

Rappel des concepts fondamentaux du paradigme orienté objet

Exemples cours

Inspirés de

- Object-Oriented Software Engineering: Practical Software Development using UML and Java - Timothy C. Lethbridge, Robert Laganière
- Effective Java Programming Language Guide - Joshua Bloch

Qu'est-ce que
l'Orientation Objet?

Paradigme Procédural

Paradigme Procédural

Le logiciel est organisé autour de la notion de *procédures*.

Paradigme Procédural

Le logiciel est organisé autour de la notion de *procédures*.

- **Abstraction de procédures**
 - ▶ fonctionne bien lorsque les données sont simples

Paradigme Procédural

Le logiciel est organisé autour de la notion de *procédures*.

- **Abstraction de procédures**
 - ▶ fonctionne bien lorsque les données sont simples
- **Abstraction de données**
 - ▶ grouper ensemble les données décrivant une même entité
 - ▶ aide à réduire la complexité du système

Paradigme Orienté Objet

Paradigme Orienté Objet

**Organiser la solution d'un problème autour
du concept d'objets.**

Paradigme Orienté Objet

Organiser la solution d'un problème autour du concept d'objets.

- Les **objets** sont des instances de **classes**:
 - ▶ ce sont des abstractions de données
 - ▶ contenant des abstractions de procédures

Paradigme Orienté Objet

Organiser la solution d'un problème autour du concept d'objets.

- Les **objets** sont des instances de **classes**:
 - ▶ ce sont des abstractions de données
 - ▶ contenant des abstractions de procédures
- Programme = un ensemble d'objets collaborant entre eux afin d'effectuer un tâche donnée

Concepts clé POO

Concepts clé POO

- **Objets** (ou Identités)
 - ▶ chaque objet a une existence distincte

Concepts clé POO

- **Objets** (ou Identités)
 - ▶ chaque objet a une existence distincte
- **Classes**
 - ▶ décrire des ensembles d'objets

Concepts clé POO

- **Objets** (ou Identités)
 - ▶ chaque objet a une existence distincte
- **Classes**
 - ▶ décrire des ensembles d'objets
- **Héritage**
 - ▶ mécanisme permettant d'hériter des propriétés et comportements dans une hiérarchie de classes

Concepts clé POO

- **Objets** (ou Identités)
 - ▶ chaque objet a une existence distincte
- **Classes**
 - ▶ décrire des ensembles d'objets
- **Héritage**
 - ▶ mécanisme permettant d'hériter des propriétés et comportements dans une hiérarchie de classes
- **Polymorphisme**
 - ▶ mécanisme suivant lequel il peut exister plusieurs méthodes pour la même opération

Objet

- un ensemble structuré de données s'exécutant dans un logiciel
- a des **propriétés**
 - ▶ représentant son état
- a un **comportement**
 - ▶ définissant comment il (ré)agit
 - ▶ simulant le comportement d'un objet du monde réel

Classe

- Représente des **objets similaires** (ses instances)
- **Unité d'abstraction** dans un programme OO
- Est un module logiciel
 - ▶ décrivant la **structure** de ses instances (**propriétés**)
 - ▶ contenant des **méthodes** définissant leur **comportement**

Nommer une classe

- Toujours débiter par une ***lettre majuscule***
 - ➔ Banque non banque
- Utiliser le ***singulier***
 - ➔ Banque non Banques
- Utiliser le ***bon niveau de généralité***
 - ➔ Chat préférable à Animal
- Nom ***non ambigu*** ayant un sens précis
 - ➔ Pièce peut avoir plusieurs sens

Variables d'instances

- Des variables définies à l'intérieur d'une classe
- **Attributs**
 - ▶ des données simples
 - ▶ exemple : `nom`, `dateDeNaissance`
- **Associations**
 - ▶ relations avec d'autres classes
 - ▶ exemple : `superviseur`, `coursSuivis`

Variables vs. Objets

- Une **variable**
 - ▶ référence vers un objet
 - ▶ peut référencer différents objets à différents instants
- Un **objet** peut être simultanément référencé par plus d'une variable
- Type d'une variable
 - ▶ détermine quelle classe d'objets elle peut référencer

Variables de classe

- La valeur d'une **variable de classe** (ou **variable statique**) est partagée par toutes les instances de la classe
- Utiles pour:
 - ▶ représenter des constantes (e.g. PI)
 - ▶ représenter des propriétés s'appliquant à une classe en général
- **Attention**
 - ▶ ne pas en faire un usage excessif

Opérations

- Une abstraction procédurale de haut niveau correspondant à un comportement spécifique
- Indépendante de toute implémentation
 - ▶ exemple : calcul de l'aire d'une figure

