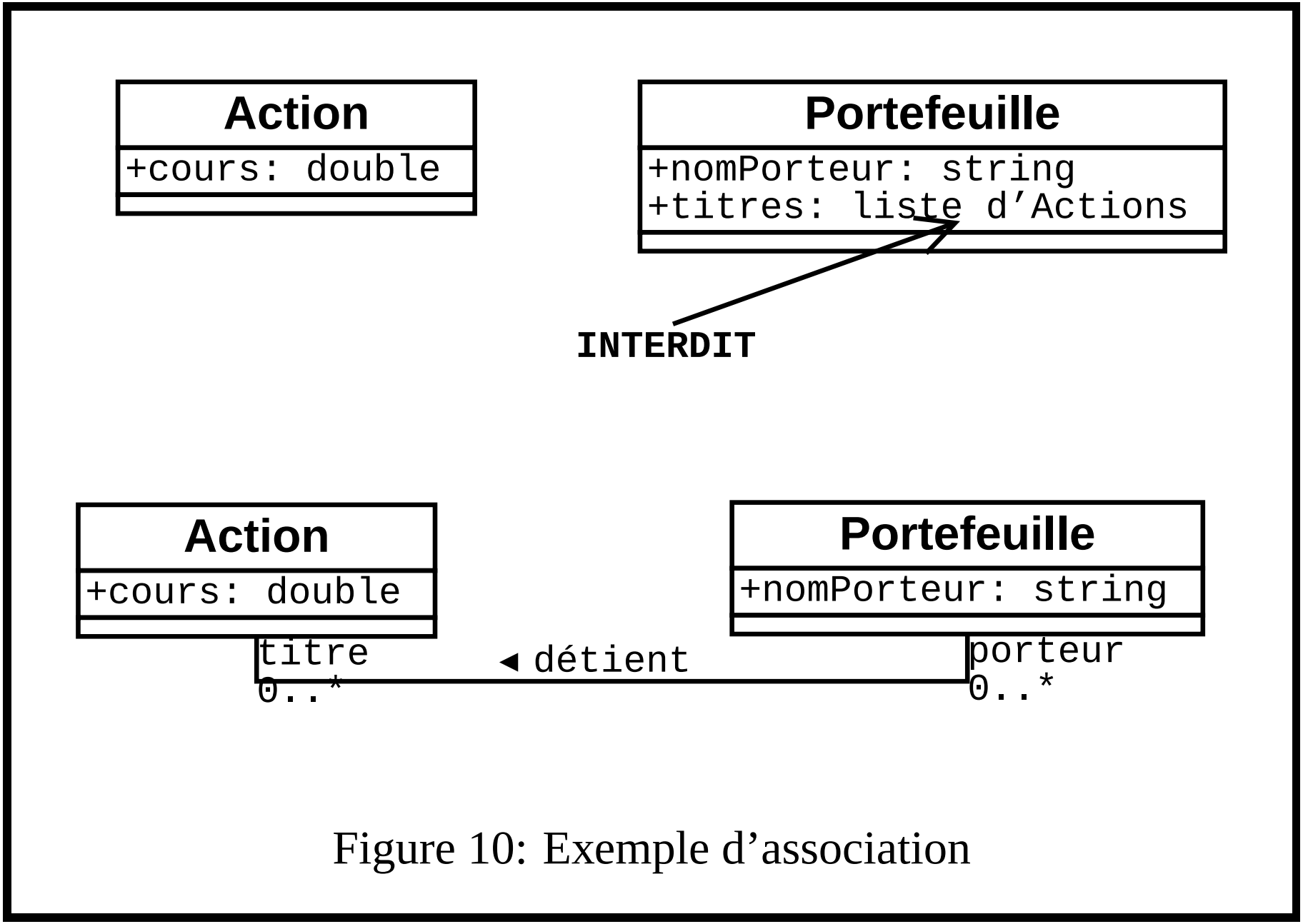


Figure 10: Exemple d'association

Cardinalités

Les cardinalités sont associées aux rôles et indique le nombre d'objets d'une même classe participant à l'association :

- 1 : obligatoire (un et un seul)
- 0..1 : optionnel (O ou 1)
- 0..* ou * : quelconque
- n..m : entre n et m
- l,n,m : l, n ou m

Sous-type et généralisation

Correspond à la notion d'héritage en C++ et Java.

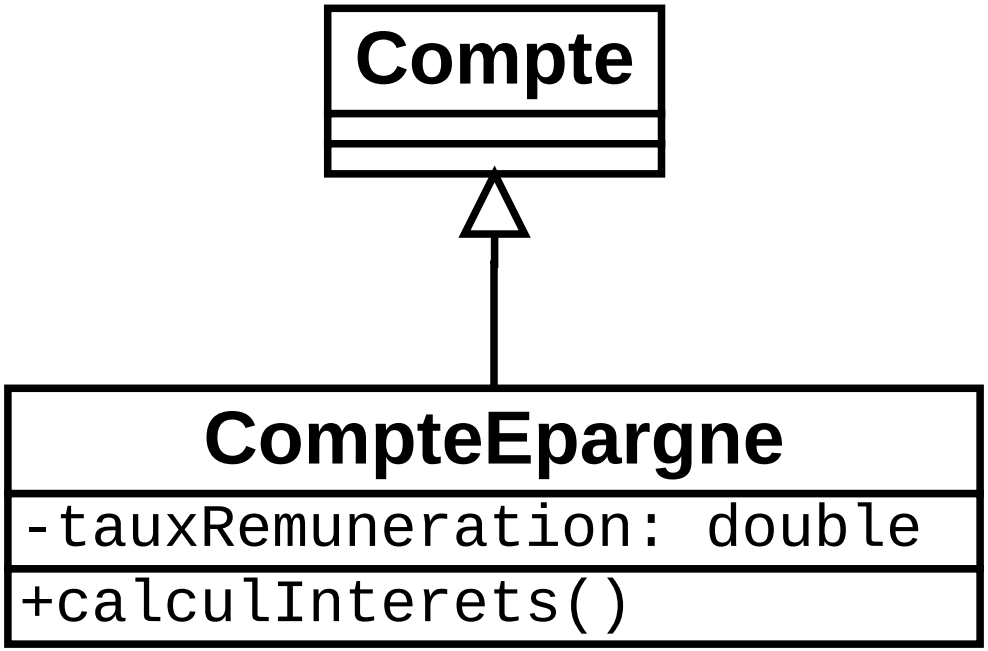


Figure 11: Exemple de généralisation

Agrégation et composition

Concerne les relations “partie de” qui peuvent être vues de 2 manières.

