



Programmation orientée objet en  
Java

## Introduction au paradigme Orientée Objet (OO)

**Dominique Blouin**

**Télécom Paris, Institut Polytechnique de Paris**

**[dominique.blouin@telecom-paris.fr](mailto:dominique.blouin@telecom-paris.fr)**

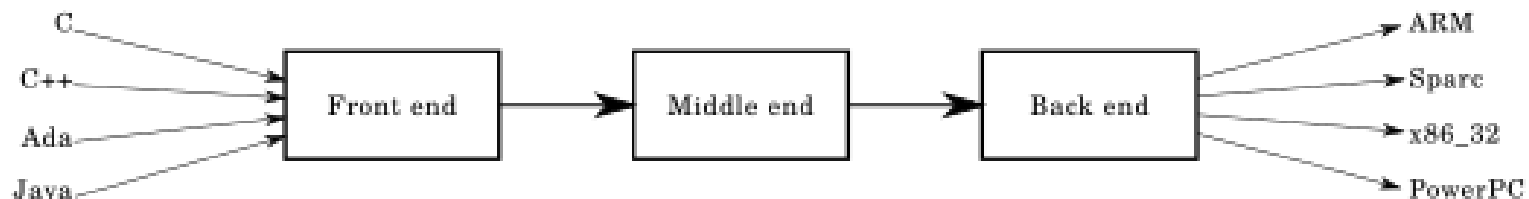


# Objectifs d'apprentissage de cette période

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

# Langages de programmation

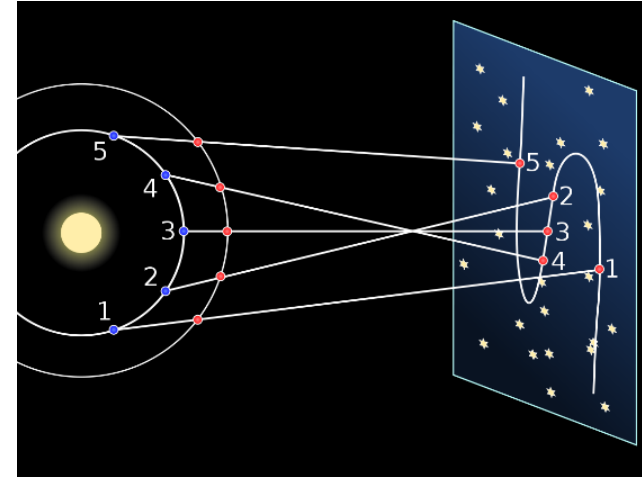
- Un langage de programmation permet au programmeur d'écrire son programme avec des concepts de haut niveau.
  - Par exemple, tous les langages proposent la notion de liste de données numériques.
- Le **compilateur** traduit ces concepts de haut niveau en instructions pour l'UAL (Unité Arithmétique et Logique).



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

# Paradigmes de programmation

- Paradigme : représentation du monde, manière de voir les choses, modèle cohérent du monde qui repose sur un fondement défini (matrice disciplinaire, modèle théorique, courant de pensée)
  - Source : Wikipédia
- Paradigmes scientifiques :
  - Géocentrisme versus héliocentrisme en astronomie.
  - Importance des **changements** de paradigmes.
- Paradigmes d'ingénierie : manière de **solutionner un problème**.
  - Concernent tout le cycle de vie du système (conception, implémentation, maintenance, etc.).
  - Concernent également l'**environnement** dans lequel l'ingénierie se fait :
    - Méthodes et outils tels que modèles et leurs langages, processus de développement, etc.
    - Par exemple, la méthode Agile peut être vue comme paradigme d'ingénierie.
- Caractéristiques communes entre paradigmes scientifiques et d'ingénierie :
  - Moyens de **caractériser** un ensemble **artefacts** utilisés dans un **environnement** pour solutionner un problème.
- Paradigmes de programmation :
  - Caractérisent un **langage de programmation** (artefact) par sa **syntaxe** et sa **sémantique**.



# Types de paradigmes de programmation

- Paradigmes impératifs (ou procéduraux) : programmes constitués de **commandes** dont l'exécution est déterminée par des **structures de contrôle**.
  - 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.
- On doit dire à l'ordinateur **comment** faire.
- C'est le 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

- **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.
  - Correspondance directe avec les **objets du réel**.

# Paradigmes déclaratifs

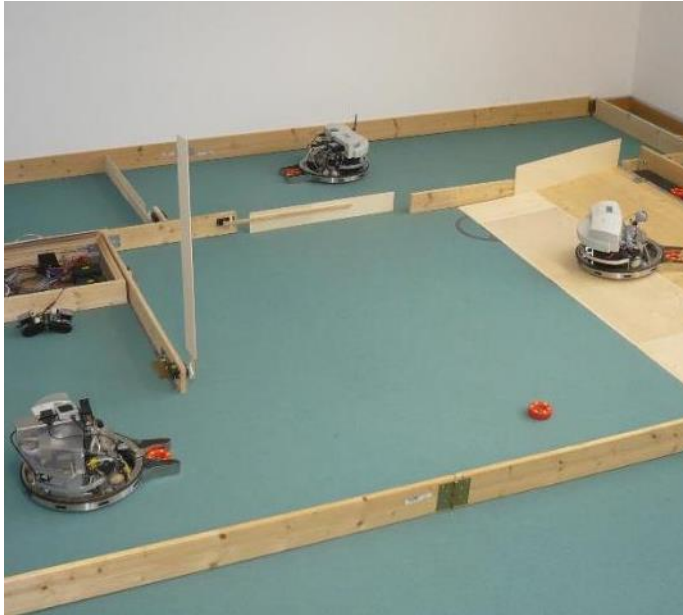
- Décrivent un **problème** (souvent sous forme de formules logiques).
  - L'exécution du programme consiste à trouver une ou des **solution(s)** au problème donné.
- Exemples :
  - SQL : un select dit **quelles données sont requises** (le quoi ou le problème) et le moteur d'exécution trouve un parcours des tables (le comment ou une solution) pour récupérer les données.
  - Prolog ...
- Extensions des paradigmes déclaratifs :
  - Programmation fonctionnelle.
  - Programmation par contraintes.
  - Programmation orientée graphes
  - Etc.