Méthodes

- Une abstraction procédurale utilisée pour implémenter un comportement dans une classe donnée
- Plusieurs classes différentes peuvent avoir des **méthodes** de même nom
 - ▶ elles réalisent la même *opération* d'une manière propre à chaque classe
 - ▶ exemple : le calcul de l'aire d'un rectangle et d'un cercle

Organisation des classes en hiérarchies

Super-classe

- contient les éléments communs à un ensemble de sous-classes

Arbre d'héritage

- montre la relation entre super- et sous-classes

Héritage

- le mécanisme selon lequel toutes les sous-classes possèdent implicitement les éléments de leurs super-classes

Généralisation

Généralisation

- Vérifie la règle “***est-un(e)***”

Généralisation

- Vérifie la règle “**est-un(e)**”
 - ▶ “Un chat **est un** animal”

Généralisation

- Vérifie la règle “**est-un(e)**”
 - ▶ “Un chat **est un** animal”
 - ▶ “Un compte chèque **est un** compte de banque”

Généralisation

- Vérifie la règle “**est-un(e)**”
 - ▶ “Un chat **est un** animal”
 - ▶ “Un compte chèque **est un** compte de banque”
 - ▶ “Un village **est une** municipalité”

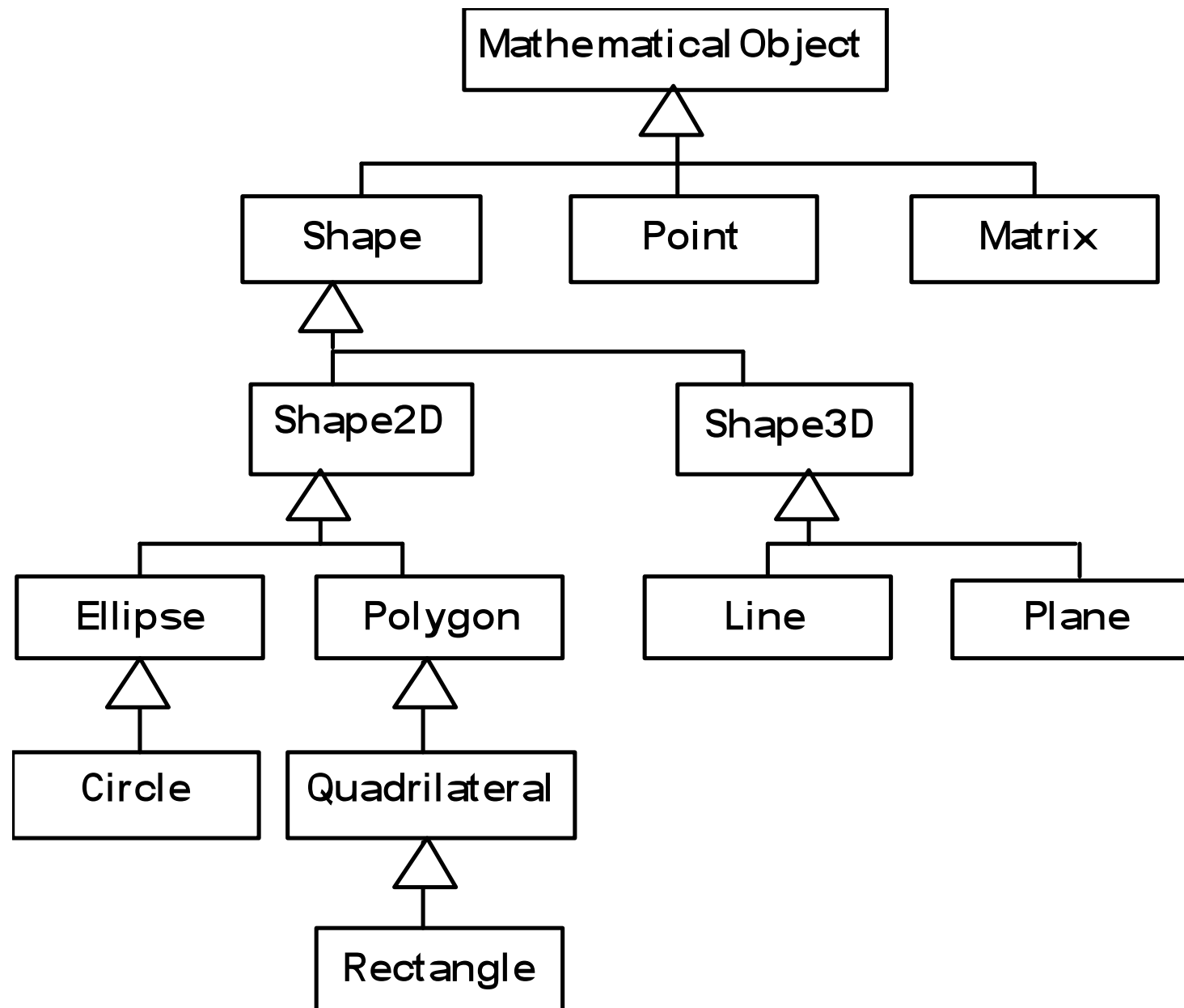
Généralisation

- Vérifie la règle “**est-un(e)**”
 - ▶ “Un chat **est un** animal”
 - ▶ “Un compte chèque **est un** compte de banque”
 - ▶ “Un village **est une** municipalité”
- Est-ce qu’une Province devrait être sous-classe de Pays?