- La composition, représentée par un losange noir, signifie que l’objet est une partie indissociable de l’objet auquel il est lié. Par exemple, une chambre d’hôtel et son hôtel de rattachement : si ce dernier disparait, la chambre disparaitra aussi.
- L’agrégation, représentée par un losange blanc, signifie que l’objet est référencé par l’objet auquel il est lié, il n’en est pas la propriété. Par exemple, les wagons d’un train peuvent être ultérieurement rattachés à un autre train.

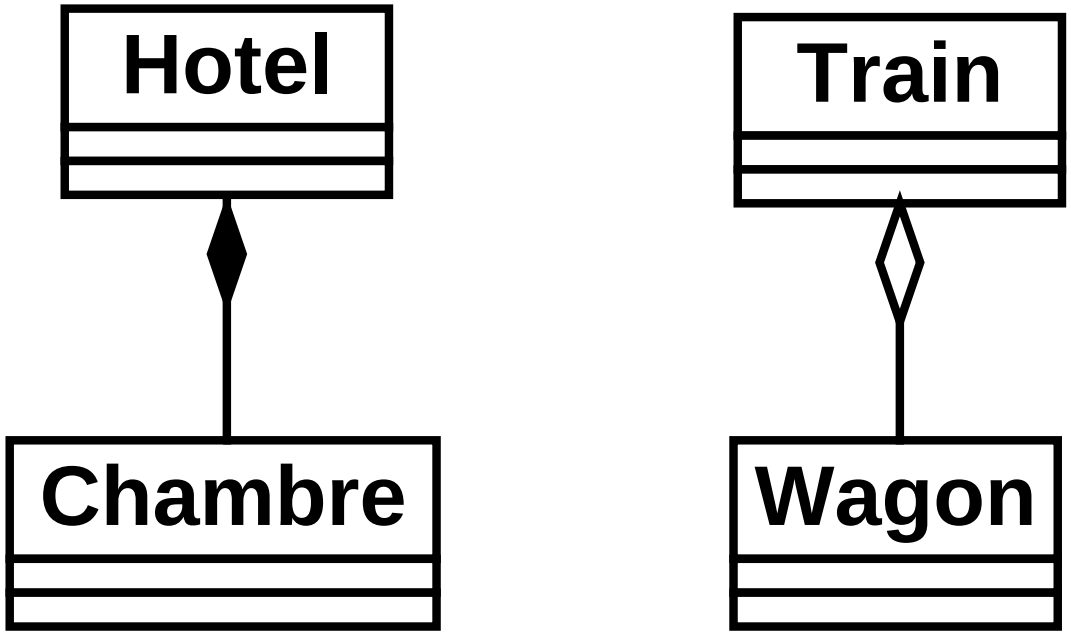


Figure 12: Exemple de composition et d’agrégation

Compléments et notions avancées

- Classe abstraite : classe générique qui n'est pas instanciée dans l'application
- Classe paramétrée : correspond à des templates en C++.
- Interface : décrit un comportement générique de classe. Similaire aux interfaces Java.
- Associations n-aire
- Attributs et classes d'association

2.4 Décrire des programmes : diagrammes de séquence, d'état et d'activité

Collaboration entre les objets, diagrammes de séquence

- Les diagrammes de séquences ont pour objet de décrire des scénario particuliers de comportements des acteurs vis-à-vis du système.
- Ensemble de colonnes représentant chacun un objet ou un acteur. La longueur de la colonne correspond à la durée de vie de son interaction avec les autres.
- Les flèches entre les colonnes correspondent aux messages envoyés.

