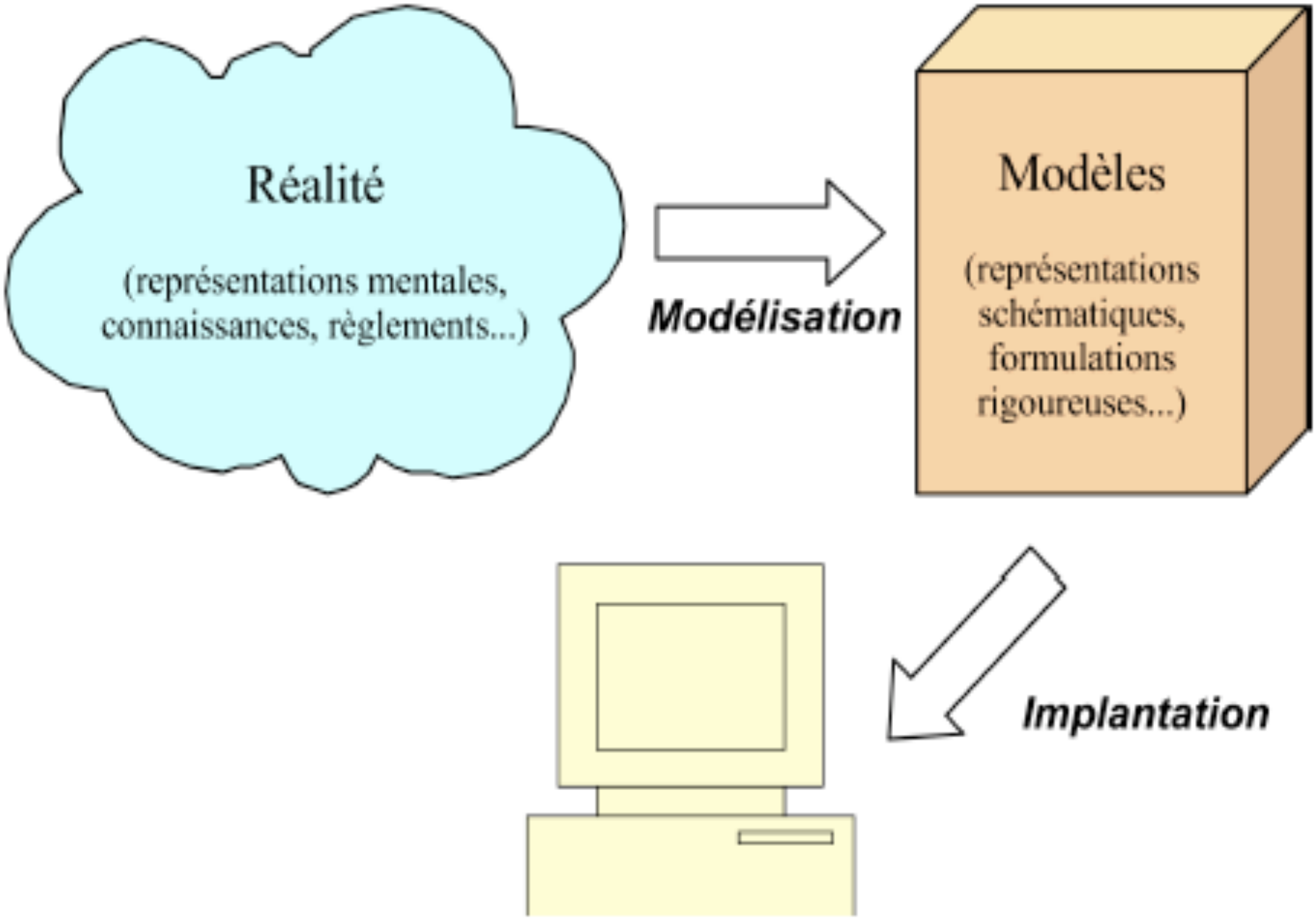
Conception orientée objet

Introduction


Réalité
(représentations mentales, connaissances, règlements...)
Modélisation
Modèles
(représentations schématiques, formulations rigoureuses...)
Implantation

Répartition des activités

Actuellement, dans un **projet logiciel** bien conduit :

40 % **Analyse, Spécification, Conception**

20 % **Programmation**

40 % **Intégration, Vérification, Validation**

C’est quoi la COO ??

- *La conception orientée objet (COO) est une méthode de développement logiciel :*
  - On a un problème pour lequel on veut une solution logicielle
  - La COO permet de déterminer :
    - quelles sont les classes Java (par exemple) écrire,
    - quelles sont les relations entre les classes,
    - quelles méthodes il faut mettre dans les classes,
    - quels sont les algorithmes généraux

C’est quoi la COO ??

- Sur la dénomination COO

– Il s’agit en fait d’un abus de langage courant : dans la terminologie informatique, la conception est une simple phase de la vie d’un logiciel au même titre que la programmation

- La COO ne se limite pas à produire du Java
  - Autres langages orientés objet comme C++ ou Ada95
  - Et même du C, du Fortran ou encore du Matlab en s’imposant des règles de programmation
- La COO relève largement de l’artisanat !
- – Multiples solutions correctes + multiples risques de se tromper
- – L’expérience joue un rôle important dans la qualité du logiciel résultant

Plan (partie 1)

- Problèmes du développement logiciel dans l'industrie
- Approche orientée objet
- Introduction à UML

Problèmes du développement logiciel

En 1995, une étude du Standish Group dressait un tableau accablant de la conduite des projets informatiques. Reposant sur un échantillon représentatif de 365 entreprises, totalisant 8 380 applications, cette étude établissait que :

- 16,2 % seulement des projets étaient conformes aux prévisions initiales,
- 52,7 % avaient subi des dépassements en coût et délai d'un facteur 2 à 3 avec diminution du nombre des fonctions offertes,
- 31,1 % ont été purement abandonnés durant leur développement.

Pour les grandes entreprises (qui lancent proportionnellement davantage de gros projets), le taux de succès est de

- 9% seulement,
- 37% des projets sont arrêtés en cours de réalisation,
- 50% aboutissent hors délai et hors budget.

L'examen des causes de succès et d'échec est instructif : la plupart des échecs proviennent non de l'informatique, mais de la maîtrise d'ouvrage (i.e. le client).

Suite

Un autre symptôme de cette crise se situe dans la non qualité des systèmes produits. Les risques humains et économiques sont importants

- arrêt de Transpac pour 7.000 entreprises et 1.000.000 d'abonnés : surcharge du réseau,
- TAURUS, un projet d'informatisation de la bourse londonienne : définitivement abandonné après 4 années de travail et 100 millions de £ de pertes,
- mission VENUS : passage à 500.000 km au lieu de 5.000 km à cause du remplacement d'une virgule par un point,
- non différenciation entre avion civil et avion militaire : guerre du Golfe – Airbus iranien abattu : 280 morts,
- échec d'Ariane 501 (Erreur Fortran).

