Programmation orientée objet et temps réelle avec Java

Introduction à la programmation orientée objet

Dominique Blouin
Ingénieur de recherche
Télécom Paris, Institut Polytechnique de Paris
dominique.blouin@telecom-paris.fr

Objectifs d’apprentissage du cours

- **Maîtriser les bonnes pratiques de programmation et les savoir-faire permettant de travailler dans un contexte de développement logiciel professionnel.**

- **Avoir une bonne maîtrise de la programmation orientée objet au travers du langage Java d'usage répandu dans l'industrie.**

Travaux Pratiques

- **Le cours propose une alternance régulière entre Cours Magistraux (CM) et Travaux Pratiques (TP)**

- **Les TP se font sur ordinateur avec l'environnement de développement Eclipse**
  - Bien que les TP présentent l'utilisation de Eclipse, vous pouvez utiliser l'environnement de développement que vous préférez.
  - L'idéal est d'utiliser votre propre ordinateur, sinon les ordinateurs des salles de TP.

- **Les premiers TP numérotés de 1 à 4 ne sont pas à rendre.**

Projet

- **Développer un simulateur de système de production robotisé.**
  - Exemple: https://www.hpi.uni-potsdam.de/giese/public/cpslab/

- **Le développement du projet sera guidé à partir du TP 5:**
  - Il faudra bien conserver vos sources: idéalement utiliser un outil de gestion de version tel que Git.
  - Il faudra me rendre les sources à la date limite de rendu de projet.
    - A déterminer ensemble…

- **L'évaluation du projet se fera selon plusieurs critères:**
  - Bon fonctionnement du simulateur selon le cahier des charges fourni.
  - Bonnes pratiques de programmation dans le code.

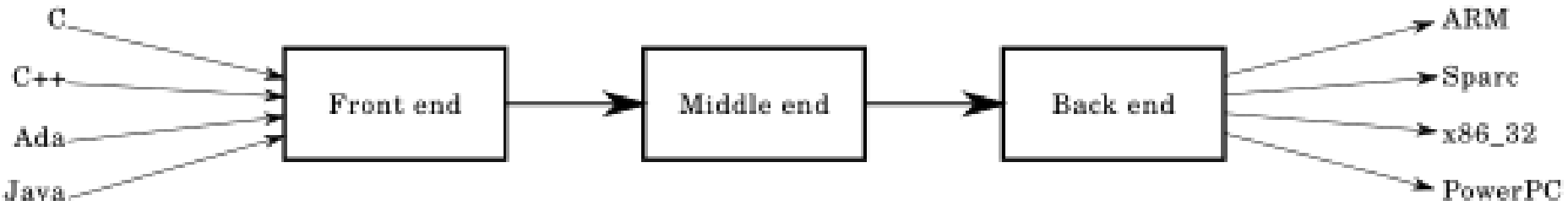
Objectifs d’apprentissage de cette période

- **Programmes et programmation.**
- **Introduction au paradigme de programmation Orienté Objet (OO).**

Langages de programmation

- Un langage de programmation permet au programmeur d'écrire son programme avec des concepts de haut niveau.
  - Par exemple, tous les langages proposent la notion de liste de données numériques.

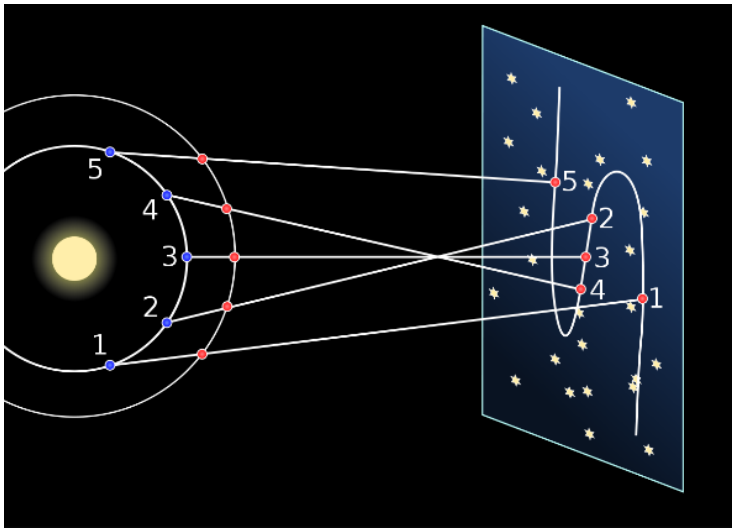
- Le **compilateur** traduit ces concepts de haut niveau en instructions pour l'UAL (Unité Arithmétique et Logique).



- Codées en octets, ces instructions seront chargées en mémoire pour l'exécution du programme.

Paradigmes de programmation

- **Paradigmes scientifiques:**
  - Géocentrisme versus héliocentrisme en astronomie.
  - **Changements** de paradigmes.



- **Paradigmes d'ingénierie:**
  - Concernent le cycle de vie du système (développement, méthodes et outils, processus de développement, etc.).
  - Concernent également *l'environnement* dans lequel l'ingénierie engineering se fait

- **Caractéristiques communes:**
  - Moyens de *caractériser* un ensemble *artefacts* utilisés dans un *environnement* pour solutionner un problème

- ***Paradigmes de programmation:***
  - Caractérisent un langage de programmation (*syntaxe* et *sémantique)*

Types de paradigmes de programmation

- **Paradigme impératif:** les programmes de ces langages sont des commandes dont l'exécution est déterminée des structures de contrôle. On parle de langage de programmation impératif ou procédural.
  - Il y a des structures de données comme par exemple un tableau de nombres.
  - Des procédures prennent en paramètres ces structures de données pour effectuer des actions sur ces données et/ou calculer des résultats à partir de ces données.
  - Exemple: une procédure pour trier (réordonner) les éléments du tableau en ordre croissant.

- **Type de programmation le plus répandu.**
  - Permet de résoudre les problèmes pour lesquels on peut construire une suite de commandes apportant une solution.