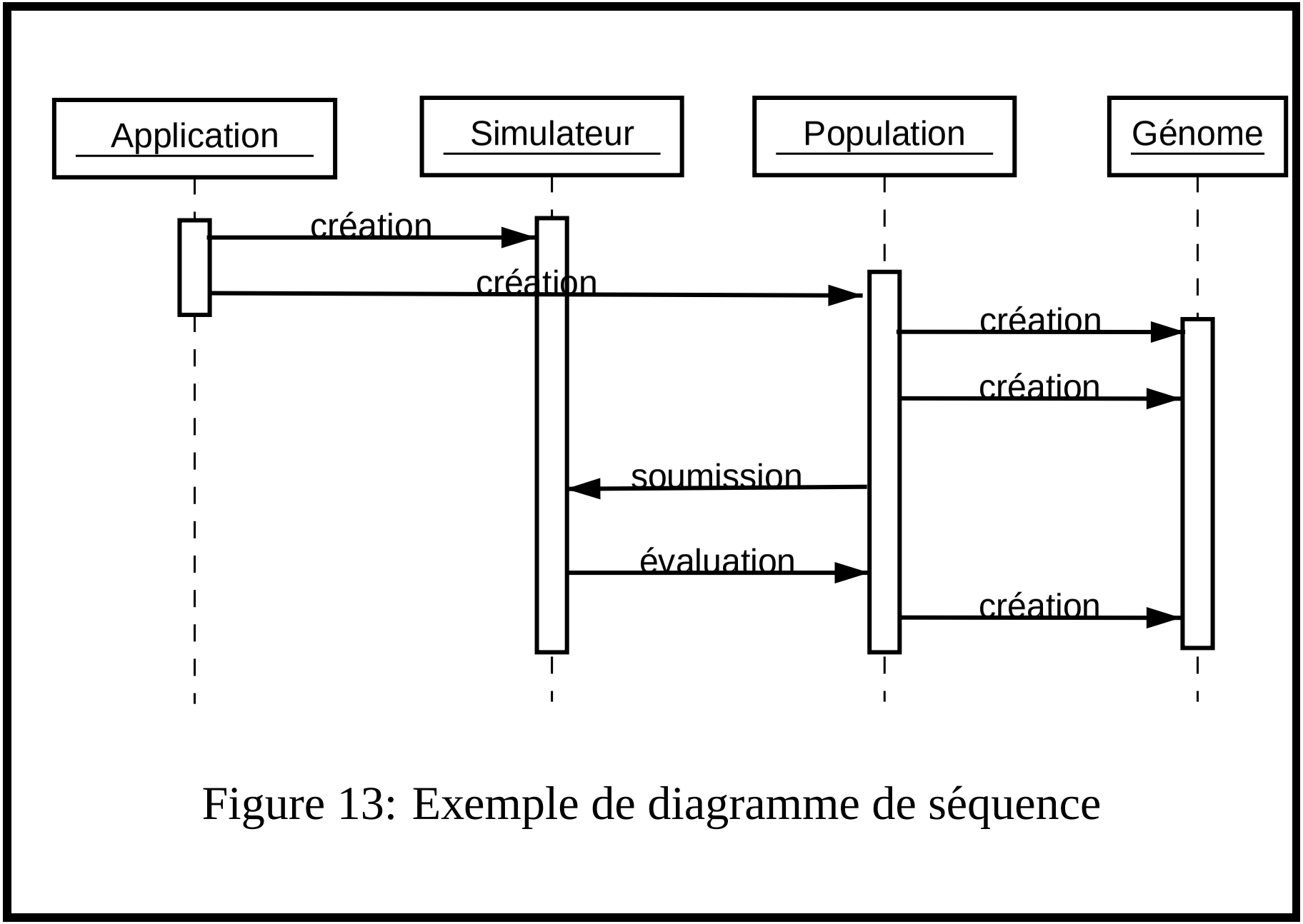


Figure 13: Exemple de diagramme de séquence

- Un programmeur doit pouvoir coder à partir d’un diagramme de séquence.
- Eviter les diagrammes tentaculaires. Un diagramme de séquence doit correspondre à un algorithme simple.

Aspect dynamique d’un objet : automate et diagramme d’état

Les diagrammes d’état indiquent les changement d’état d’un objet à travers les cas d’utilisation dans lesquels il est impliqué.

- Etat-transition De la forme <nomEvenementDeclancheur> [<contrainte>] / <nomActionDéclanchée>.

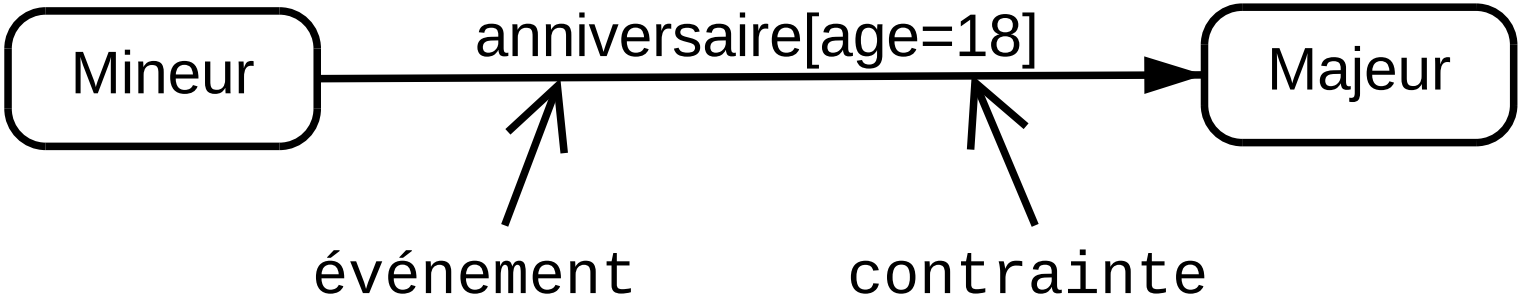


Figure 14: Exemple d’état-transition

- Une transition peut provoquer une activité, notée “do/action” : opération durable, interruptible, associée à un état. Exemple de la modélisation d’un réveil à 3 boutons : alarm on/ alarm off/ stop sonnerie