Une nouvelle discipline : le Génie Logiciel

- Afin de parer à ces difficultés une nouvelle discipline a été créée de toutes pièces dès 1968 : le Génie Logiciel
- But : produire du logiciel de qualité
- Moyens :
  - Définition d'un langage commun pour traiter les problèmes du logiciel
    - Etapes de la vie du logiciel, ou cycle de vie
    - Critères permettant de mesurer sa qualité
  - Définition de méthodes de développement du logiciel
    - Doivent rester cohérentes sur tout le cycle de vie
    - Doivent prendre en compte le système dans son intégralité
    - Si possible, unifier les méthodes sur l'ensemble du cycle de vie

Cycle de vie du logiciel

- On retrouve en général les étapes suivantes dans la vie du logiciel :

– Analyse

• Ce que doit faire le système, son interface avec les utilisateurs ou les autres systèmes en fonction du cahier des charges

– Conception

• Architecture du système en terme de modules, communication entre les modules

• Conception détaillée de chaque module : données et algorithmes

– Programmation

• Codage des modules, éventuellement par différentes équipes

• Tests unitaires de chaque module

– Intégration

• Test de l'intégration des modules pour constituer l'application

– Validation

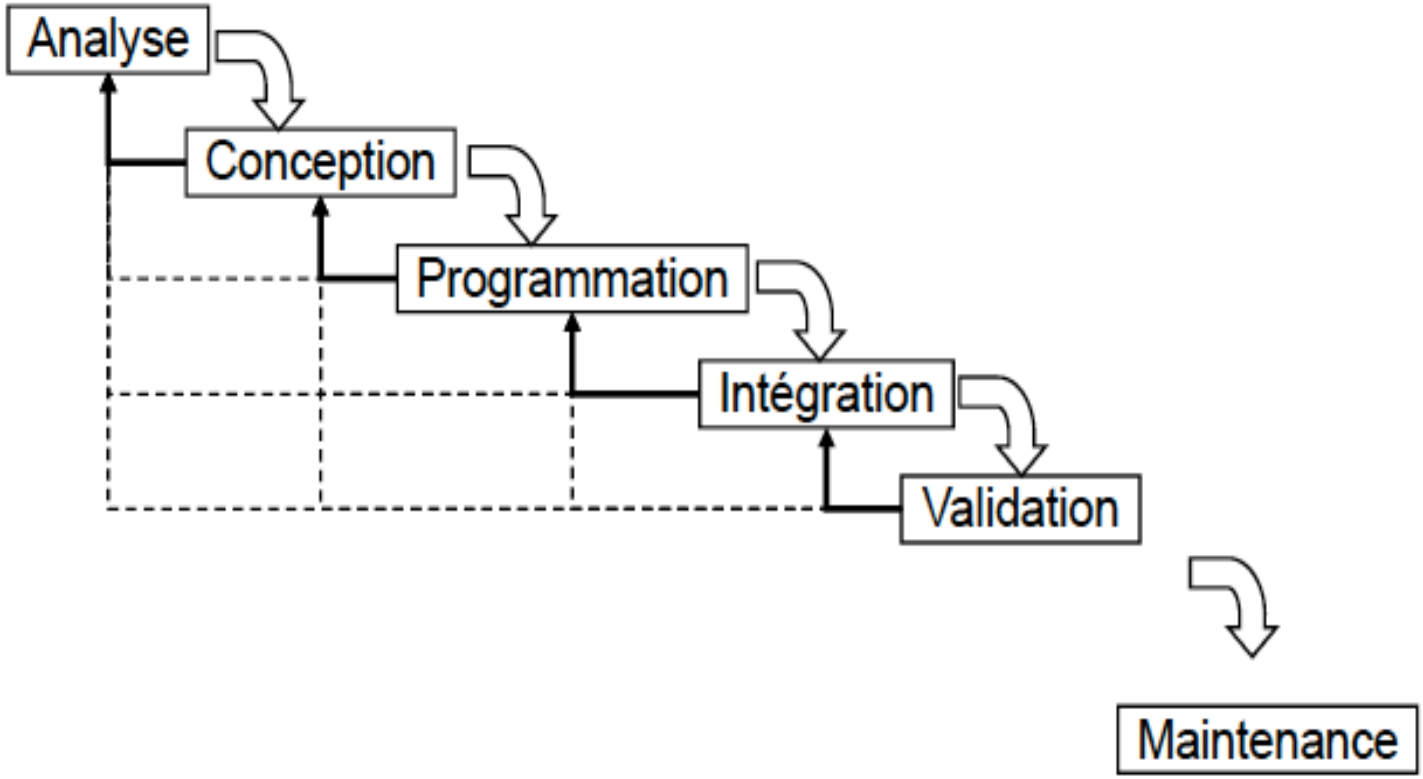
• Vérification de la conformité la spécification (fonctionnalités, performances)

– Maintenance

• Suivi du produit en service (réparations, ajout de fonctions)

Cycle de vie du logiciel

- Le cycle en cascade classique



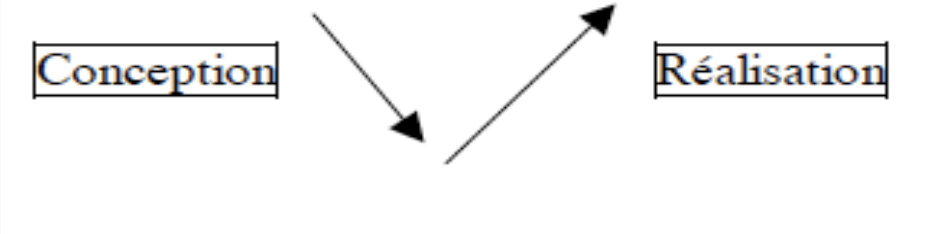
- Existe beaucoup variations sur ce modèle
  - Itérations, incréments