# Naissance de l'OO

- La programmation OO a été introduite par le langage **Simula** (SIMple Universal LAnguage) 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.
- Un programme centralisé qui pilote les robots en modifiant une structure de données représentant l'état de **tous les robots** et de **l'environnement** n'est pas impossible mais très difficile.
- La solution : l'orienté objet.



# Principes de l'OO



- Décrire chacun des éléments du problème par des **classes**.
  - Cela concerne les robots, mais aussi tous les éléments du problème, par exemple l'usine qui contient les robots, les salles, les portes, les machines de production, etc.
- Une classe décrit les **données** contenues dans un 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.
- C'est similaire à la description d'une **structure de données** telle que vous l'avez apprise en **langage C** (INF107).
- 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 contrôle centralisé** qui gère l'ensemble des actions de tous les robots et de leur environnement.

# 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 de tous ses éléments**.
- Et il est tout aussi difficile d'imaginer un contrôle centralisé devant gérer cette interface graphique.

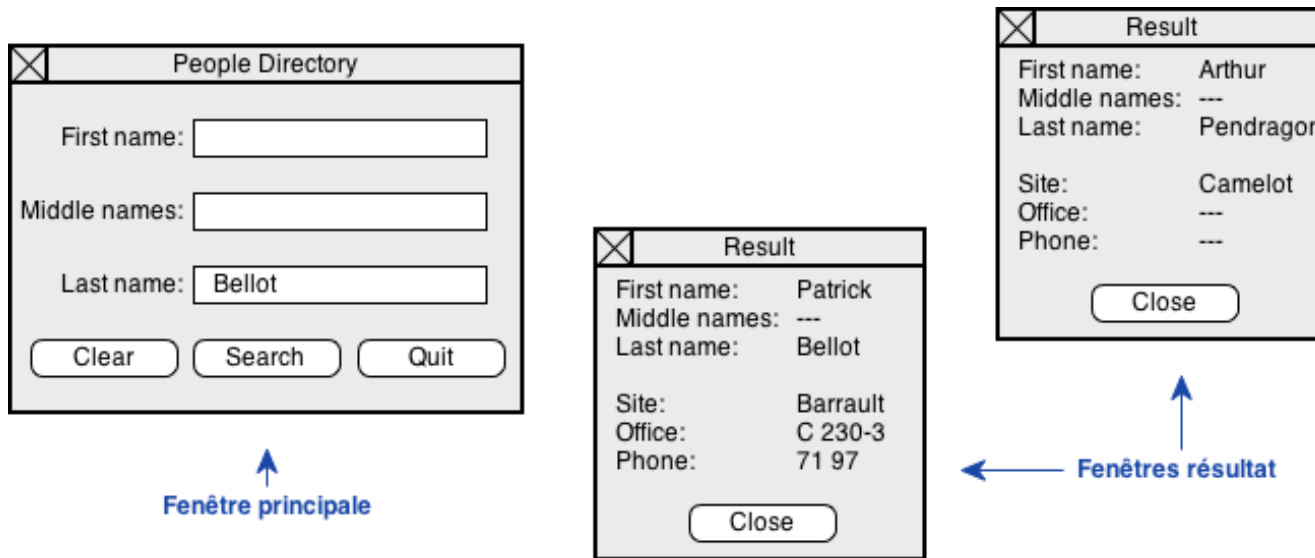
# L'approche OO

- Dans l'approche OO des interfaces graphiques, chaque élément apparaissant sur l'écran est un **objet**.
- 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.

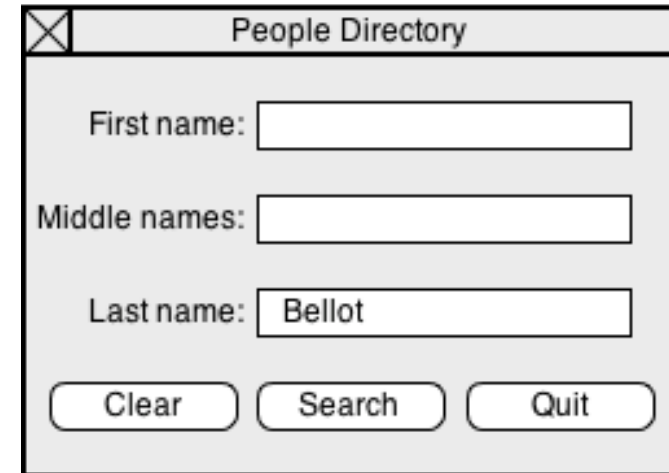
# Architecture des objets (1)

- 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 doit connaître quel objet ?
  - Environnement de l'objet.



## Architecture des objets (2)

- La fenêtre principale contient trois **objets** de type **champs d'entrée**.
- Comme elle doit pouvoir 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 doit pouvoir **créer** des fenêtres résultats.
- Lorsque l'application se terminera, elle devra également faire **disparaître** ces fenêtres.
  - La fenêtre principale doit donc avoir un **attribut** qui est une **liste de références** sur les fenêtres résultats.



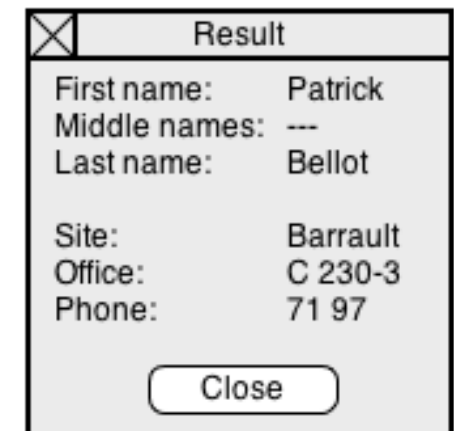
People Directory

First name:

Middle names:

Last name:

Clear Search Quit



Result

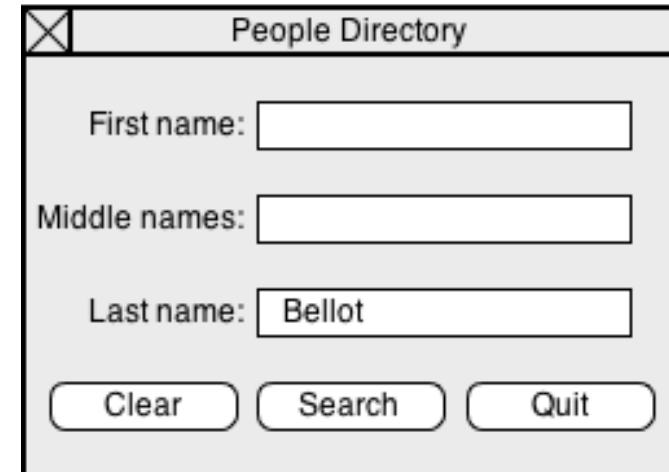
First name: Patrick  
Middle names: ---  
Last name: Bellot

Site: Barrault  
Office: C 230-3  
Phone: 71 97

Close

## 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 **préviene** (envoyer un message à) la fenêtre principale.
- Chaque fenêtre résultat doit donc avoir un **attribut** qui est une **référence sur la fenêtre principale**.