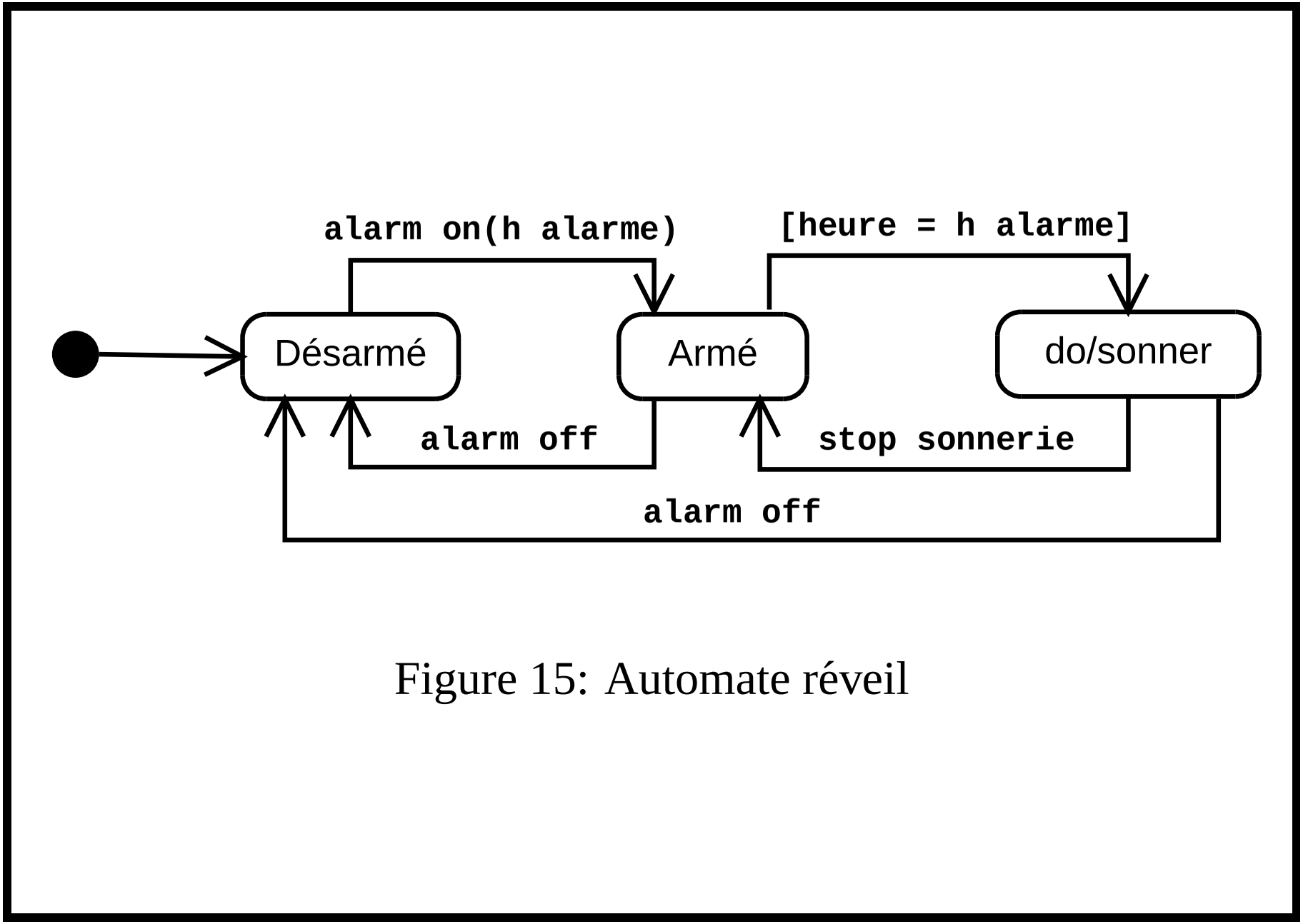


Figure 15: Automate réveil

- Action en entrée et en sortie :
  - Action en entrée (entry) exécutée à chaque entrée dans l'état (ex : redessiner le contenu de la fenêtre quand la souris y est).
  - Action en sortie (exit) exécutée à chaque sortie d'un état (ex : mettre à jour un compteur de visite).

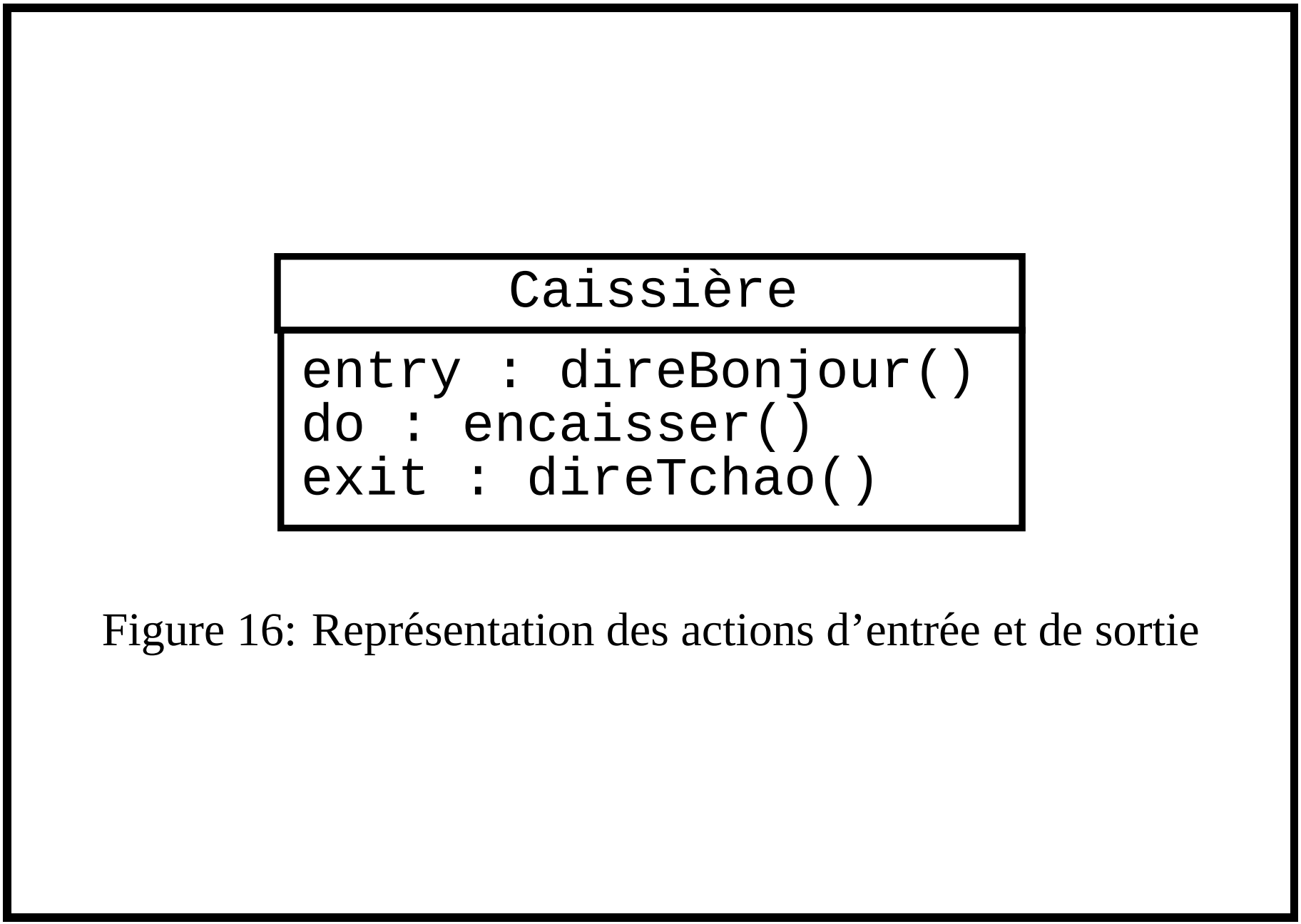


Figure 16: Représentation des actions d’entrée et de sortie

- Diagrammes hiérarchiques
  - Permettent de structurer la description et en facilite la lecture.
  - On entre dans l’état englobant puis dans ses sous-états.