Inconvénients

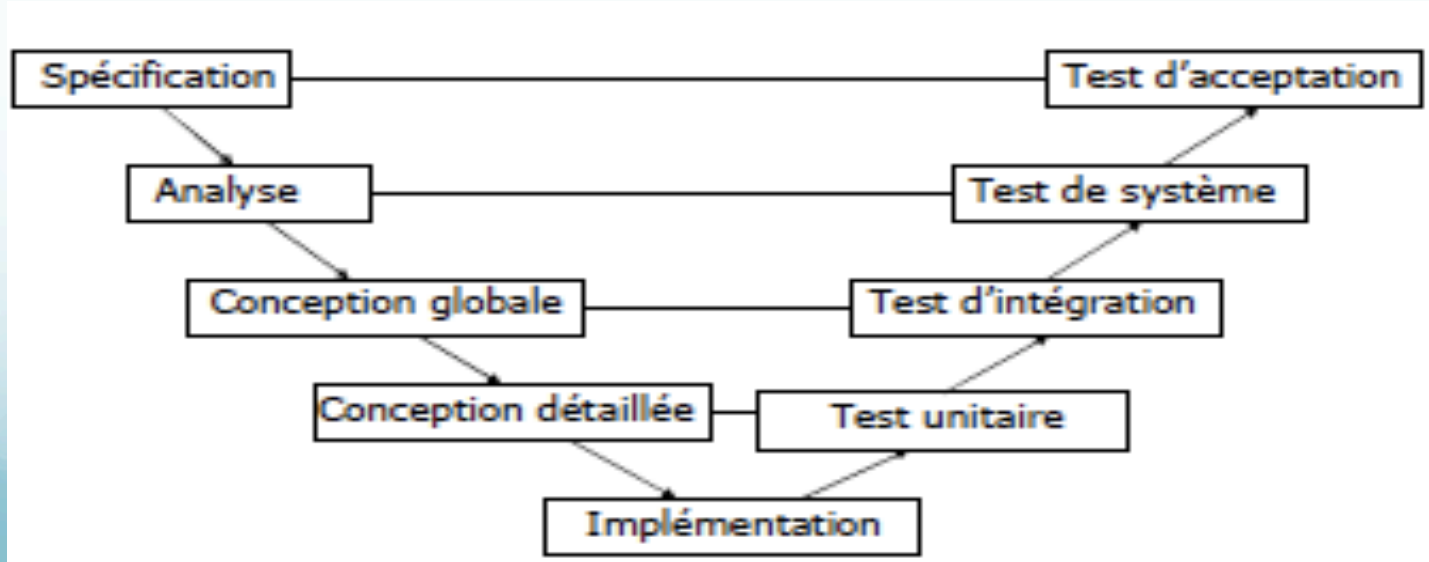
- Le processus est trop rigide: une phase ne peut commencer que lorsque la phase précédente est terminée
- Aucune validation entre les phases
- Difficile de détecter des erreurs
- Détecter des problèmes trop tard: erreurs d'analyse ou erreurs de conception sont très coûteuses

Le processus en V

Dans cette méthode, la conception et la réalisation forment les deux branches du cycle en V :



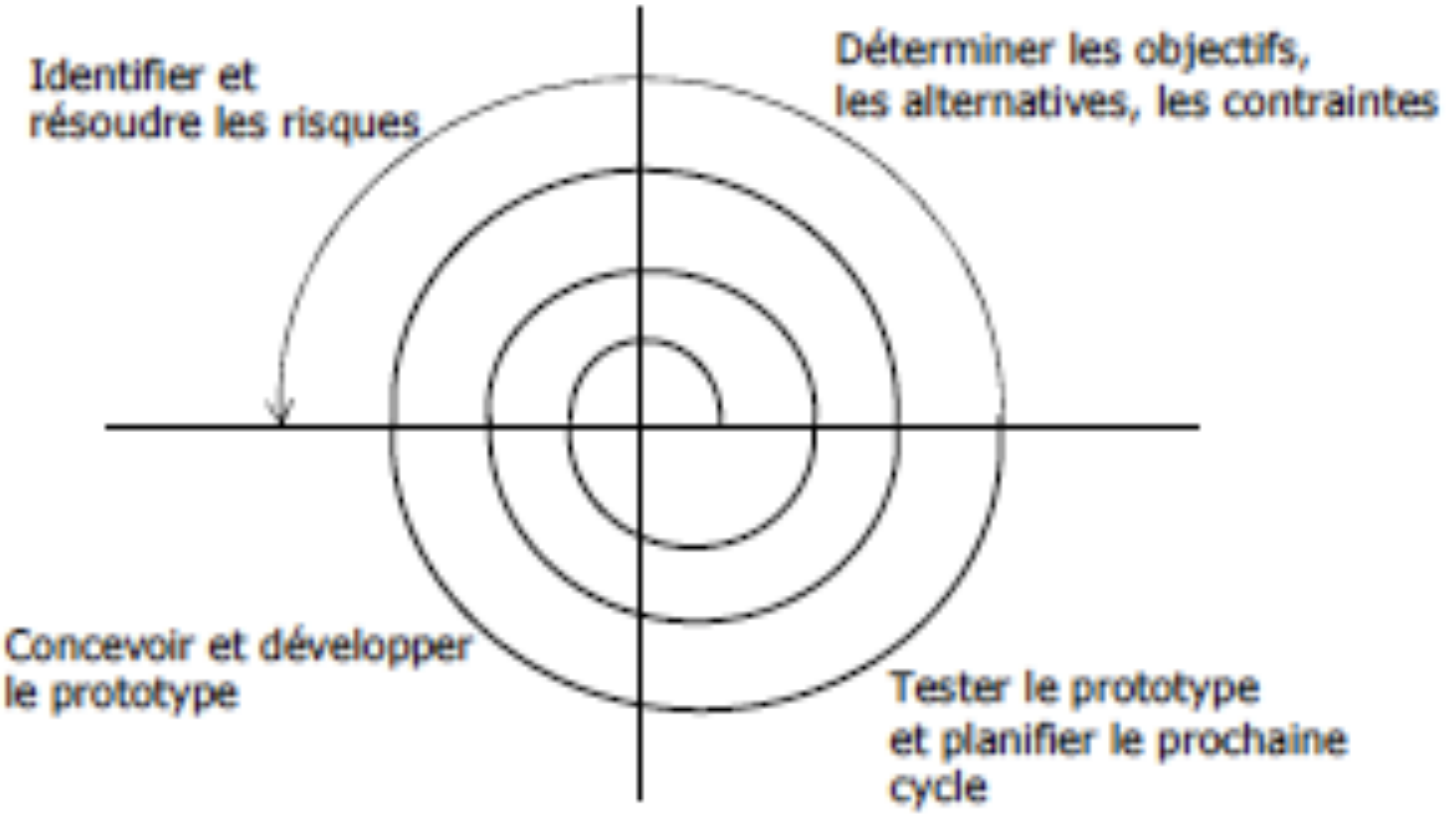
Ces deux étapes sont détaillées



Le processus spiral

- Idée : fournir le plus rapidement possible un prototype exécutable permettant la validation concrète et non sur le document
- Progression du projet par incréments successifs des versions du prototype ⇒itérations
- Certains prototypes sont montrés aux clients et aux utilisateurs pour recevoir le plus tôt possible leurs réactions

Le processus spiral (Suite 1)



Le processus spiral (Suite 2)

Avantages

- Réduire les risques
- Recevoir très tôt les réactions des utilisateurs
- Réduire la complexité
- Faciliter l'adaptation au changement de la technologie
- Validation concrète et non sur documents

Méthodes de développement de logiciel

- Besoin : pourquoi a-t-on besoin d'une méthode ?
  - Pour arriver à faire ce logiciel
    - dans de bonnes conditions,
    - avec une certaine confiance dans le résultat final
  - Pour expliquer sa structure, son fonctionnement (aux clients, aux nouveaux développeurs)