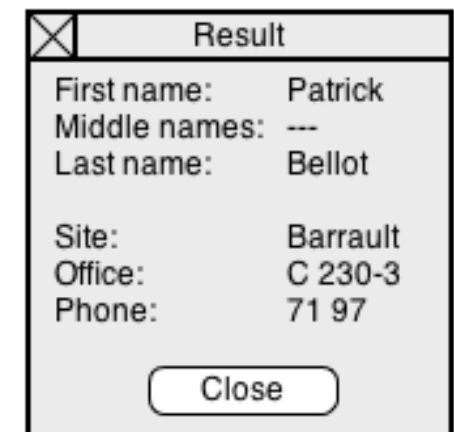
People Directory

First name:

Middle names:

Last name:

Clear Search Quit



Result

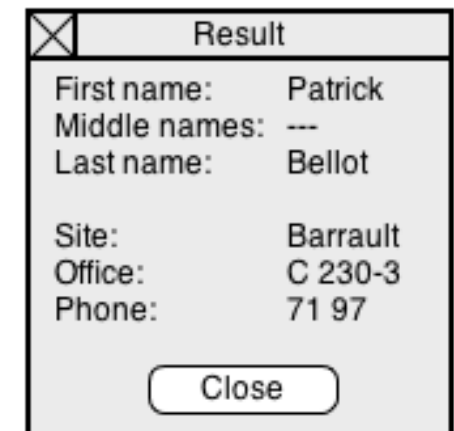
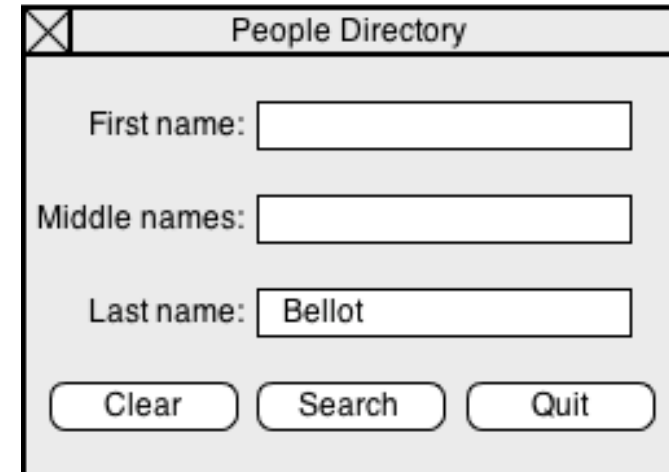
First name: Patrick  
Middle names: ---  
Last name: Bellot