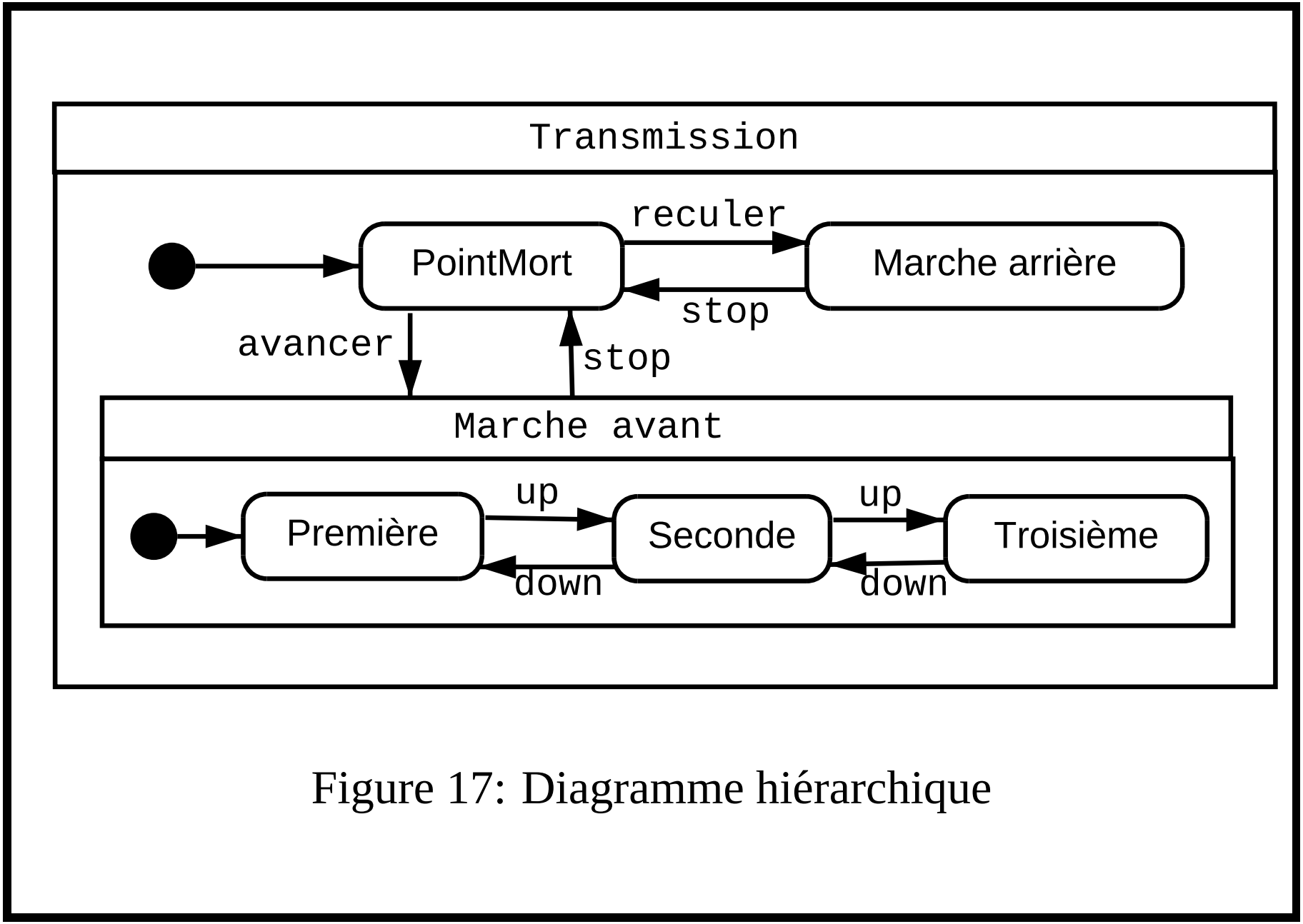


Figure 17: Diagramme hiérarchique

- Parallélisme et synchronisme

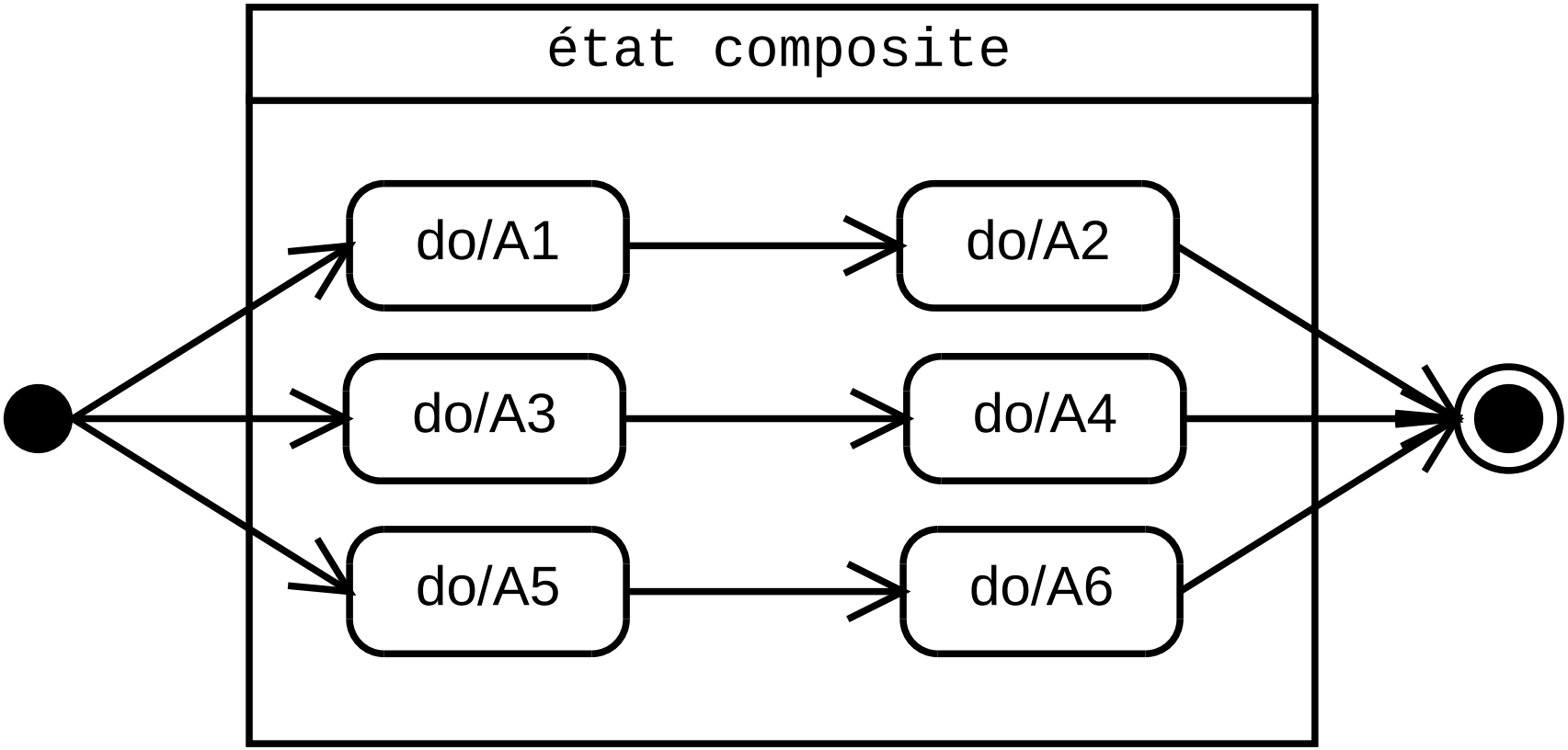


Figure 18: Etat à traitement parallèle

Diagramme d’activité

- Doit correspondre au comportement d’une seule méthode
- Doit permettre de mettre en évidence les contraintes de séquentialité et de parallélisme