Extensions du paradigme impératif

- **Fonctionnel:** le programme est défini par des fonctions au sens usuel du terme. On demande à l'ordinateur de calculer des expressions.
  - Lisp, Haskel, OCamel, Java, etc…

- **Orienté Objet (OO):** le langage permet de décrire ou de modéliser un problème par une collection d'objets qui communiquent entre eux par envoi de **messages**.
  - Java, C#, Smalltalk, Python, Simula, etc.
  - Langages de **modélisation** *(Modelica, Simulink, Scade, etc…)*

- L'OO apporte de la facilité de programmation et une vision plus claire du problème.

Paradigme déclaratif

- **Paradigme déclaratif:** les programmes de ces langages décrivent un problème (souvent sous forme de formule logique) et l'exécution du programme consiste à trouver une ou des solution(s) au problème.
  - SQL, Prolog (langages de programmation par contraintes)

Naissance de l'OO

- La programmation OO a été introduite par le langage Simula à l'Université d'Oslo en 1967.
- Le problème posé était celui de la simulation d'un ensemble de robots dans une entreprise.
- Essayer de résoudre ce problème à l'aide de la programmation impérative classique est un vrai casse-tête tant les interactions entre les objets sont nombreuses et imprévisibles.
- Tentez d'imaginer un programme centralisé qui pilote les robots en modifiant une structure de données représentant l'état des robots et de l'environnement. Ce n'est pas impossible mais très difficile.
- La solution: l'orienté objet.

Principes de l'OO

- En OO, on va décrire chacun des éléments du problème par une **classe**. Cela concerne les robots mais aussi tous les éléments du problème, par exemple le poste de contrôle où siège l'humain.

- **La classe décrit les données contenues dans l'objet.**
  - Un robot aura par exemple un **niveau d'énergie** entre 0 et 100, une **position** composée de **deux coordonnées**, un **vecteur vitesse**, etc.
  - Ce sont les **attributs** de l'objet.
  - Ce n'est pas différent de la description d'une **structure de données** classique.

- La différence est que les objets peuvent s'envoyer des **messages**. Un objet recevant un message doit **répondre** à ce message après avoir effectué des **actions** (calculs).

Les robots de Simula

- Exemples de messages:
  - L'utilisateur envoie un message à un robot pour lui dire de démarrer ou d'arrêter son fonctionnement.
  - Un robot signale au contrôle que son niveau d'énergie est trop bas.
  - Deux robots se heurtent, les capteurs envoient des messages.
  - Etc.

- La classe décrit également comment les objets **répondent** aux messages. Ce sont les **méthodes** de l'objet.
  - Par exemple, si un robot reçoit le message **`start()`**,il démarre et renvoie la réponse **`done`** si le démarrage s'est bien passé ou la réponse **`failed`** si le démarrage a échoué.

Simulation des robots

- Le programmeur décrit chacun des objets de l'usine par une classe.
- Au lancement du programme, le programmeur crée des **objets** :
  - Un poste de contrôle
  - Des robots
  - Etc.
- Les objets interagissent par envoi de messages.
- Les messages initiaux sont envoyés par l'utilisateur du simulateur.
- Il n'y a pas de programme centralisé qui gère l'ensemble.

Exemple avec une interface graphique simple

- Tout ce qui apparaît sur l'écran d'un ordinateur fait partie des interfaces graphiques : fenêtres, menus, boutons, champs d'entrée, zone de dessin, etc.
- L'utilisateur, par ses interactions avec le clavier et la souris, déclenche des actions du programme:
  - Création d'une nouvelle fenêtre
  - Action associée à un bouton
  - Etc.
- Certains changements sont automatiques: les animations.
- Comme dans le cas des robots, on peut imaginer une structure de données décrivant l'état de l'interface graphique.
- Et il est tout aussi difficile d'imaginer un programme centralisé devant gérer cette interface graphique.

L'approche OO

- Dans l'approche OO des interfaces graphiques, chaque élément apparaissant sur l'écran est un objet au sens de Simula.

- Chacun de ces objets a des **attributs** :
  - Ses coordonnées sur l'écran
  - Ses dimensions
  - Sa profondeur (qui est devant qui ?)
  - etc.

- Chacun de ces objets a des **méthodes** qui lui permettent de répondre aux messages qu'il reçoit.

- Les attributs et les méthodes d'un objet dépendent de son **type**:
  - Fenêtre, bouton, etc.

- Chaque type d'objet est décrit par une **classe** contenant les déclarations d'attributs et de méthodes.

Les messages dans les interfaces graphiques

- Le principal émetteur de messages est l'utilisateur de l'ordinateur :
  - Un clic de souris avec l'un des boutons
  - Un déplacement de la souris
  - La frappe d'une touche du clavier
  - Etc.