Site: Barrault  
Office: C 230-3  
Phone: 71 97

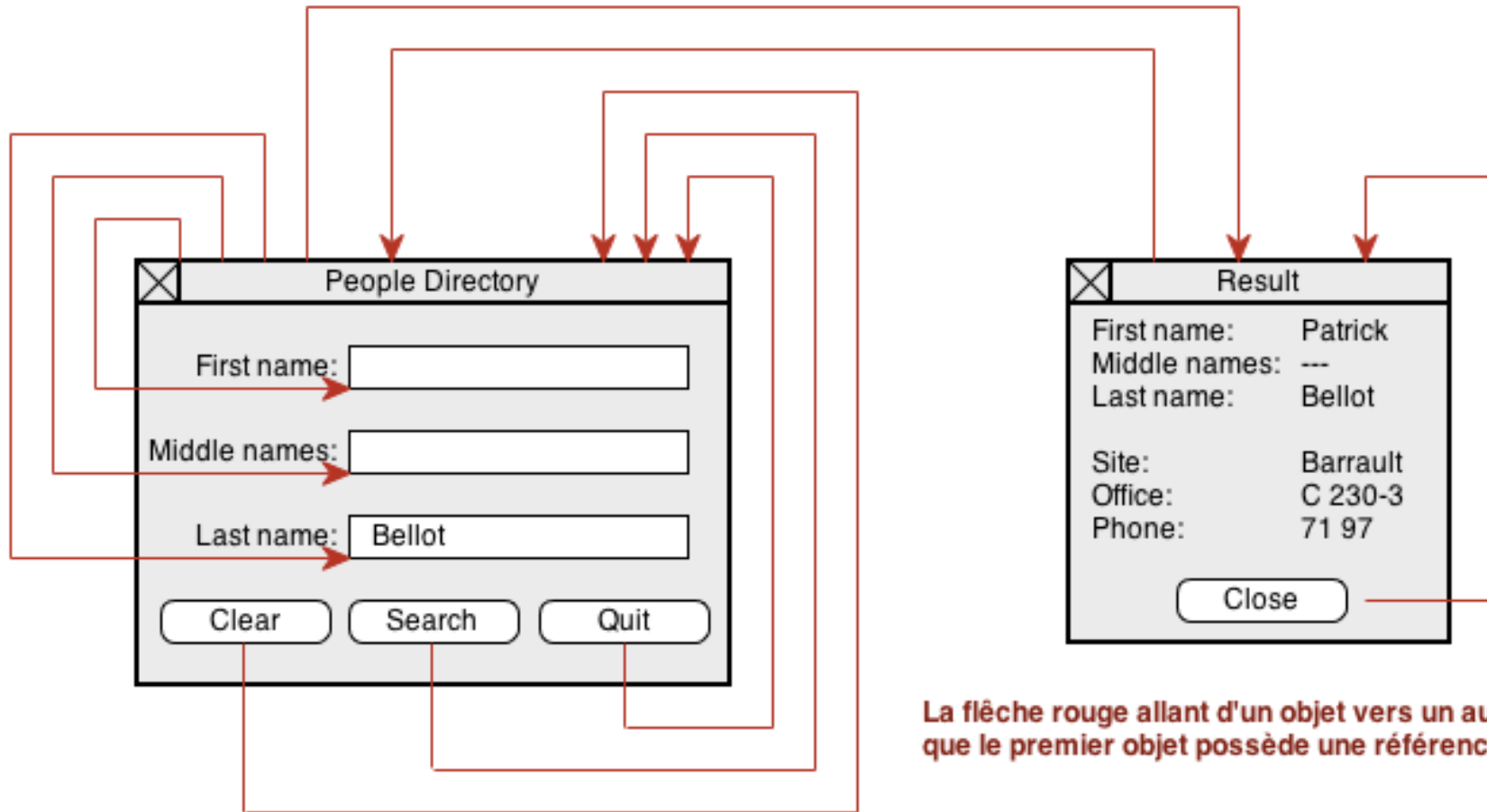
Close

## Architecture des objets (4)

- Tous les boutons déclenchent des **actions** :
  - *Clear*, *Search* et *Quit* dans la fenêtre principale.
  - *Close* dans les fenêtres résultat.
- Lorsque l'utilisateur cliquera sur l'un de ces boutons, le plus simple est que le bouton **demande** à la **fenêtre qui le contient** de faire le travail.
  - Par exemple, *Clear* **demande** à la fenêtre principale d'effacer **ses** champs.
- Donc, chaque bouton devra avoir un **attribut** qui sera une **référence** sur la fenêtre qui le contient.



# Visualisation de l'architecture des objets



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 doit-il se passer lorsque l'utilisateur clique sur le bouton *Clear* ?
  - Il faut alors **effacer** les trois champs d'entrée.
- Lorsque l'utilisateur cliquera sur le bouton *Clear*, le bouton recevra un message `click()`.
- Le bouton se contente alors d'envoyer le message `clearFields()` à la **fenêtre qui le contient**.
- Et cette fenêtre enverra le message `clear()` à chacun de ses champs d'entrée.
- A la réception du message `clear()`, chaque champ d'entrée **efface** son texte et renvoie un message `ok` à la fenêtre qui le contient.
- Lorsque la fenêtre aura reçu les trois messages `ok` des trois boutons, elle renverra un message `ok` au bouton *Clear*.

# Illustration des messages envoyés entre les objets : le bouton *Clear*

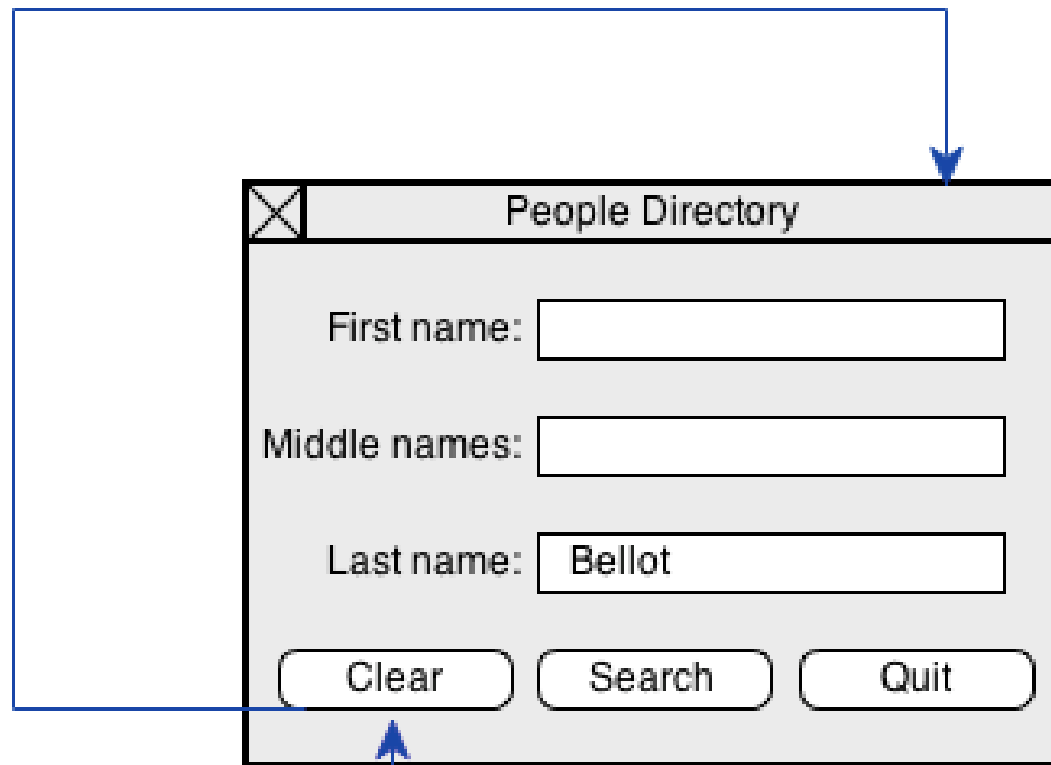
The image shows a Java Swing window titled "People Directory". It contains three text input fields: "First name:" (empty), "Middle names:" (empty), and "Last name:" (containing "Bellot"). Below the fields are three buttons: "Clear", "Search", and "Quit". A blue arrow points from the text "Utilisateur" and "0: click()" to the "Clear" button.

