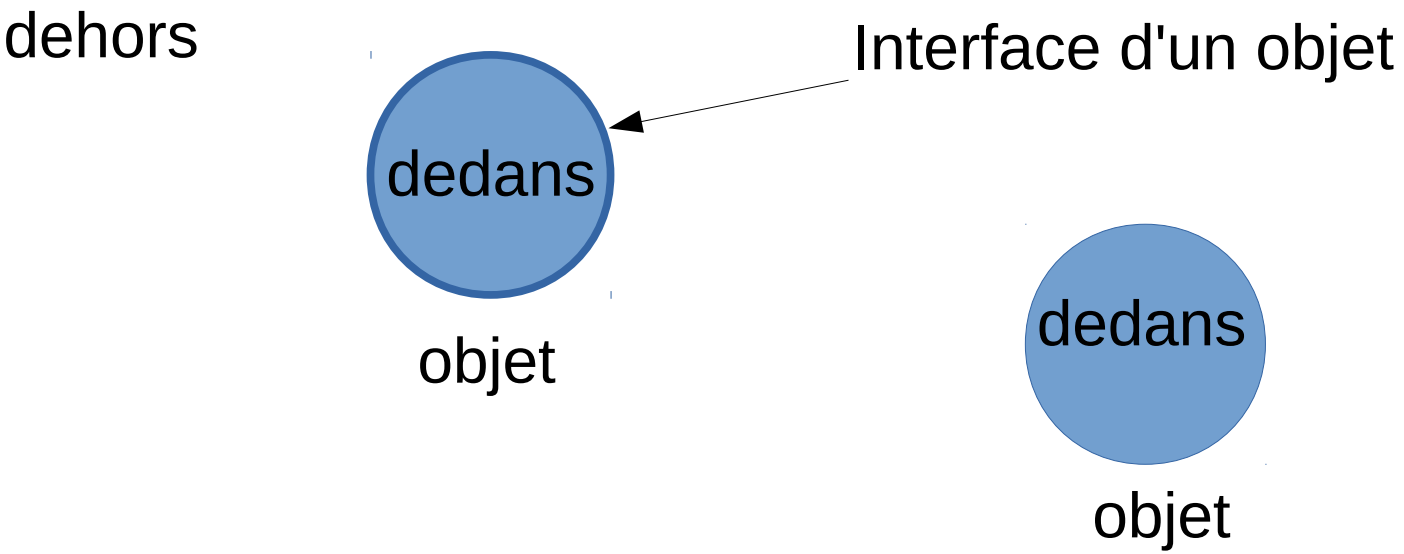
Qu'est ce qu'un objet ?

- Un objet définit un **en-dedans** et un **en-dehors**
- L'idée est que le **dehors** ne doit pas connaître la façon dont le **dedans** fonctionne



Responsabilité & intégrité

- Une classe doit avoir 1 et 1 seule responsabilité
- Une classe est elle seule responsable de l'intégrité de ses données (encapsulation)
- Un système complexe est donc modélisé par des interactions entre des objets simples