- Contenu de la méthode
  - comment modéliser et construire des logiciels
    - représentation, le plus souvent graphique
    - règles de mise en œuvre : le processus

La méthode que nous allons voir

- Méthode basée sur
  - l'approche objet
  - la notation graphique UML (Unified Modeling Language)

Approche fonctionnelle

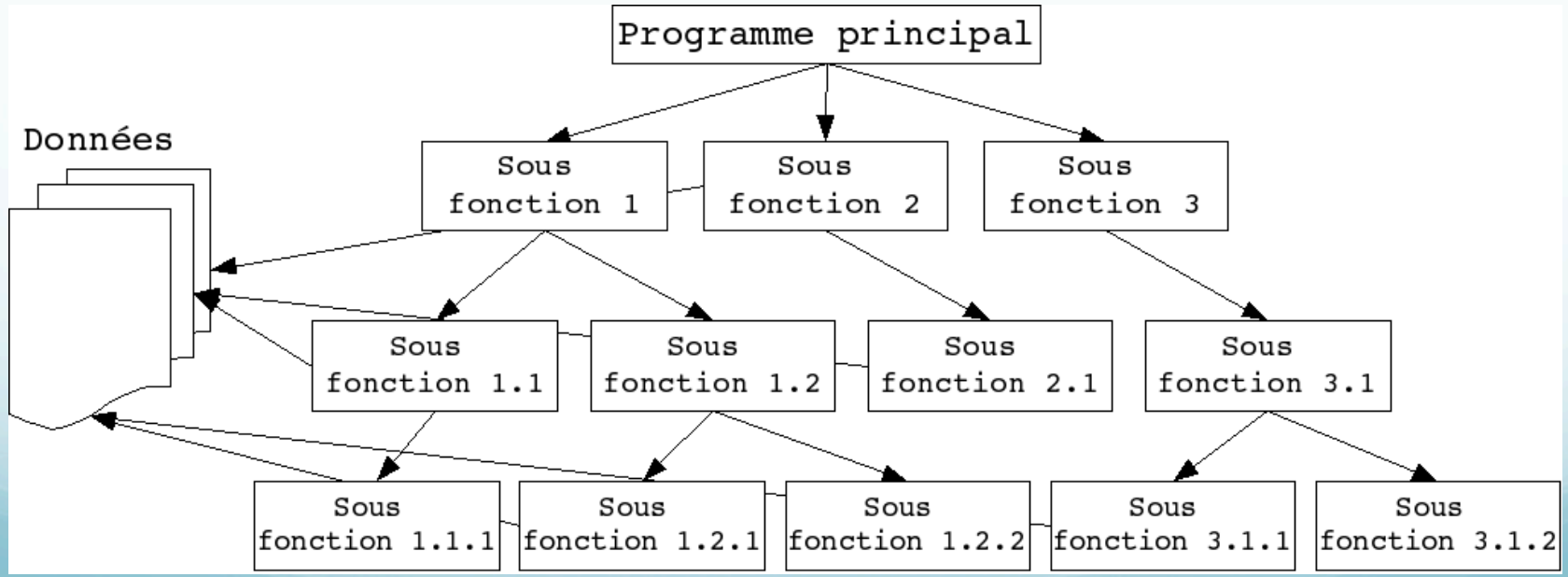
&

Approche orientée objet

Approche fonctionnelle/procédurale

En se Basant sur les fonctions spécifiées du système, il se compose de plusieurs fonctions:

- Décomposition des fonctions en sous fonctions
- Un système se compose des sous-systèmes
- Un sous-système se décompose en plus petits sous-systèmes, ...



Les fonctions se communiquent en utilisant des données partagées ou transferts de paramètres.

Avantages

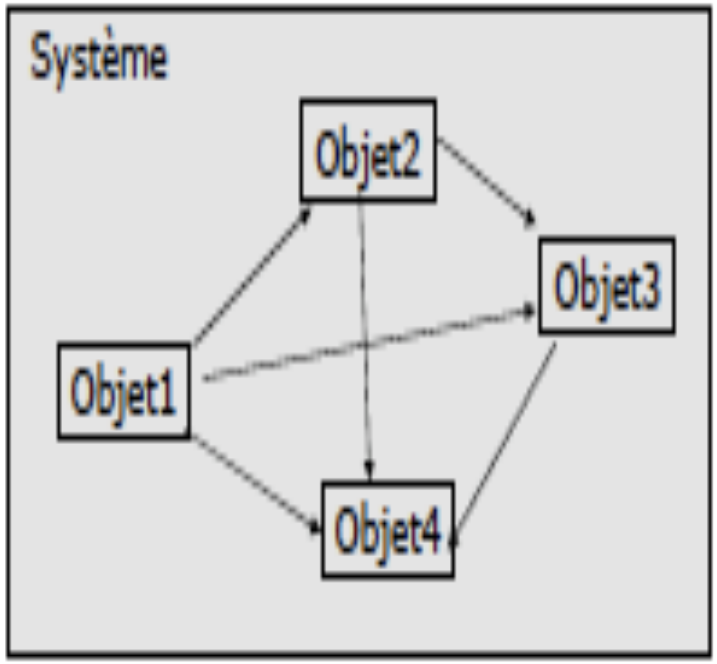
- Facile à appliquer
- Fonctionne bien lorsque les données sont simples
- Aide à réduire la complexité
- Obtention des résultats attendus

Inconvénients

- Les fonctions sont séparées des données
- La structure du système est définie en fonction des fonctions, alors un changement des fonctions causera des difficultés du changement de la structure
- Difficile de réutiliser
- Un coût important pour la maintenance

Approche orientée objets

- La solution d'un problème est organisé autour du concept d'objets
- L'objet est une abstraction des données contenant aussi des fonctions
- Un système se compose des objets et des relations entre eux
- Les objets se communiquent en échangeant de messages pour réaliser une tâche
- Pas de variables globales
- Encapsulation
- Héritage



Approche orientée objets

Avantages

- Très proche du monde réel
- Facile à réutiliser
- Cacher l'information (encapsulation)
- Coût de développement moins élevé(héritage)
- Appropriée aux systèmes complexe
- etc

Dans l'approche orientée objets

- Objets
- Classes
- Encapsulation
- L'héritage
- Polymorphisme
- Abstraction

Les objets

- L'objet est le concept décrivant les entités dans le monde réel
- Il existe des relations entre les objets