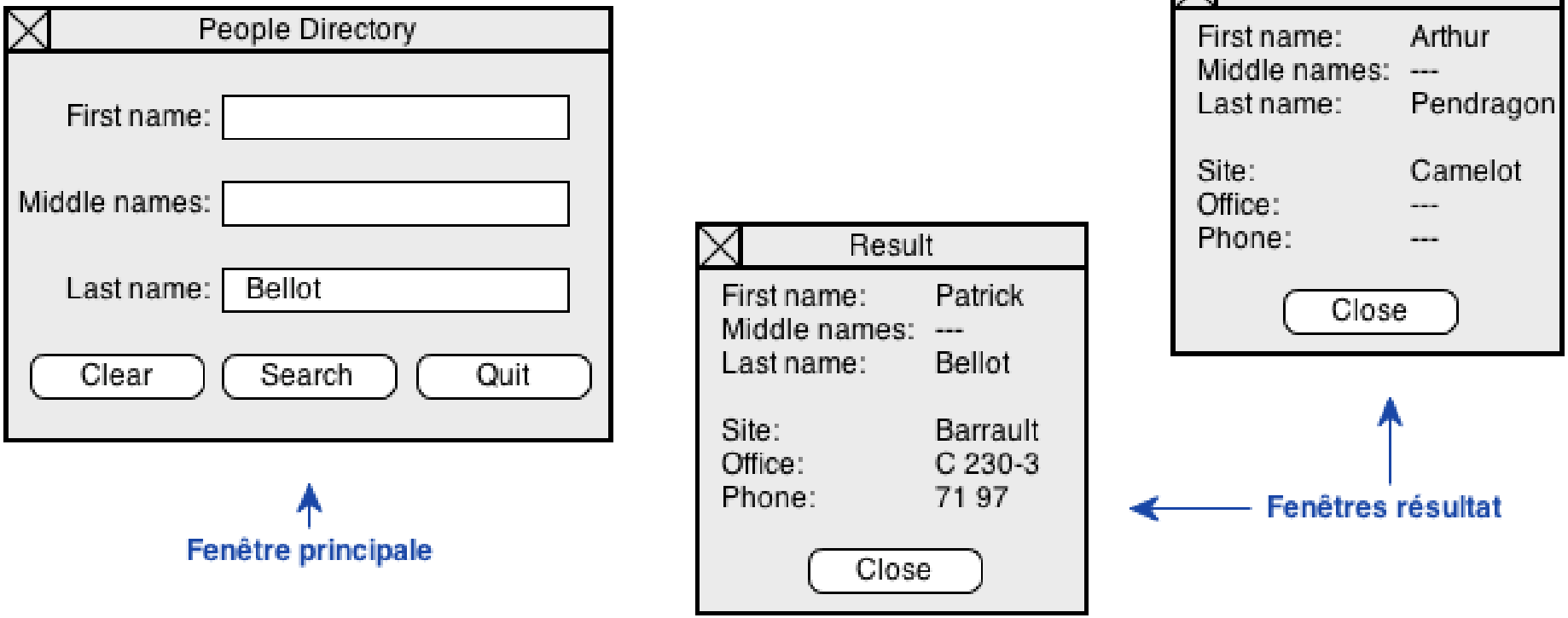
- En réponse à une action de l'utilisateur sur l'un des objets de l'interface graphique, cet objet peut émettre des messages à destination d'autres objets de l'interface graphique.

- Voyons cela sur un exemple...

Exemple pour un logiciel d'annuaire

- Dans ce logiciel, on entre des données sur une personne recherchée dans la fenêtre principale de l'application.
- Puis on clique sur un bouton *Search* et une fenêtre est créée pour afficher le résultat de la recherche.
- Un bouton *Clear* permet de réinitialiser les champs d'entrée.
- Un bouton *Quit* permet de terminer l'application.

Exemple pour un logiciel d'annuaire



Architecture des objets (1)

- Pour qu'un objet puisse envoyer un message à un autre objet, il faut qu'il connaisse cet autre objet.
- Un objet connaît un autre objet s'il possède une référence sur cet autre objet.
- Une référence sur un objet est un attribut qui identifie l'objet référencé.
- Le programmeur doit déterminer les références entre les objets : quel objet connaît quel objet ?

Architecture des objets (2)

- La fenêtre principale contient 3 champs d'entrée. Comme elle doit y accéder, elle doit connaître ces champs.
- La fenêtre principale doit donc avoir trois attributs qui sont des références sur les champs d'entrée.
- La fenêtre principale crée les fenêtres résultats. Lorsque l'application se terminera, elle devra faire disparaître toutes ces fenêtres.
- La fenêtre principale doit donc avoir un **attribut** qui est une **liste de références sur les fenêtres résultats**.

Architecture des objets (3)

- Une fenêtre résultat peut être fermée en cliquant sur son bouton *Close*.

- Mais il ne faut pas oublier que la fenêtre principale maintient une liste des fenêtres résultats. Si une fenêtre résultat est fermée par son bouton *Close*, il faudra qu'elle prévienne la fenêtre principale.

- Chaque fenêtre résultat doit donc avoir un **attribut** qui est une **référence sur la fenêtre principale**.