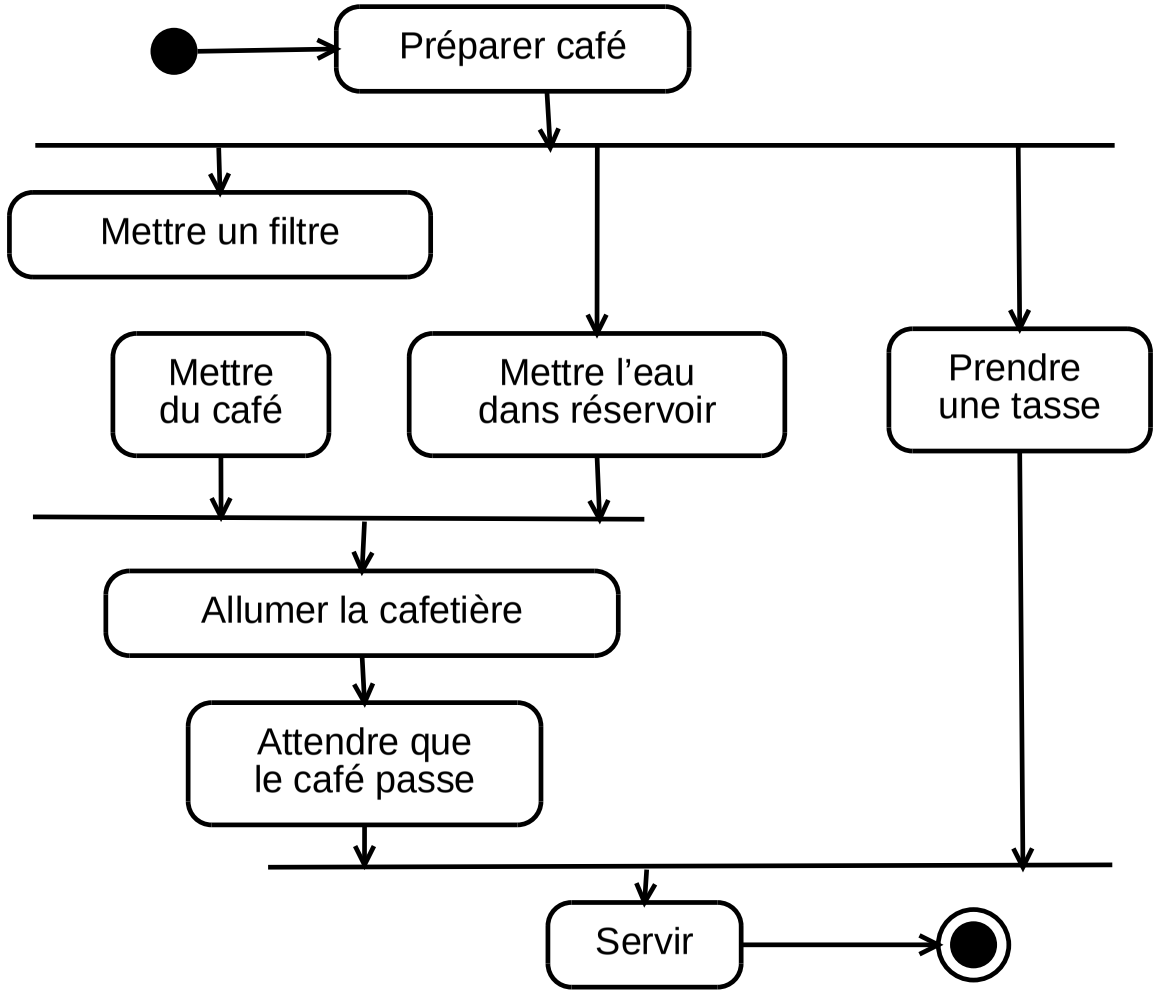
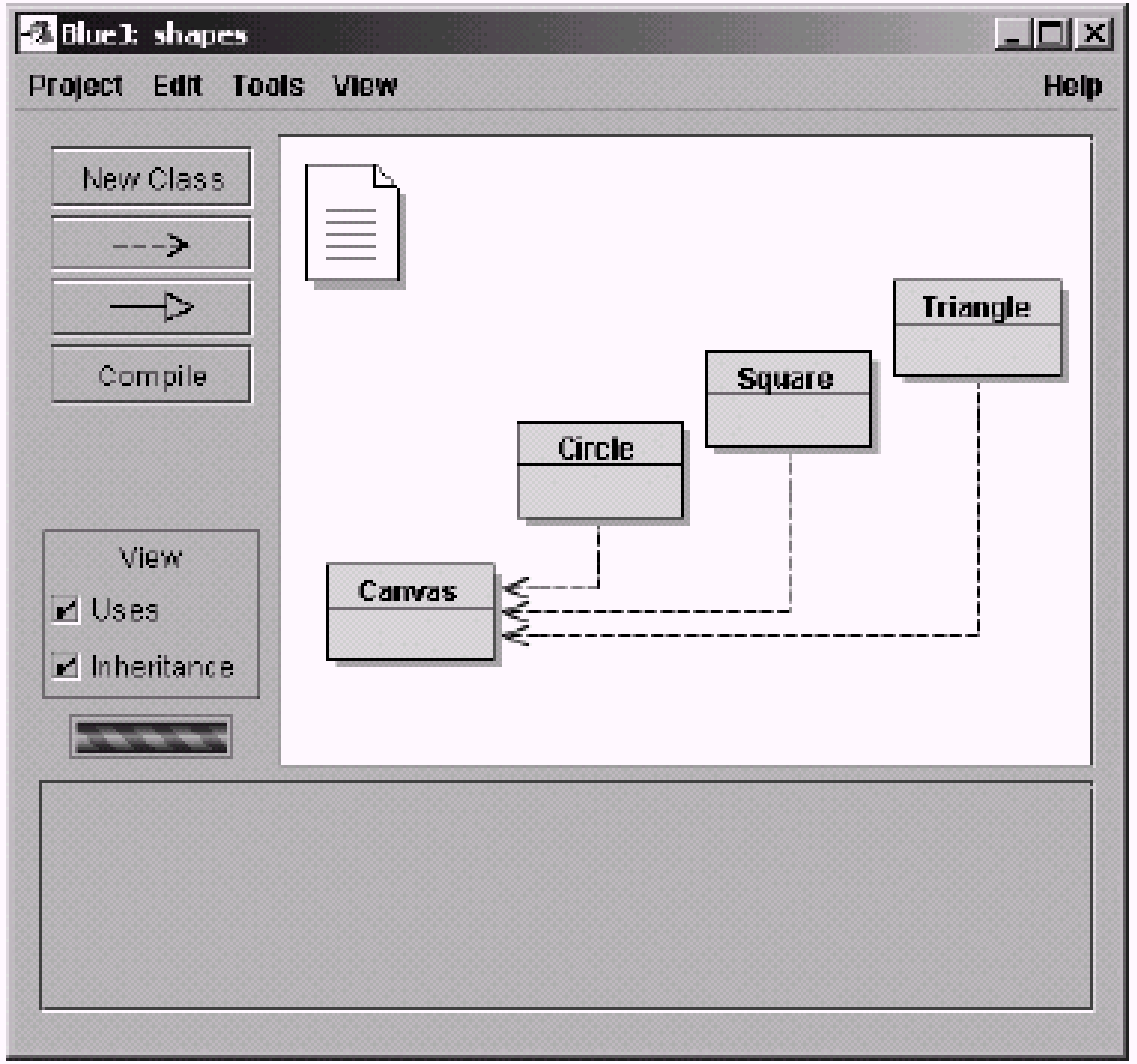