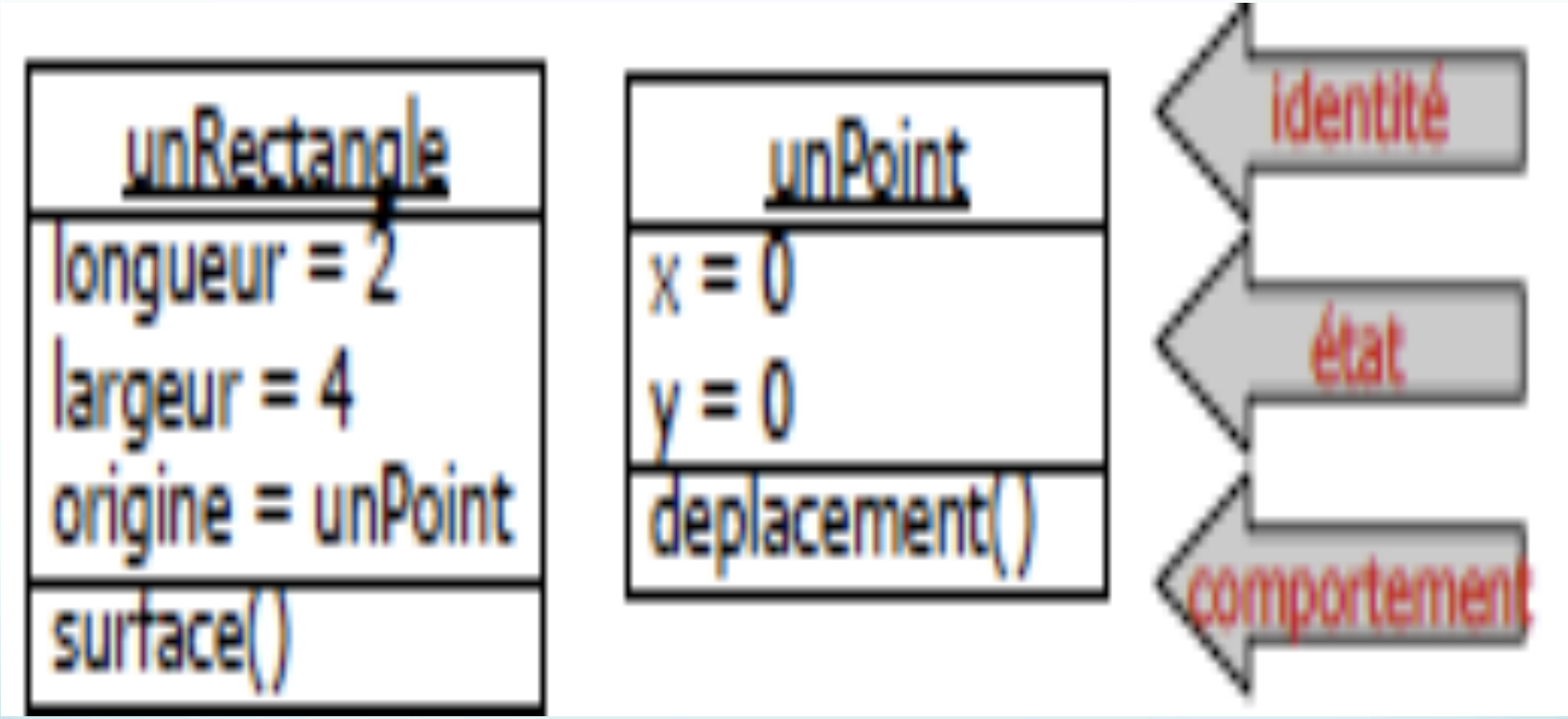
✓ L'étudiant Ali est un objet

✓ L'étudiant ne peut être pas un objet !

Objet = état + comportement + identité

- L'état (données) décrit les caractéristiques d'un objet à un instant donné, et est enregistré dans les variables
- Le comportement est exprimé par les fonctions de l'objet
- Chaque objet possède une identité unique

Exemple:


unRectangle
longueur = 2
largeur = 4
origine = unPoint
surface()
unPoint
x = 0
y = 0
deplacement()
identité
état
comportement

- État = ensemble d'attributs

Un attribut décrit une propriété de l'objet.

- à chaque instant, un attribut a une valeur dans un domaine déterminé

Exemple: La voiture a des propriétés: poids, couleur, longueur, largeur, nombre de kilomètres, ...

-Une Renault 207 pèse 1300 kilos, elle est rouge, ...

- Comportement = ensemble de fonctions

Une fonction/méthode est la capacité de l'objet de réaliser une tâche

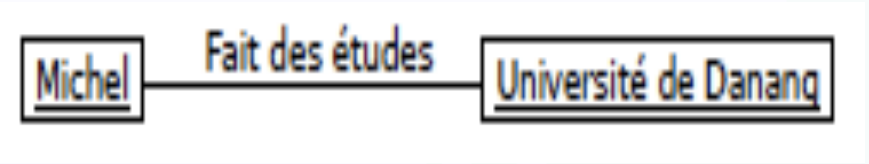
- le comportement dépend de l'état

Exemple: Une voiture peut démarrer, rouler, ...

Liens

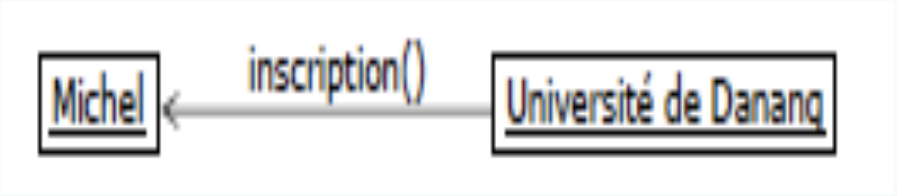
Entre les objets, il peut exister les liens

Exemple:



Communication entre objets

Envoie des messages



Types de messages:

- Constructeur
- Destructeur
- Accesseur
- Modificateur
- Autres fonctions

Classes

- Une classe est une description abstraite d'un ensemble d'objets ayant
  - Des propriétés similaires
  - Des comportement communs
  - Des relations communes avec d'autres objets
- Classe = une abstraction

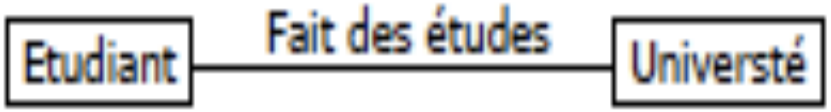
Abstraction: chercher des aspects communs et omettre des aspects différents des objets

Réduire la complexité

Classes (suite 1)

Relations

- Entre classes, il peut exister des relations
- Une relation entre des classes est un ensemble des liens entre des objets de ces classes