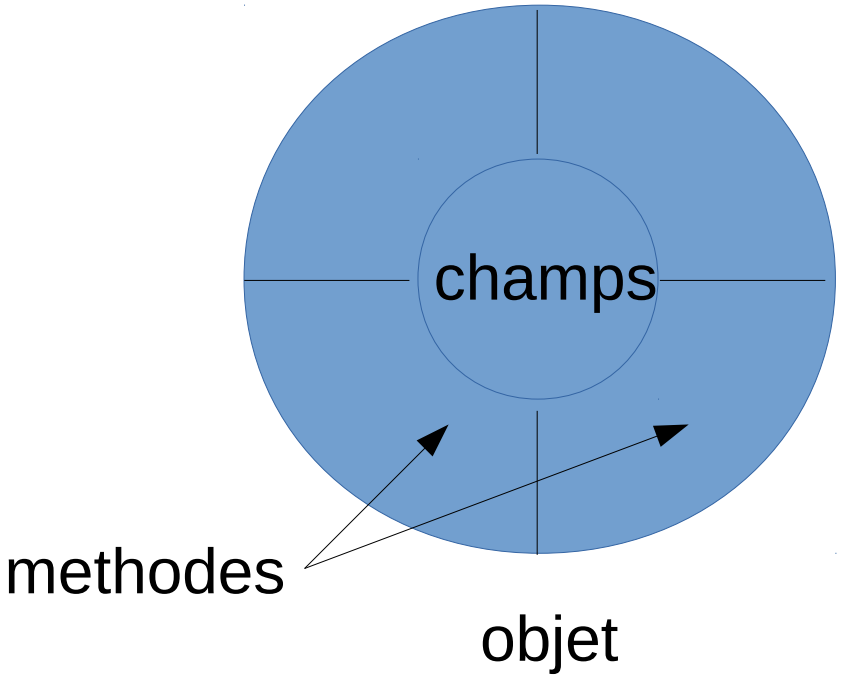
Instance de classe et objet

- Une classe définit un moule à partir duquel on fabrique des objets
  - On appelle ces objets des instances de la classe

Methodes & POO

Une méthode représente un point d'accès vis à vis de l'extérieur d'un objet

Le code d'une méthode a accès à l'intérieur d'un objet



Encapsulation

“ la seule façon de modifier l'état d'un objet est d'utiliser les méthodes de celui-ci ”

- Restreint l'accès/modification d'un champ à un nombre fini de code
  - Les méthodes de la classe

- Permet de contrôler qui fait les effets de bord
  - Le “mieux” est de ne pas faire d'effet de bord !

Intégrité & Contrat

“ La classe doit garantir les invariants ”

- Exemple d'invariant (une propriété)
  - par ex, le champ x est toujours positif
- Le constructeur sert de point d'entrée
- Pas de setter inutile !

Interface (compilation)

- Pour éviter la duplication de code, si un code manipule indifféremment des objets de plusieurs classes différentes alors les objets sont manipulés à travers une interface
  - Cf principe de sous-typage de Liskov
- L'interface contient l'intersection des méthodes communes utilisées par ce code

Polymorphisme (exécution)

- Un appel d'une méthode à travers une interface exécute la méthode de la classe (dynamique) de l'objet

- Il y a redéfinition à la compilation (*override*) lorsque la substitution par polymorphisme à l'exécution est possible

Cercle vertueux de l'ignorance

- On manipule les objets par des méthodes, on ignore donc l'implantation de celles-ci
- On manipule les objets à travers leurs interfaces, on ignore donc la classe de l'objets
- Cela permet de limiter l'écriture de code spaghetti et renforcer la localité

Un module

- Unité de code à forte cohésion
  - En Java, c'est souvent un package, packagé sous forme d'un jar
- Définit aussi un intérieur et extérieur
  - Visibilité de package (au lieu de privée)
  - Etend les concepts de responsabilité et de contrat
  - Etage au dessus de la classe qui permet de concevoir des systèmes complexes

Programme vs librairie

- Lorsque l'on écrit un programme, on utilise des libraries de façon externe
- Lorsque l'on écrit une librairie, on est en interne
  - On utilise les tests de cas d'utilisation pour vérifier la partie externe

Open / Close

- Une classe ou un module est soit
  - Open
    - En cours développement, où tout est possible
  - Close
    - Débuggé, Testé, Versioné

En théorie, on peut passer un module que de open vers close mais pas vice-versa

Et en pratique ?

Bonnes pratiques - checklist

- Responsabilité
  - 1 classe == 1 responsabilité
- Encapsulation
  - champs private !
- Intégrité
  - Un seul constructeur exigeant, tester les arguments !
  - Pas de getter et surtout setter inutiles !
- Evolutivité
  - Les méthodes publiques prennent et retournent des interfaces
- Interface explicite et adaptée au besoin
  - Noms corrects, verbe d'action pour les méthodes
  - Pas d'interface si pas d'utilisation commune de 2 classes
  - Pas de méthodes inutiles (- de 10 méthodes SVP)

Hierarchie de types

- Pas de classes inutiles
- Ne pas se précipiter pour construire des hiérarchies

- Héritage est souvent une fausse bonne idée
  - Couplage trop fort entre les classes
  - Privilégier la délégation qui est plus souple

  Héritage: lien entre classes à la compilation

  Délégation: lien entre objets à l'exécution

Open / Close

- Close them all !

- Pas mal de design patterns permettent d'utiliser/tourner autour du fait qu'une classe peut être fermée