Figure 19: Exemple de diagramme d'activité, la pause café

Présentation rapide de BlueJ

BlueJ est un outil de développement pour l'apprentissage du langage Java. C'est un interface utilisateur qui contient un éditeur et des accès faciles aux outils du JDK.

Il permet aussi de gérer interactivement des objets dont il fait une représentation graphique qui s'appuie sur UML. Sur la figure suivante, on visualise l'interface qui représente une hiérarchie de classe.


BlueJ: shapes
Project
Edit
Tools
View
Help
New Class
Compile
View
Uses
Inheritance
Triangle
Square
Circle
Canvas

A partir de cette représentation, on peut alors, par exemple, cliquer-droit sur la classe “Circle” et choisir, à la souris, “new Circle()” : une représentation de l’instanciation apparaît alors dans le rectangle bas de l’interface.

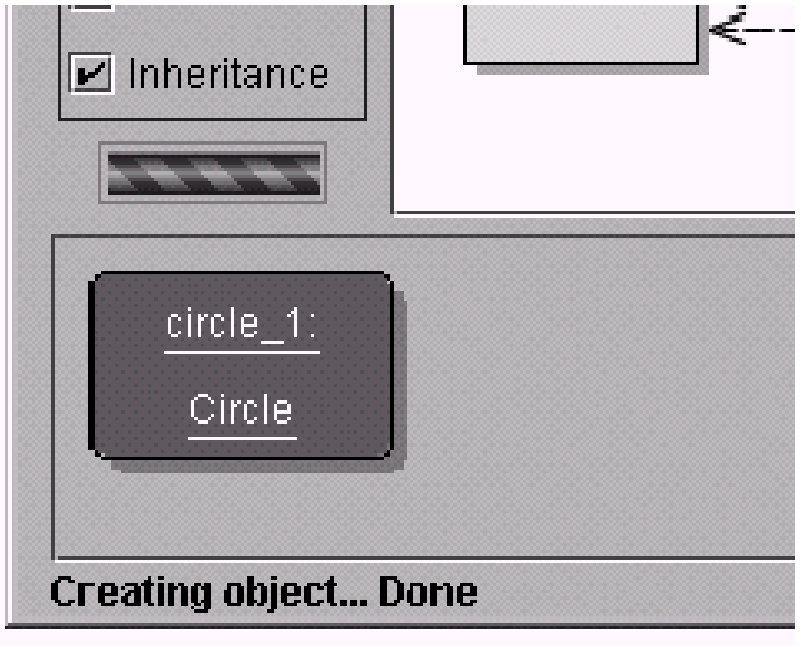


Figure 20: Visualisation de l'instanciation d'un objet de classe Circle

On peut ensuite appeler des méthodes des objets créés, en cliquant-droit sur ceux-ci et choisir la méthode dans la liste des méthodes accessibles. Sur la figure suivante, on a invoqué la méthode “makeVisible()” de l’objet cercle. La visualisation se fait alors dans une fenêtre de visu.