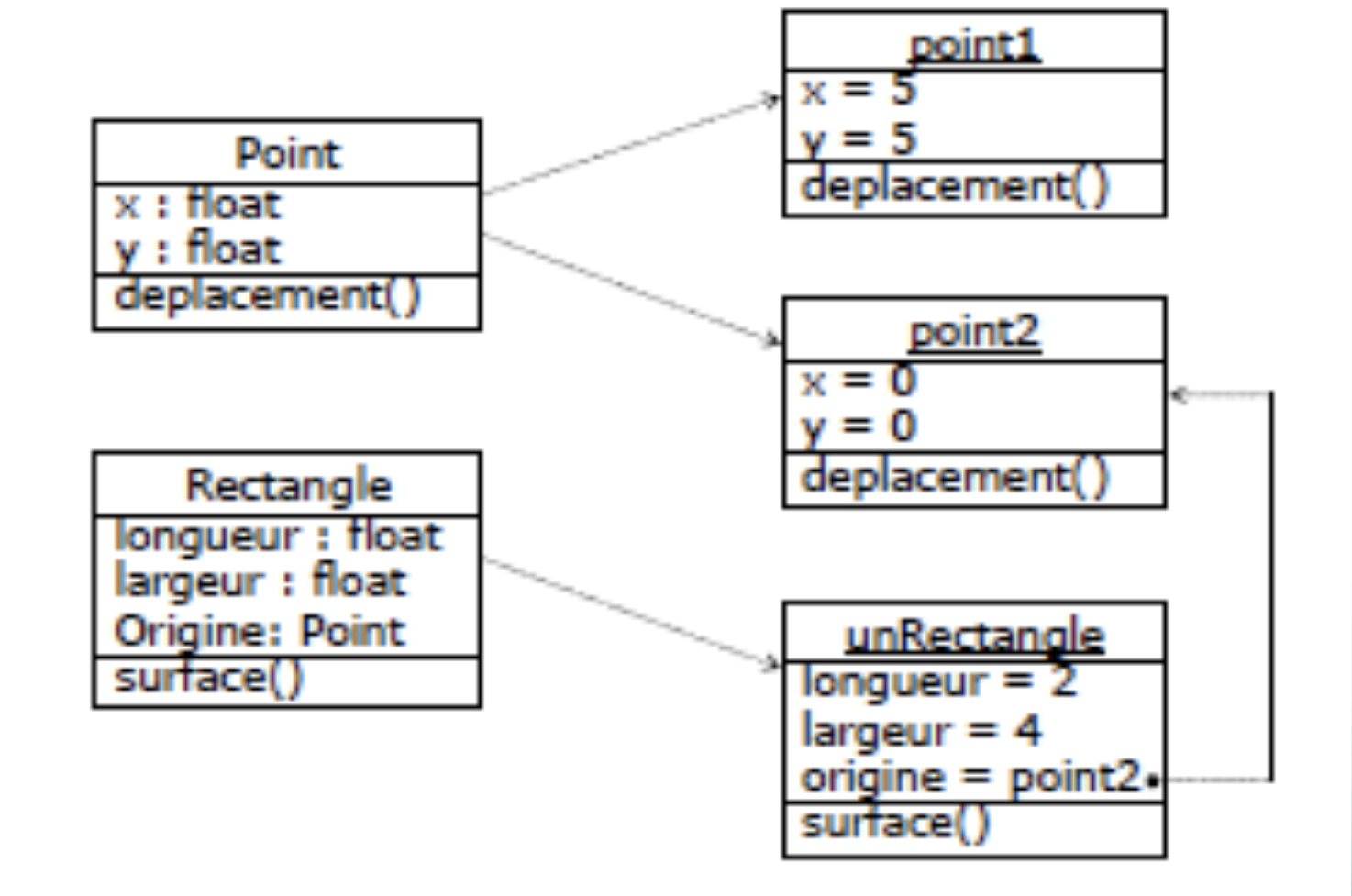


Classe / Objet

- Un objet est une instance de la classe
- Une valeur est une instance d'un attribut
- Un lien entre objets est une instance de la relation entre classes

Classes

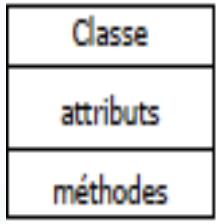
Exemple: Classe/Objet



Encapsulation

Données + Traitement des données = Objet

Attributs + Méthodes = Classe



- L'état de l'objet est encapsulé par un ensemble d'attributs (privés)
- Le comportement est réalisé par un ensemble de méthodes
  - Les utilisateurs de l'objet connaissent les messages que l'objet peut recevoir (méthodes publics)
  - Les implémentation des méthodes restent cachées aux utilisateurs externes

Avantages

- Cacher de l'information
- Restriction de l'accès à l'information depuis l'extérieur
- Eviter des changements globaux (i.e. dans tout le système): l'implémentation interne peut être modifiée sans affecter les utilisateurs externes
- Facilite la modularité
- Facile à réutiliser
- Facile à maintenir

L'héritage

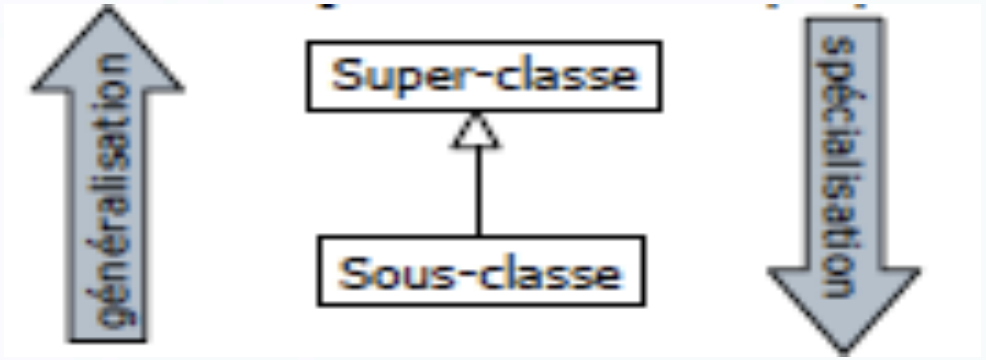
L'héritage permet la réutilisation de l'état et du comportement d'une classe par d'autres classes

- Une classe est dérivée d'une ou plusieurs classes en partageant les attributs et les méthodes
- La sous-classe hérite des attributs et méthodes de la super-classe (classe père)