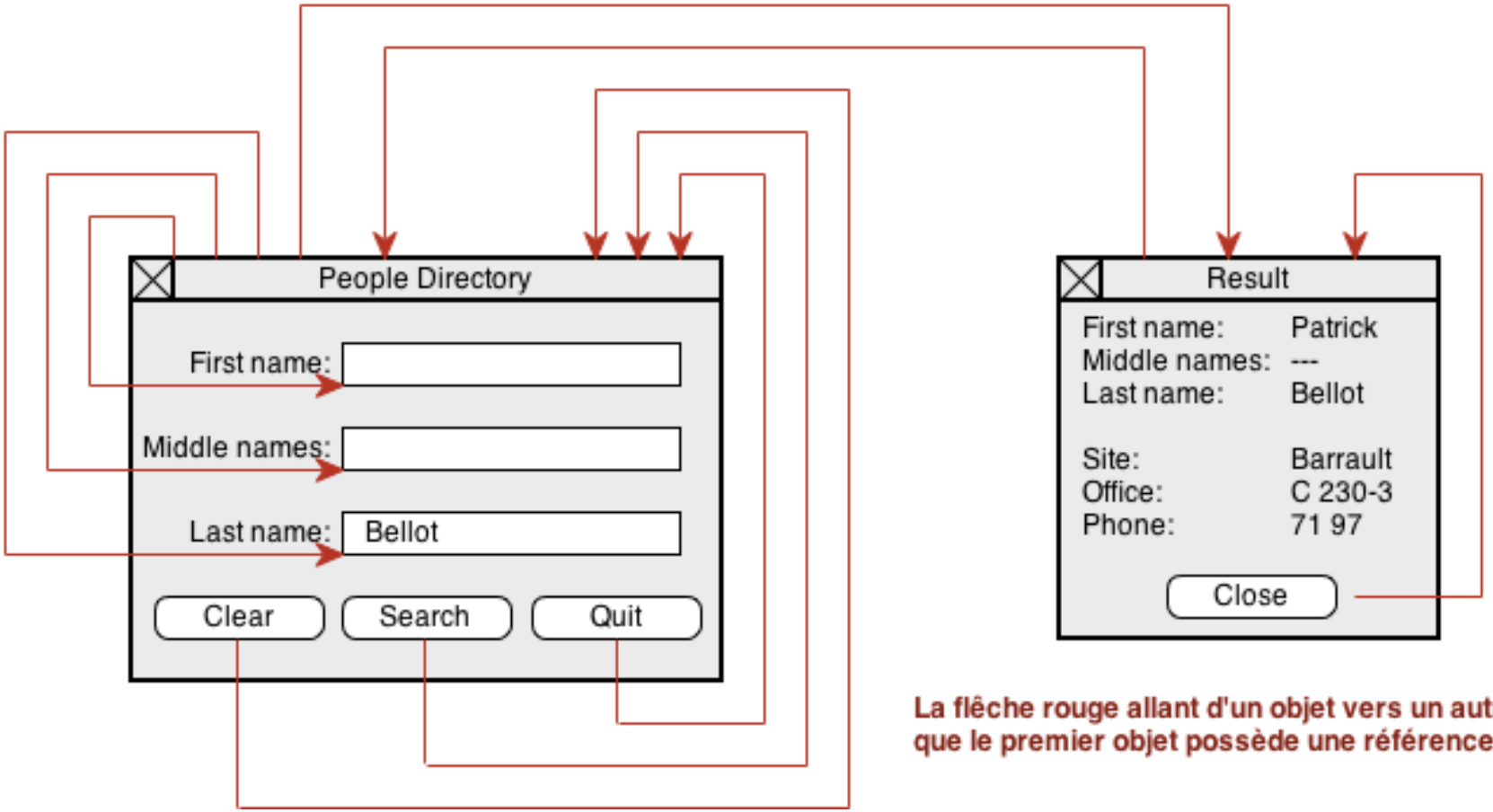
Architecture des objets (4)

- Tous les boutons déclenchent des actions: *Clear*, *Search* et *Quit* dans la fenêtre principale, et *Close* dans les fenêtres résultat.

- Lorsque l'utilisateur cliquera sur l'un de ces boutons, le plus simple est que le bouton demande à sa fenêtre de faire le travail.

- Donc, chaque bouton doit avoir un **attribut** qui est une **référence sur la fenêtre qui le contient**.

Visualisation de l'architecture

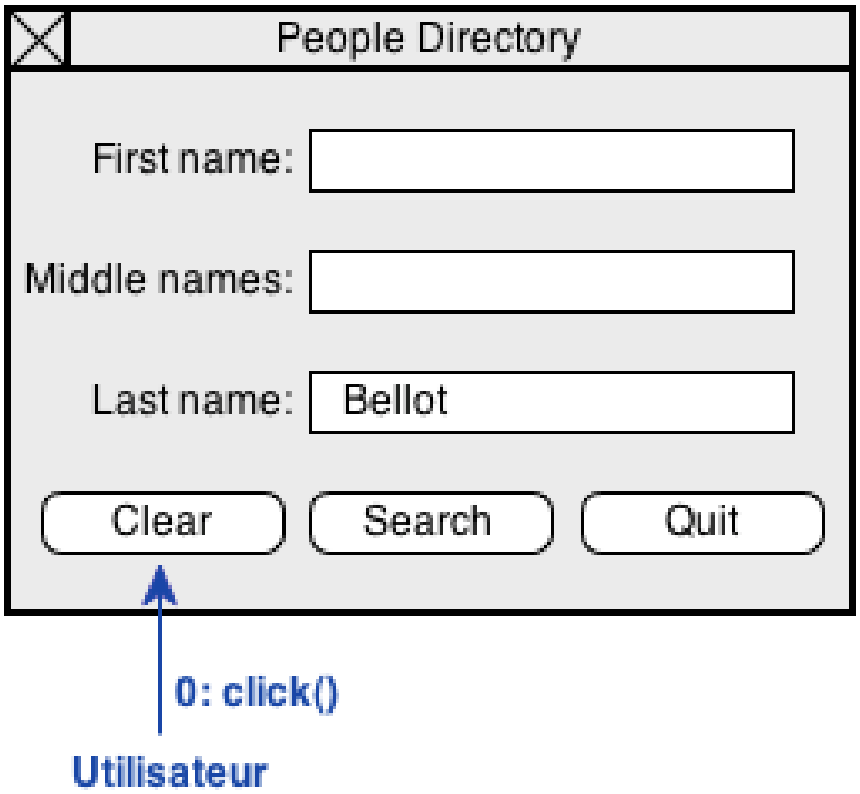


La flèche rouge allant d'un objet vers un autre signifie que le premier objet possède une référence sur le second.