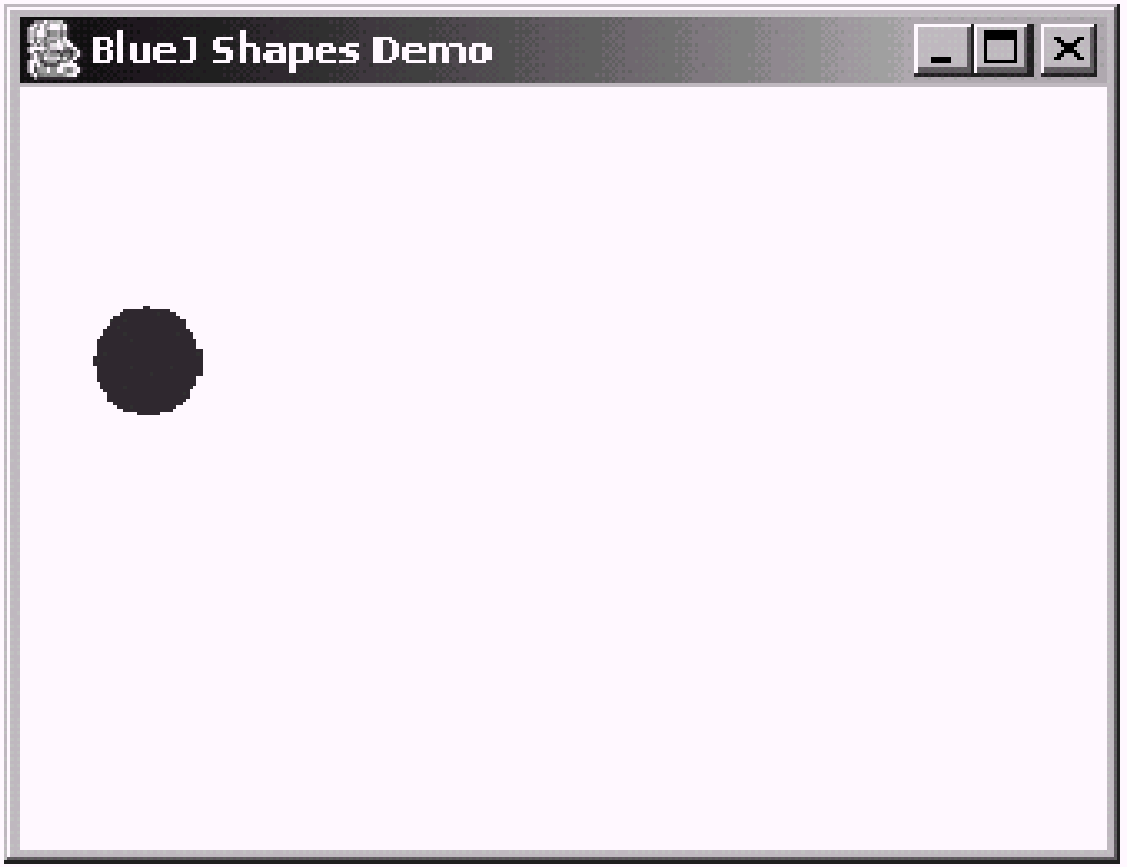


Figure 21: Tracé graphique du cercle instancié dans une fenêtre de visu

Lorsqu’une méthode attend des paramètres, à la sélection de celle-ci, une fenêtre apparaît, avec un champ de formulaire pour pouvoir saisir des valeurs aux paramètres. La figure suivante montre comment affecter une valeur au paramètre de la méthode “moveHorizontal”.

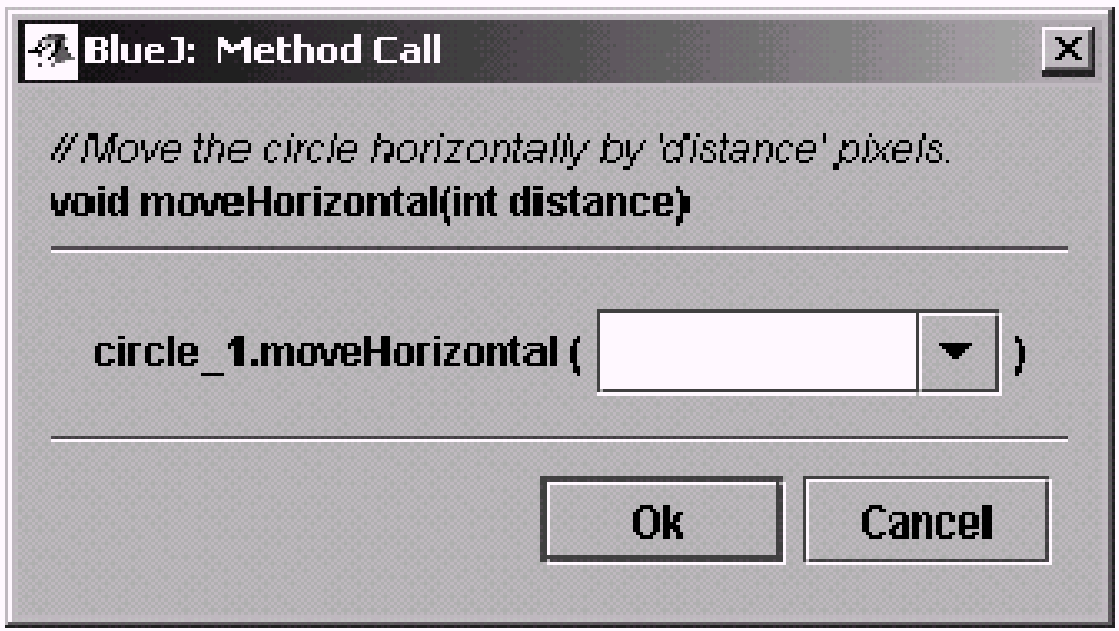
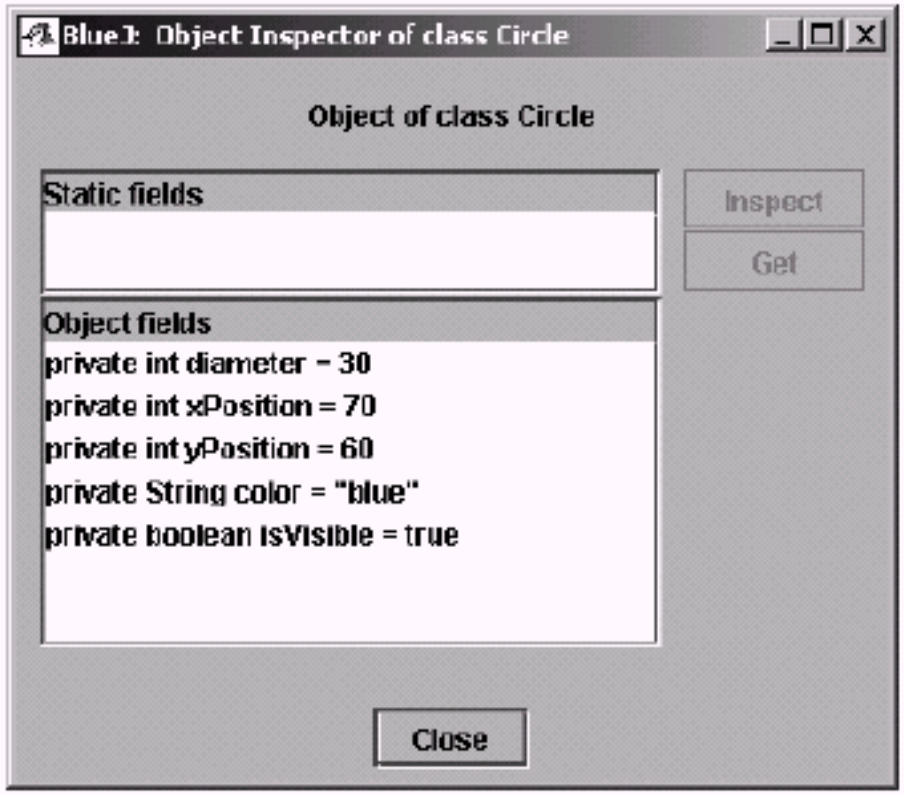


Figure 22: Saisie de la valeur d'un paramètre

BlueJ propose aussi des fenêtres d'inspection des différents attributs des objets.



Vous serez donc amené à manipuler et à utiliser BlueJ en TD/TP et à en connaître ainsi ses fonctionalités complémentaires.