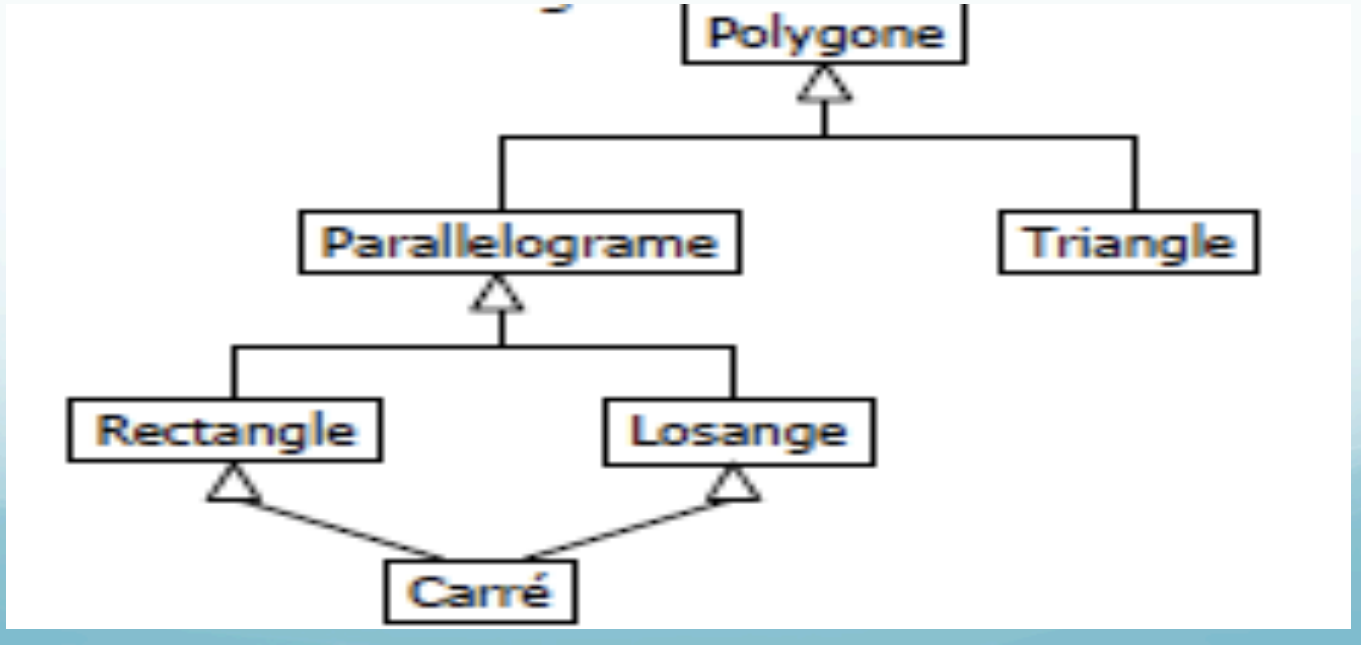
Généralisation/spécialisation

- Généralisation: les propriétés communes des sous-classes servent à construire la super-classe
- Spécialisation: les sous-classes sont construites à partir de la super-classe en ajoutant d'autres propriétés propres à elles

L'héritage (suite 1)



- Héritage simple: une sous-classe hérite d'une seule super-classe
- Héritage multiple: une sous-classe hérite de plusieurs super-classes
- Exemple : un arbre d'héritage



L’héritage (suite 2)

Avantages

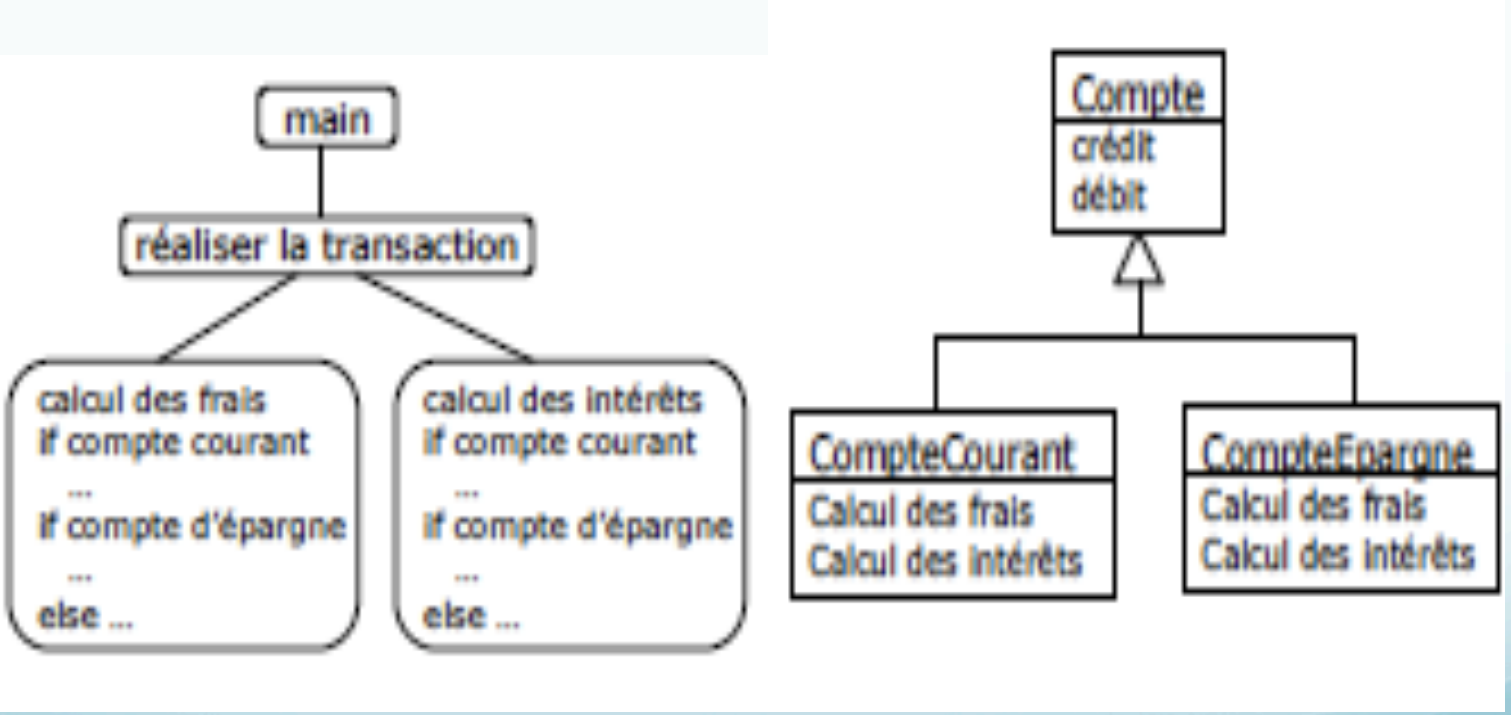
- Organisation des classes
- Les classes sont organisées hiérarchiquement
- Facile à gérer des classes
- Construction des classes
- les sous-classes sont construites à partir des super classes
- Réduction du coût de développement en évitant de récrire du code
- Permettre d’appliquer facilement la technique *polymorphisme*

Polymorphisme

- Polymorphisme des méthodes
- Les différentes méthodes sont capables de répondre à une requête
- Les méthodes portant le même nom sont définies différemment (différent comportements) dans différentes classes
- Les sous-classes héritent de la spécification des méthodes de la super-classe et ces méthodes peuvent être redéfinies pour réaliser leurs propres buts

Polymorphisme

- Réduire l'insertion des instructions conditionnelles telles que if-else ou switch
- Approche procédurale versus approche orientée objets