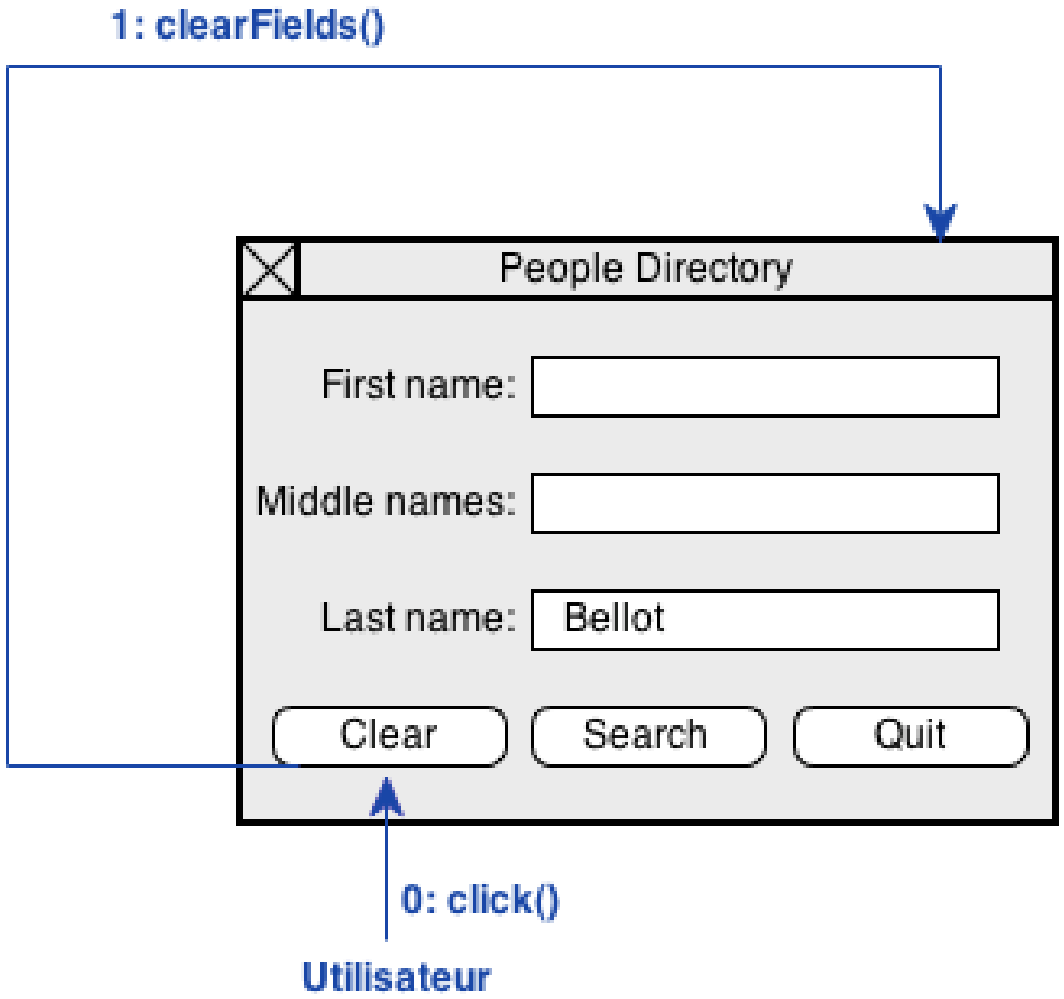
Cascade de messages

- Que se passe-t-il quand l'utilisateur clique sur le bouton *Clear* ?
  - Il faut vider les trois champs d'entrée.
- Lorsque l'utilisateur clique sur le bouton *Clear*, le bouton reçoit un message **`click().`**
- Le bouton se contente alors d'envoyer le message **`clearFields()`** à la fenêtre qu'il connaît.
- Et la fenêtre envoie le message **`clear()`** à chacun des champs d'entrée.
- A la réception du message **`clear()`**, chaque champ d'entrée efface le texte qu'il contient. Et il renvoie un message **`ok`** à la fenêtre qui le contient.
- Quand la fenêtre a reçu les trois messages **`ok()`** des trois boutons, elle renvoie un message **`ok()`** au bouton.