0: click()

Utilisateur

# Le bouton Clear

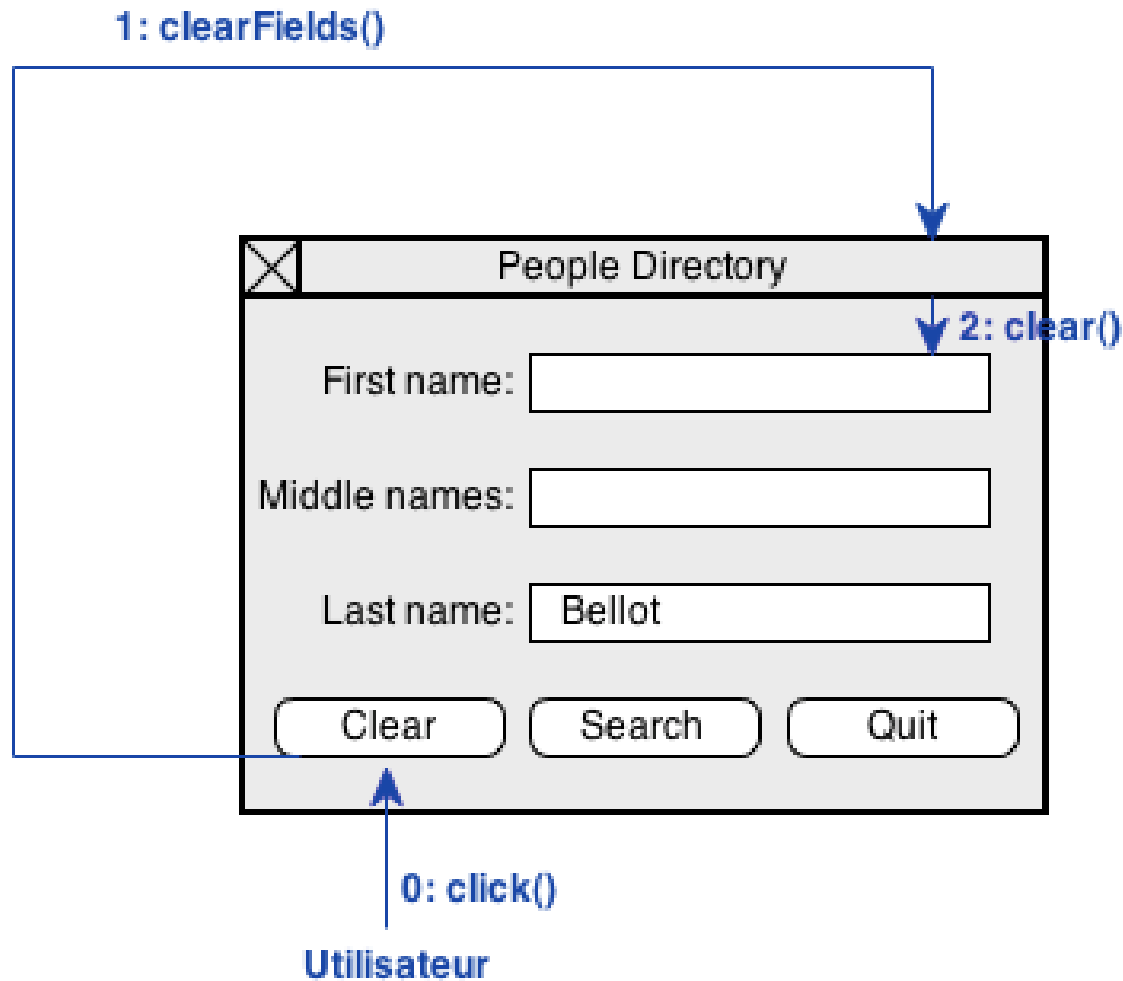
1: clearFields()



0: click()