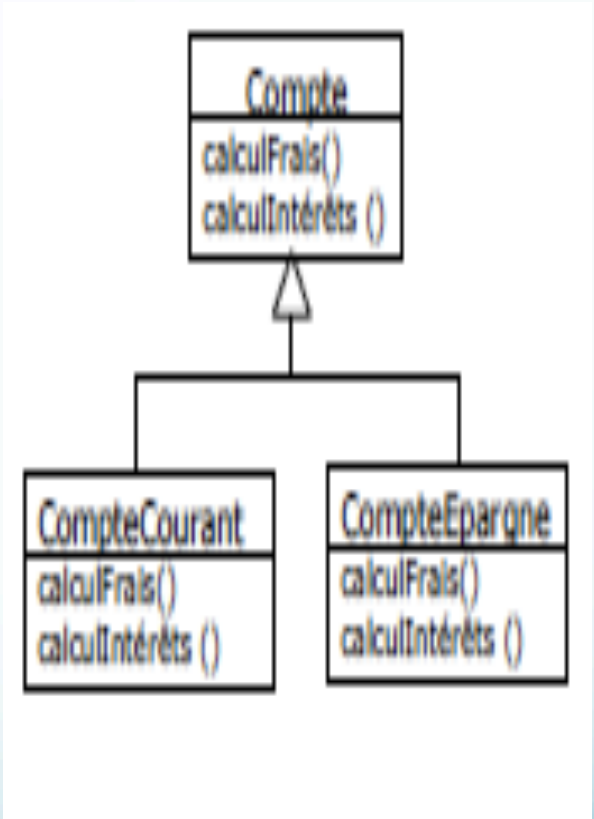


Polymorphisme: liaison dynamique

- Le polymorphisme, conséquence directe de l'héritage, permet à même message, dont l'existence est prévue dans une superclasse, de s'exécuter différemment, selon que l'objet qui le reçoit est une sous-classe ou d'une autre.

Polymorphisme (suite)

```
 void calcul(Compte cpt[], int nb)
    {
    for (int i = 0; i < nb; i++)
    cpt[i]=calculFrais(); }

void main()
    {
    Compte cpt[4];

 cpt[0] = new CompteCourant();

 cpt[1] = new CompteEpargne();

   calcul(cpt,4);

}
```



Abstraction:classe abstraite

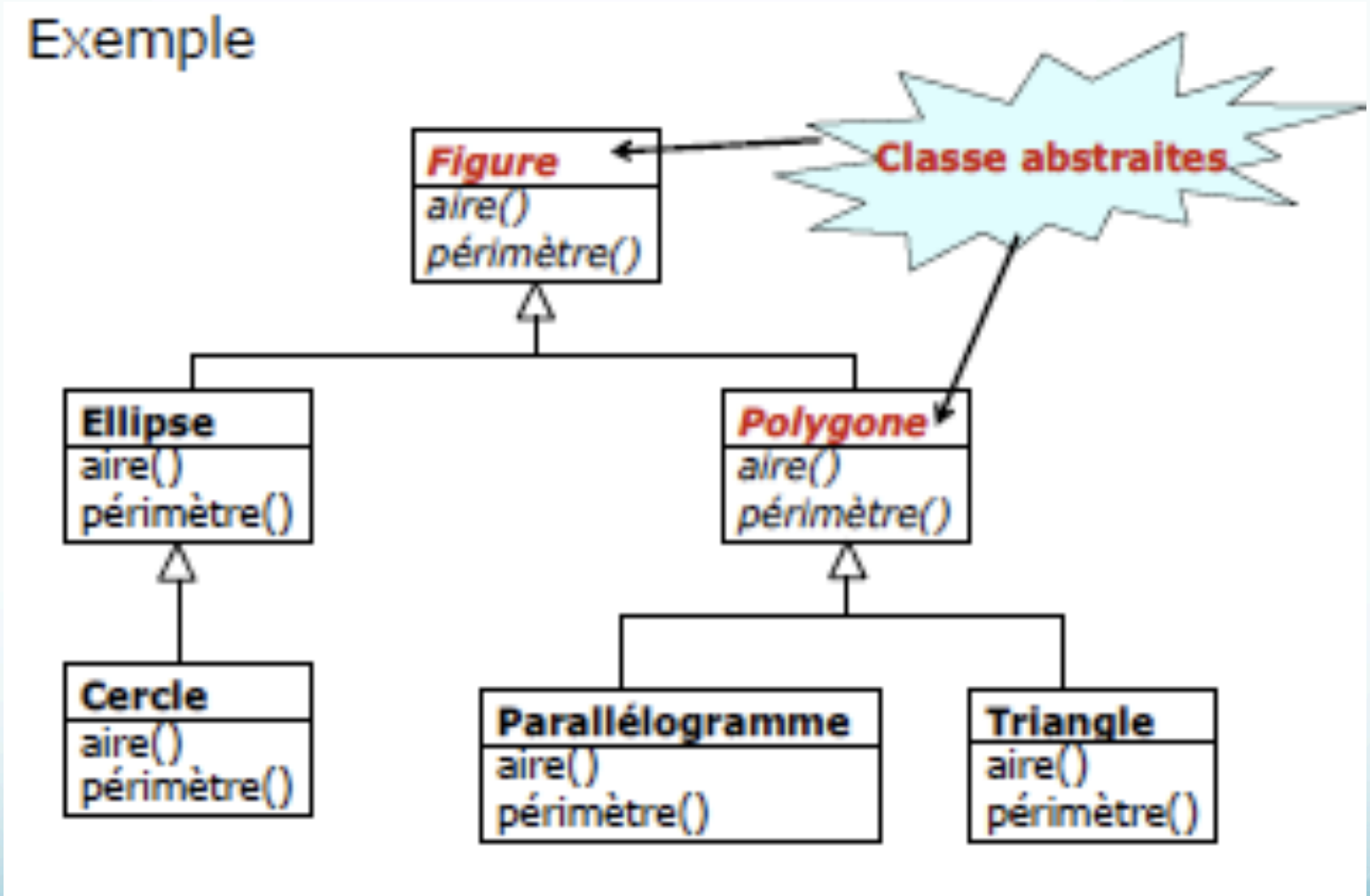
Une classe abstraite

- qui indique les caractéristiques communes de toutes ses sous-classes
- qui ne peut pas avoir des instances/objets

Une classe concrète

- qui contient une caractérisation complète des objets réels
- qui est prévue pour avoir des instances/objets

Abstraction:classe abstraite (suite)



Abstraction : méthode abstraite

Une méthode doit être définie au plus haut niveau d'abstraction possible

- A ce niveau, la méthode peut être abstraite il n'y a pas d'implémentation de cette méthode à ce niveau
- Dans ce cas, la classe devient aussi abstraite
- Si une classe possède une méthode abstraite, au moins une de ses sous-classes doit implémenter concrètement cette méthode

Toutes les méthodes d'une classe au bas de l'arbre d'héritage doivent être concrètes

Abstraction : méthode abstraite

Exemple