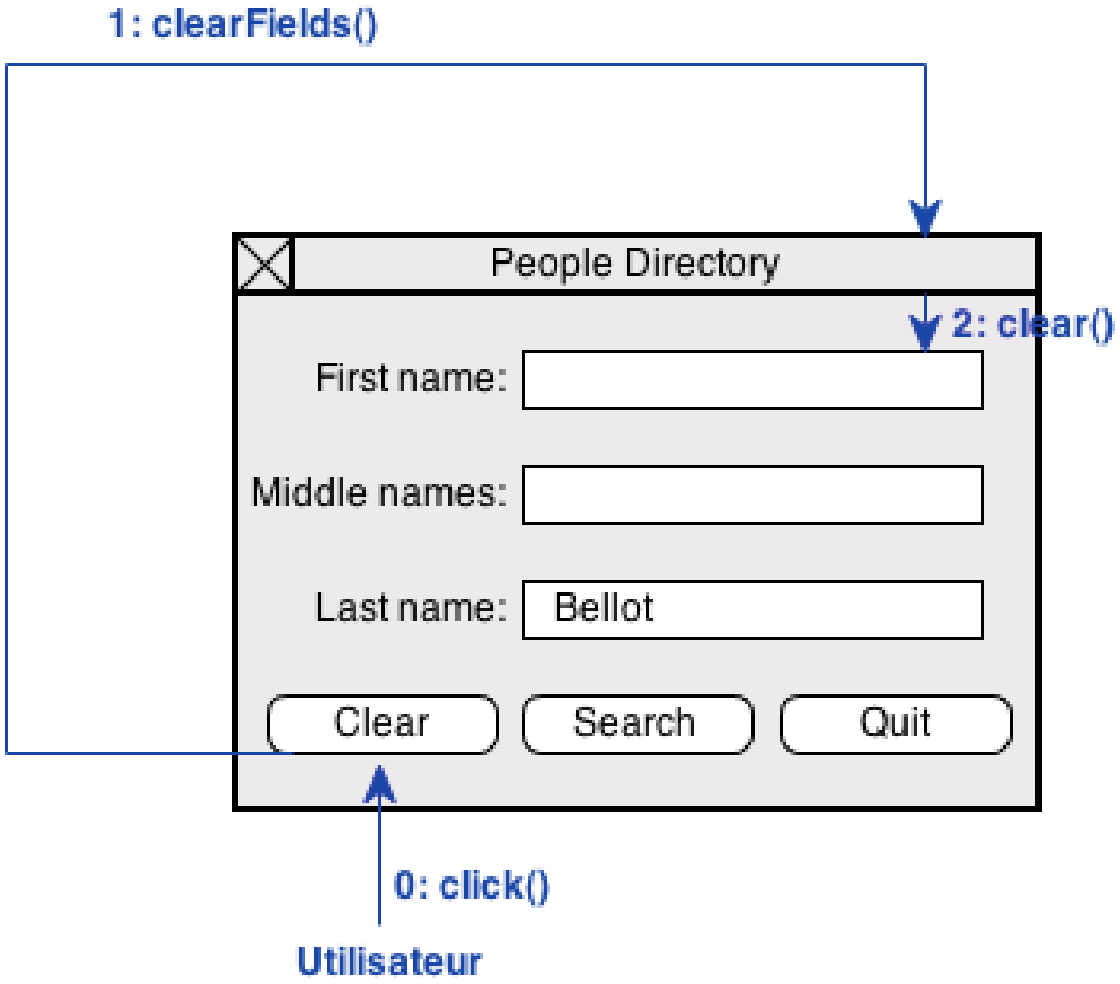
Le bouton Clear



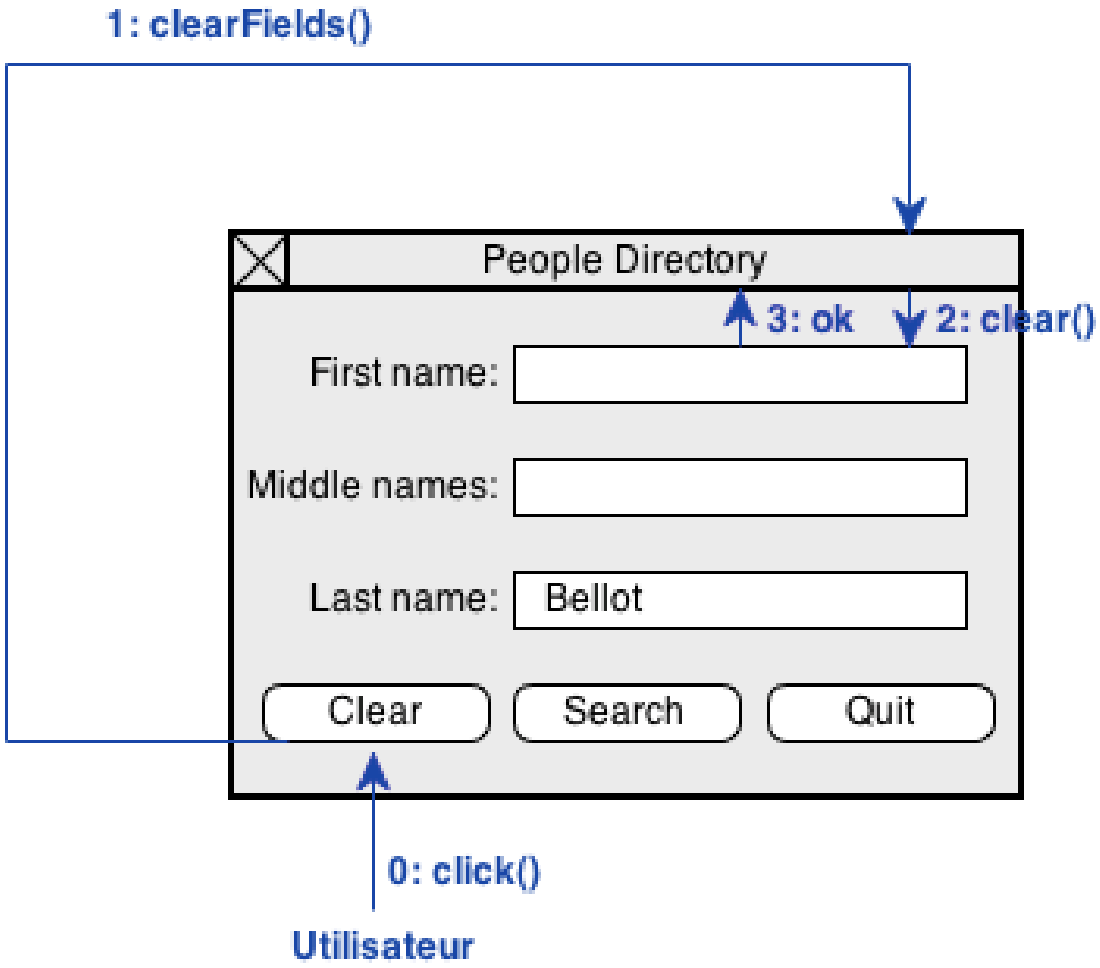
Le bouton Clear



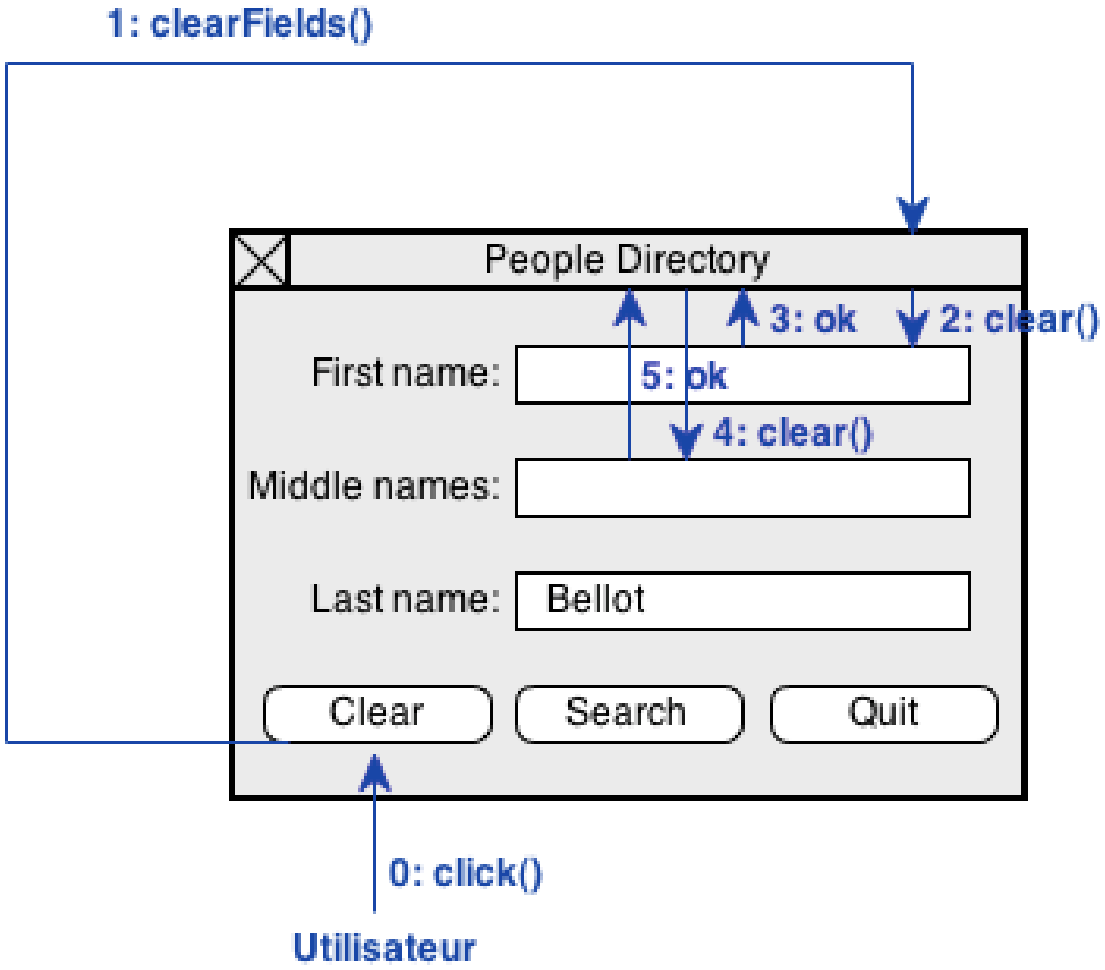
Le bouton Clear



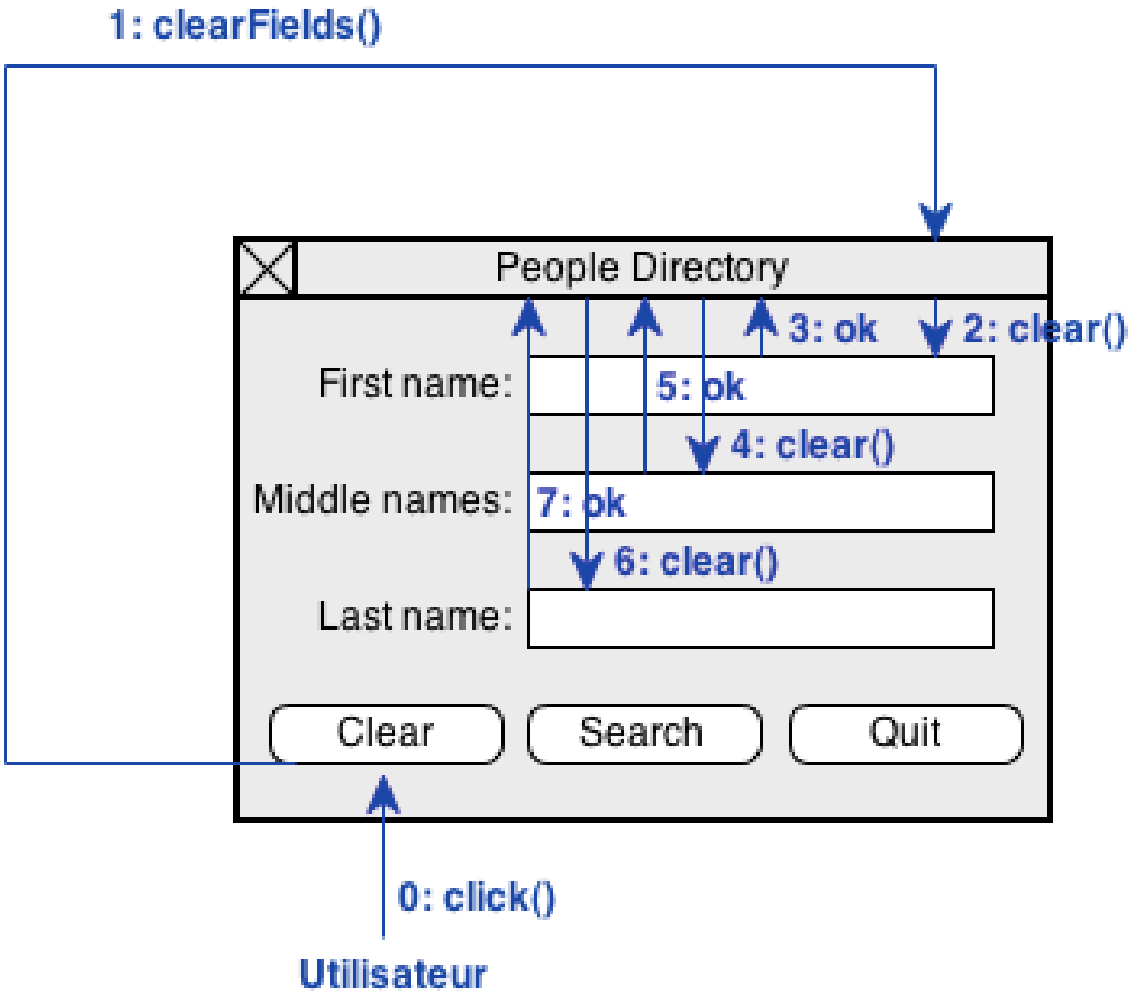
Le bouton Clear



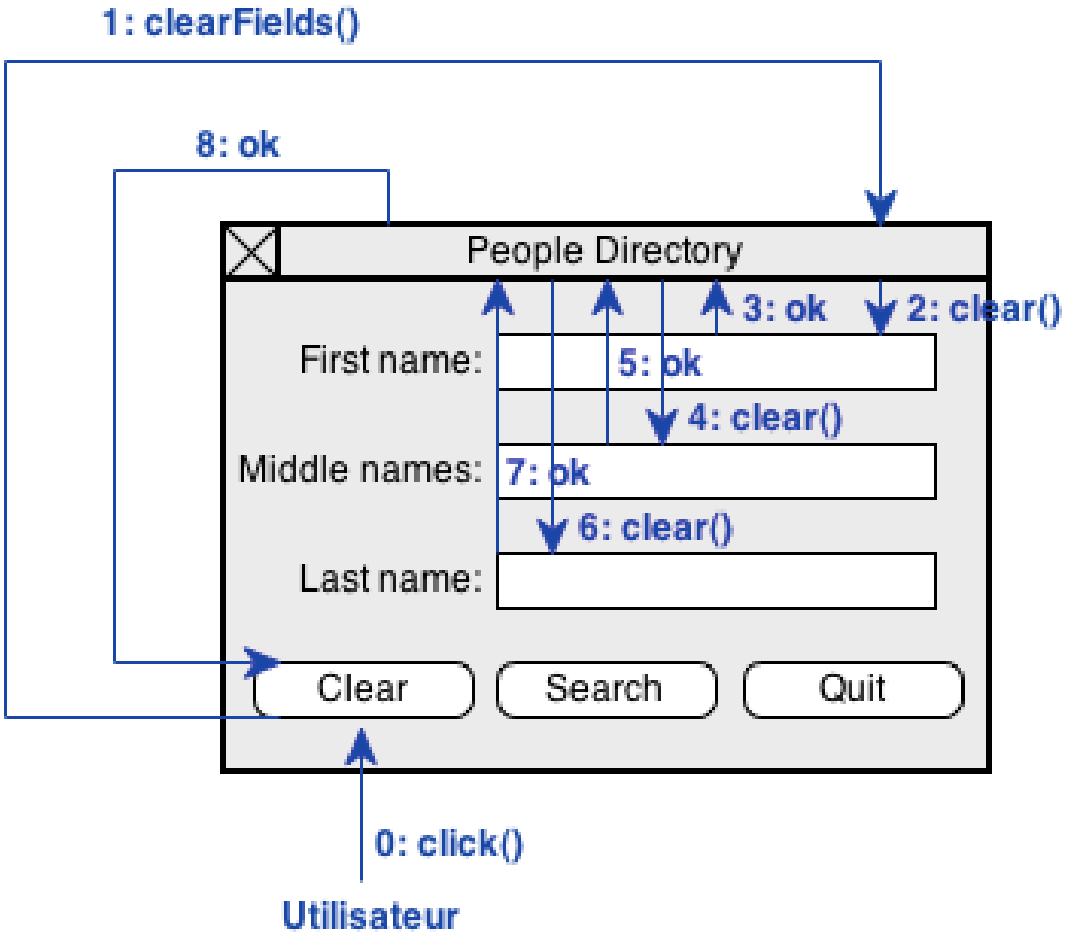
Le bouton Clear



Le bouton Clear



Le bouton Clear



Avantages de l'OO

- Le programmeur doit programmer les méthodes qui déterminent les réponses aux messages reçus par les objets.

- Ces méthodes sont généralement très simples.

- Exemple**:** la méthode correspondant au message **`click()`** pour le bouton **Clear** :
  - Se redessiner enfoncé
  - Envoyer le message **`clearFields()`** à la fenêtre
  - Attendre la réponse **ok**
  - Se redessiner relevé.

Avantages de l'OO

- Chaque objet possède ses propres **méthodes** pour répondre aux messages.
  - Les algorithmes sont morcelés en parties très simples réparties sur les objets.

- Chaque objet est responsable des interactions avec son environnement qui est composé des objets qu'il connaît et des objets qui le connaissent.