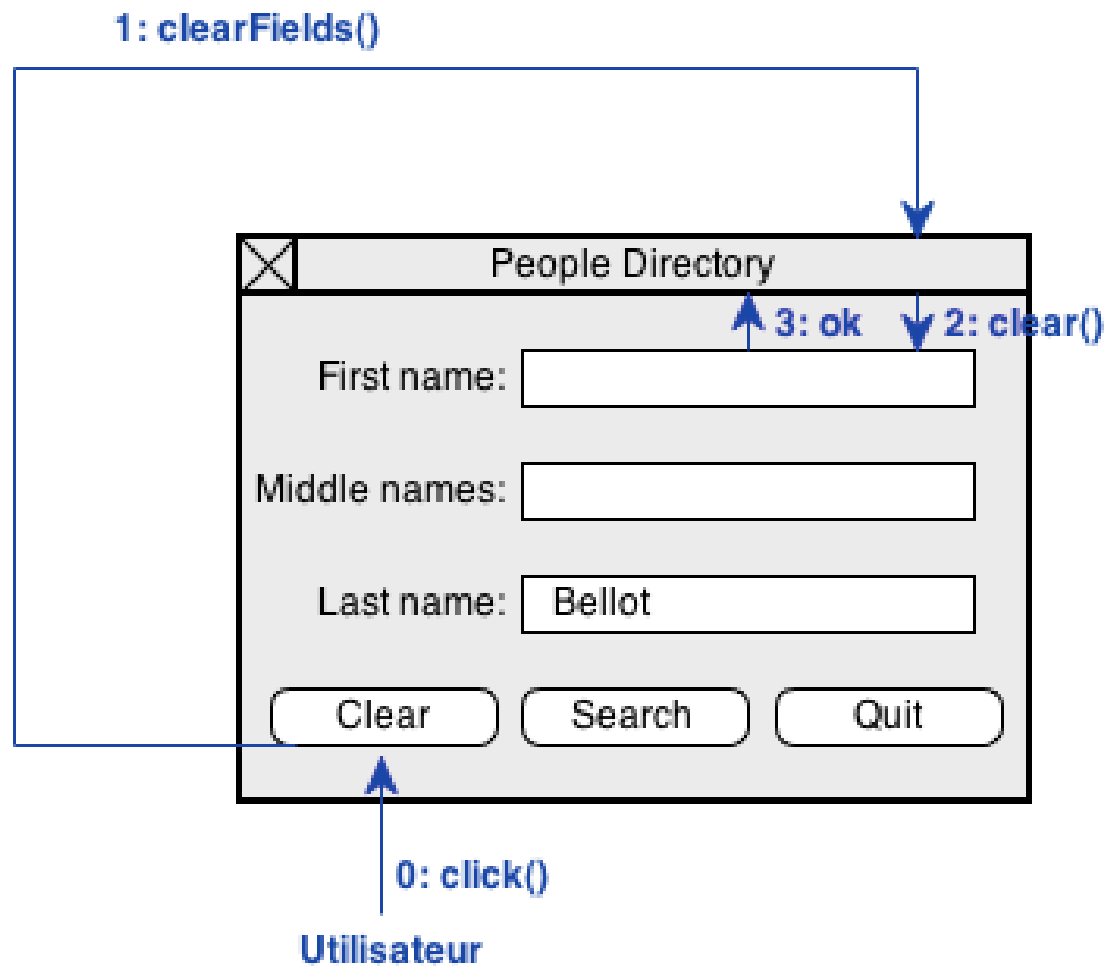
Utilisateur

# Le bouton Clear

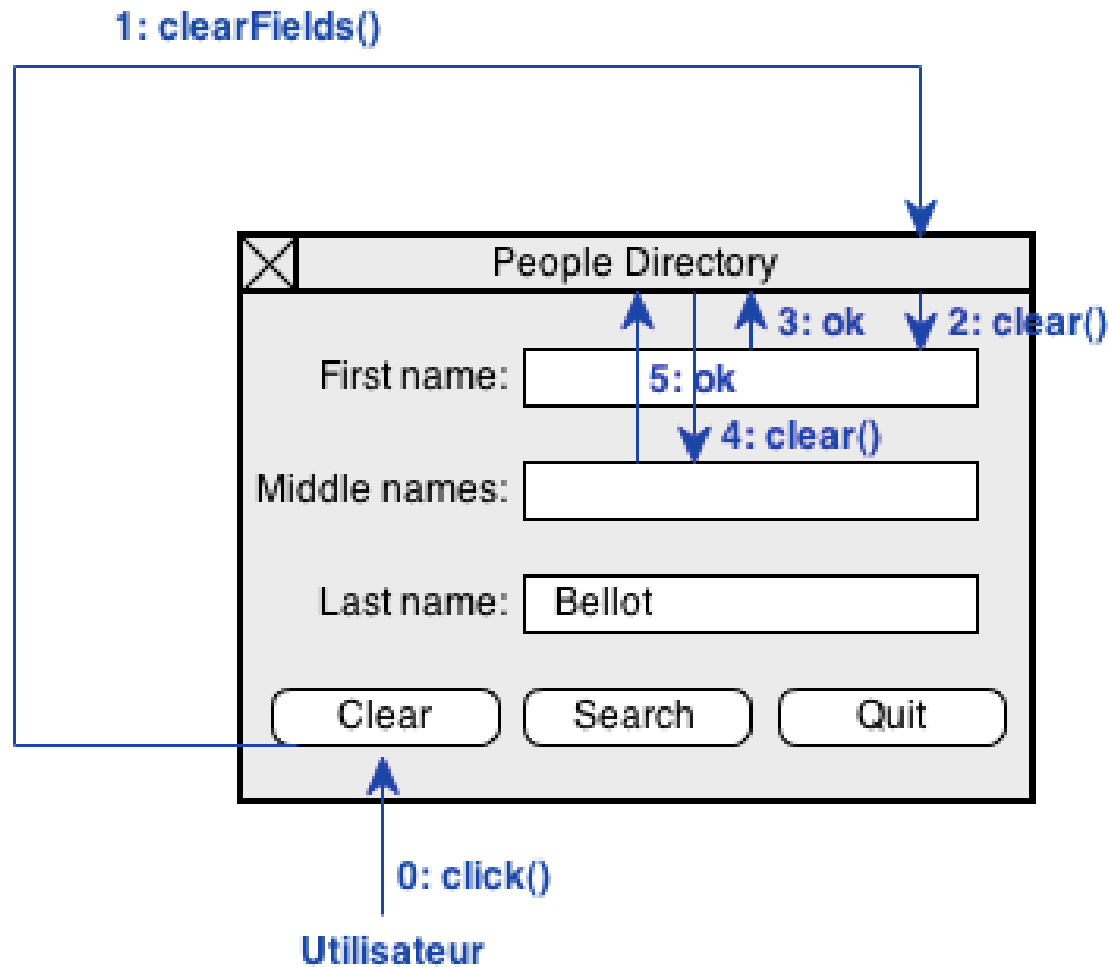




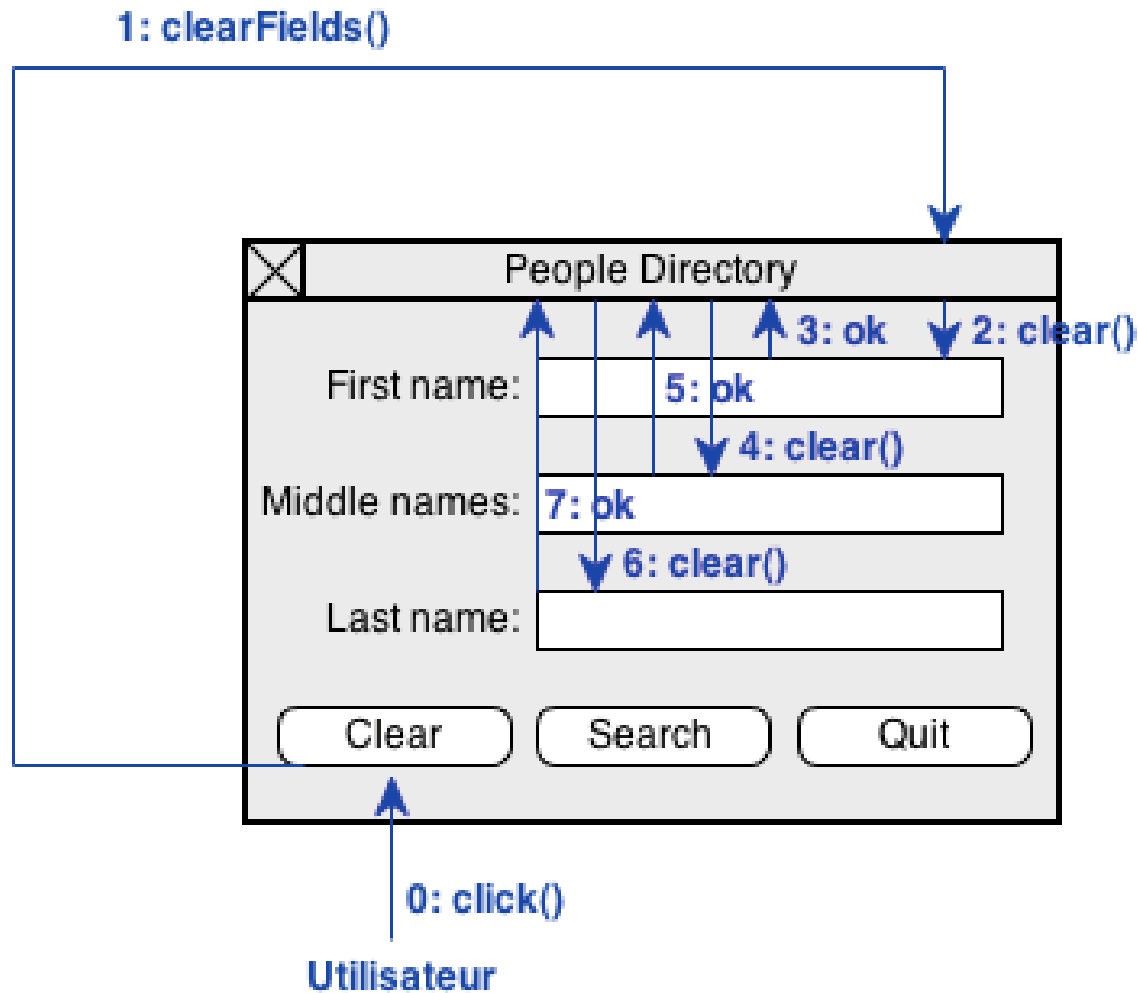
# Le bouton *Clear*



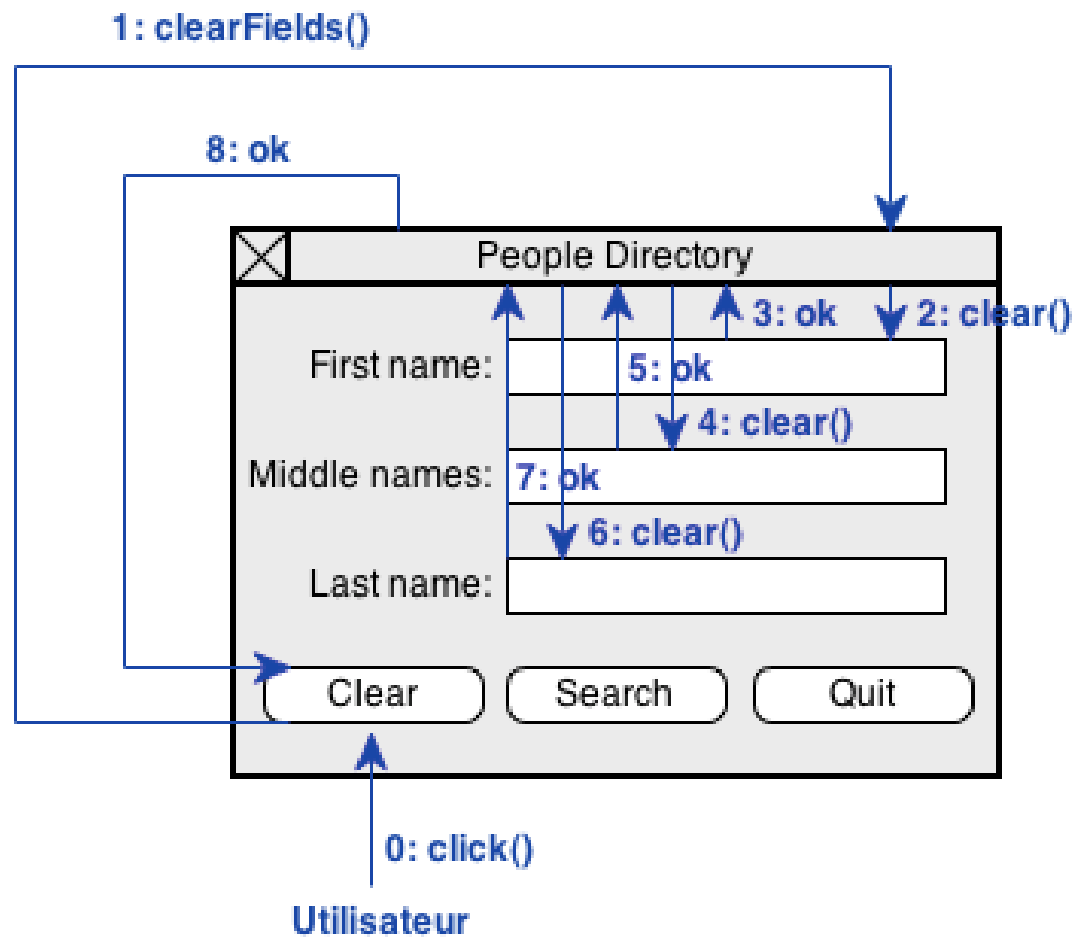
# Le bouton *Clear*



# Le bouton *Clear*



# Le bouton *Clear*



## Avantages de l'OO

- Pour chaque objet, il suffit de 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 principale.*
  - *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 plus **simples** qui sont **réparties** parmi les objets.
- Chaque objet est **responsable** des interactions avec son **environnement**.
- Cet environnement est composé des objets que l'objet **connaît** et des objets qui **connaissent** l'objet.
- Les algorithmes de chaque objet sont plus faciles à concevoir car plus **simples** qu'un algorithme **global** qui tenterait de **tout gérer** en même temps.