- Les algorithmes de chaque objet sont plus faciles à concevoir et plus concis qu'un algorithme global qui tenterait de tout gérer.

Deux principes

- Le principe de **non-intrusion** : on évite de travailler sur un objet depuis l'extérieur de l'objet.
  - Ainsi le bouton **Clear** ne va pas vider lui-même les champs d'entrée dans la fenêtre mais demandera plutôt à la fenêtre de le faire.
  - Intérêt : si l'on rajoute un champ de saisie dans la fenêtre, il ne faudra modifier que le programme de la fenêtre.

- Le principe de **délégation** : quand un travail doit être fait, on demande si possible aux autres objets de le faire.
  - Intérêt : on encapsule et on localise le code dans un nombre restreint de classes.

Flot d'exécution

- On appelle **flot d'exécution d'un programme** la suite des actions exécutées par le programme.

- Comme nous le verrons, ces actions peuvent être:
  - Des envois de messages
  - Des calculs de valeurs
  - Des entrées-sorties
  - Lire et écrire des données en mémoire
  - Etc.

- Ces actions sont exécutées **séquentiellement**.

En résumé…

- Les **objets** sont des entités informatiques qui communiquent par envois de messages.
- Les objets contiennent des valeurs appelées des **attributs**. Parmi ces attributs, on peut trouver des **références** sur d'autres objets.
- Une référence sur un objet permet de lui envoyer un **message**.
- Pour chaque type de message que l'objet peut recevoir, l'objet connaît une **méthode** associée au type de message.
- Cette méthode est une procédure qui est **exécutée par l'objet** lorsqu'il reçoit le type de message associé.

En résumé…

- Un type d'objet est décrit par une classe.
  - La classe décrit les attributs: nom et type de valeur
  - La classe décrit les méthodes utilisées pour répondre aux messages.

- Le programmeur peut créer des objets à partir de la classe. C'est le processus d'**instanciation**. On dit que les objets sont des **instances** de la classe.

- Il existe trois grandes écoles de langages OO :
  - Smalltalk – Java
  - C++, Ada
  - Eiffel