# Deux principes importants de l'OO : non-intrusion et délégation

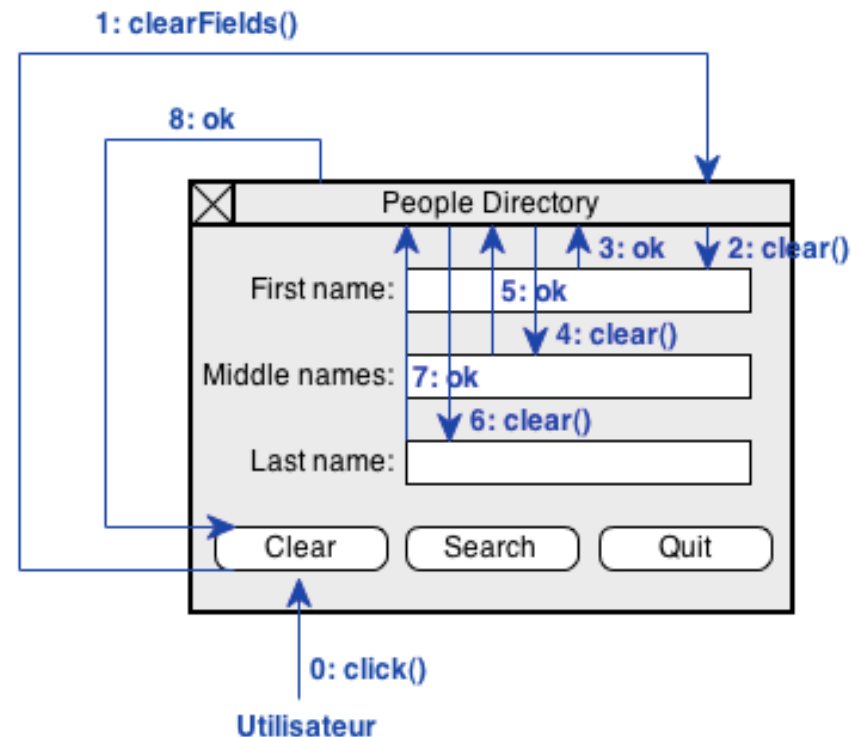


- Principe de **non-intrusion** : on évite de travailler sur un objet depuis l'**extérieur** de l'objet.
- Par exemple, si j'ai besoin d'emprunter un téléphone à quelqu'un, je ne vais pas **directement** le prendre à la personne (**non intrusif**).
- Je vais plutôt lui **demander** de me **prêter** son téléphone.
  - Envoyer un **message** à la personne et lui **déléguer** le travail de me **prêter** le téléphone.
- Ainsi, la personne possédant le téléphone pourra le **préparer** avant de me le donner.
  - Par exemple, le déverrouiller s'il est protégé par un mot de passe.



## Exemple avec le logiciel d'annuaire

- Ainsi le bouton *Clear* ne va pas **vider lui-même** les champs d'entrée dans la fenêtre principale, mais il **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 la méthode de la fenêtre principale, qui elle **connaît** ses champs.
- 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 de données.
  - Lire et écrire des données en mémoire.
  - Etc.
- Ces actions sont exécutées **séquentiellement**.

## En résumé...

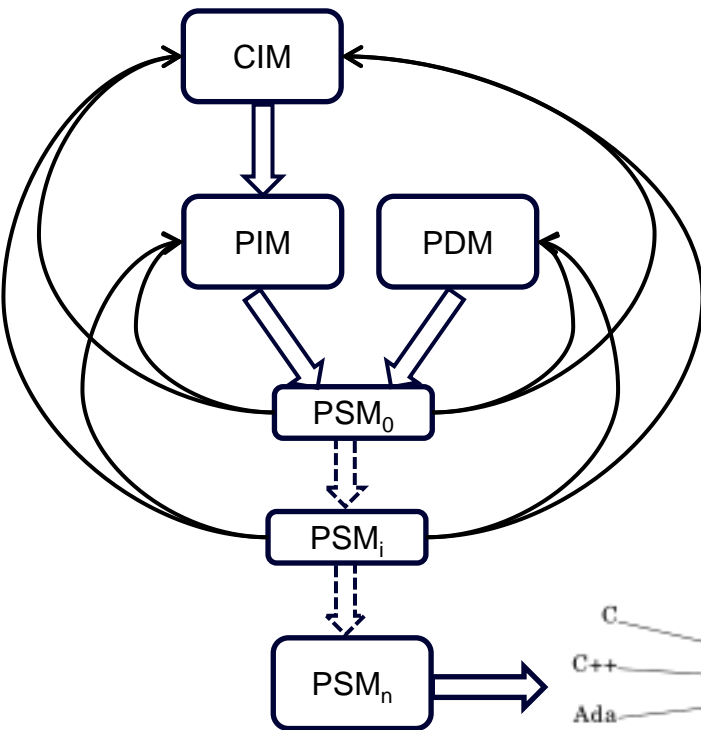
- Les **objets** sont des entités informatiques qui communiquent par **envois de messages**.
- Les objets contiennent des **valeurs** appelées des **attributs**. Parmi ces attributs, on peut trouver des **références** sur d'**autres objets**.
  - Environnement de l'objet ; les objets qu'il connaît.
- 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 **fonction** ou une **procédure** ou (voir langage C) 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**.
- Deux grandes catégories de langages de programmation OO :
  - Langages à base de classes : Smalltalk, Java, C#, C++, etc.
  - Langages à base de prototypes : JavaScript, Lua
- Au-delà des langages de programmation, l'OO est essentiel pour la **modélisation**.

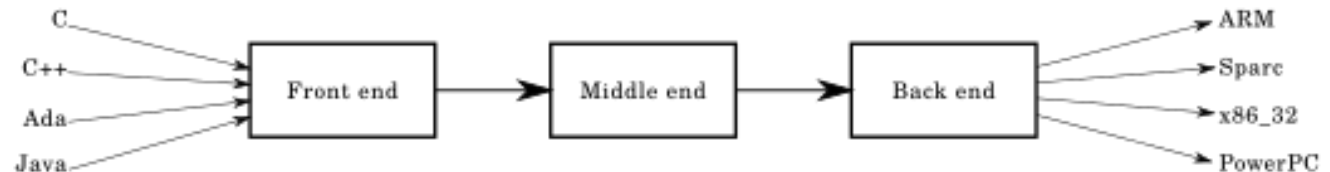
# Modéliser au lieu de programmer...

- Concevoir le système avec des **modèles** (de plus haut niveau d'abstraction), **vérifier** les modèles de la conception et **générer le code automatiquement** !



- CIM (Computation-Independent Model) : décrit les exigences du système ainsi que sa manière de fonctionner dans son environnement, alors que les détails de la structure de l'application et de sa réalisation ne sont **pas encore déterminés**.
- PIM (Platform-Independent Model) : décrit les détails du système sans montrer les détails **spécifiques à une plateforme d'exécution** ou à une technologie particulière.
- PDM (Platform Description Model) : décrit la **plateforme d'exécution** (processeurs, mémoires, bus, etc.)
- PSM (Platform-Specific Model) : décrit le logiciel **déployé** sur la plateforme d'exécution.

- $\rightarrow$  : Analyse et modifications itératives des modèles de conception
- $\Rightarrow$  : Transformation de modèles



- Une grande partie des langages de modélisation sont **orientés objet** :
  - Par exemple, UML, SysML, AADL, Modelica, etc.