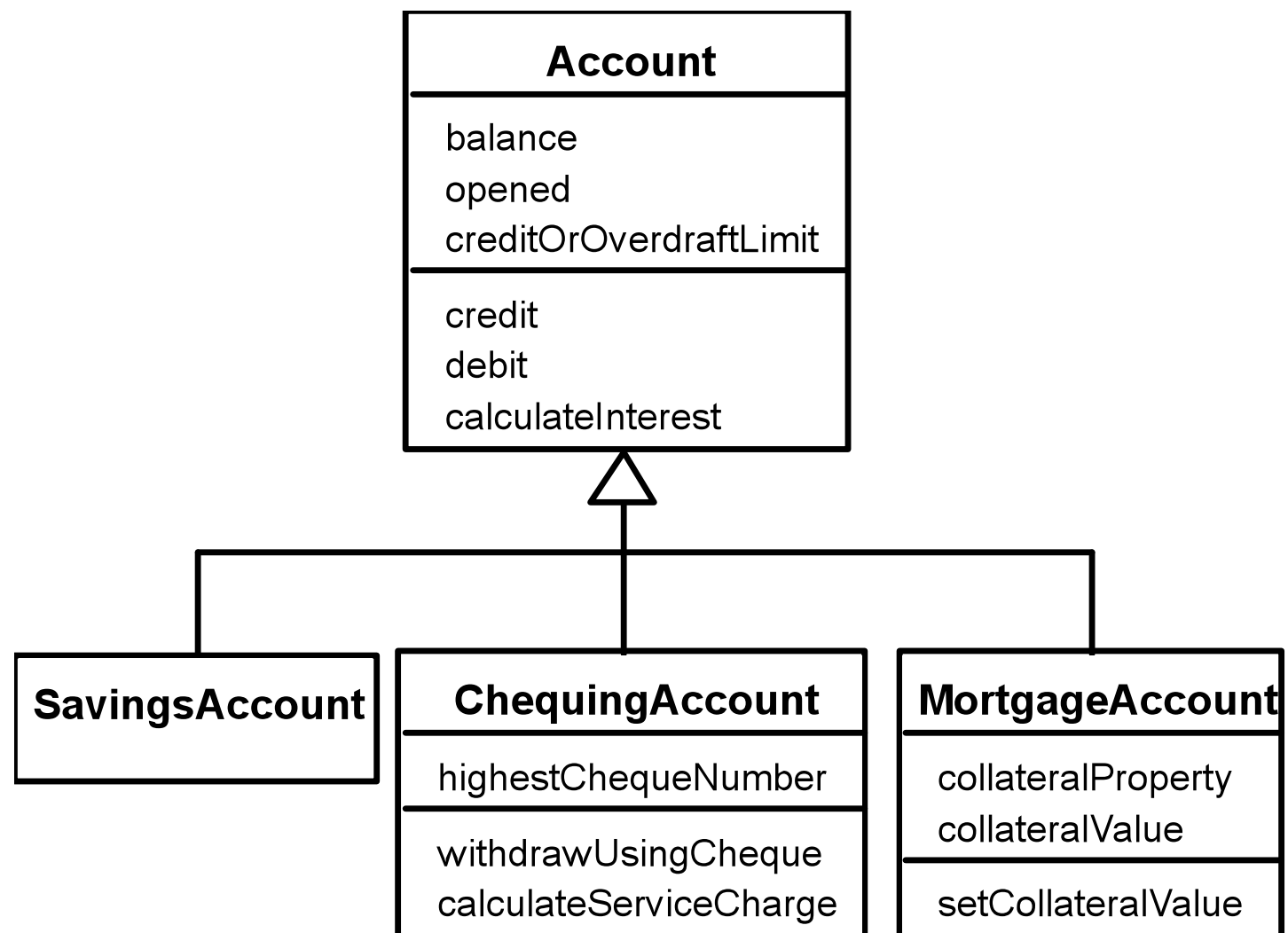
Généralisation

- Vérifie la règle “**est-un(e)**”
 - ▶ “Un chat **est un** animal”
 - ▶ “Un compte chèque **est un** compte de banque”
 - ▶ “Un village **est une** municipalité”
- Est-ce qu’une Province devrait être sous-classe de Pays?
 - ▶ “Une province **n’est pas un** pays”

Entités géométriques



Héritage: comptes bancaires

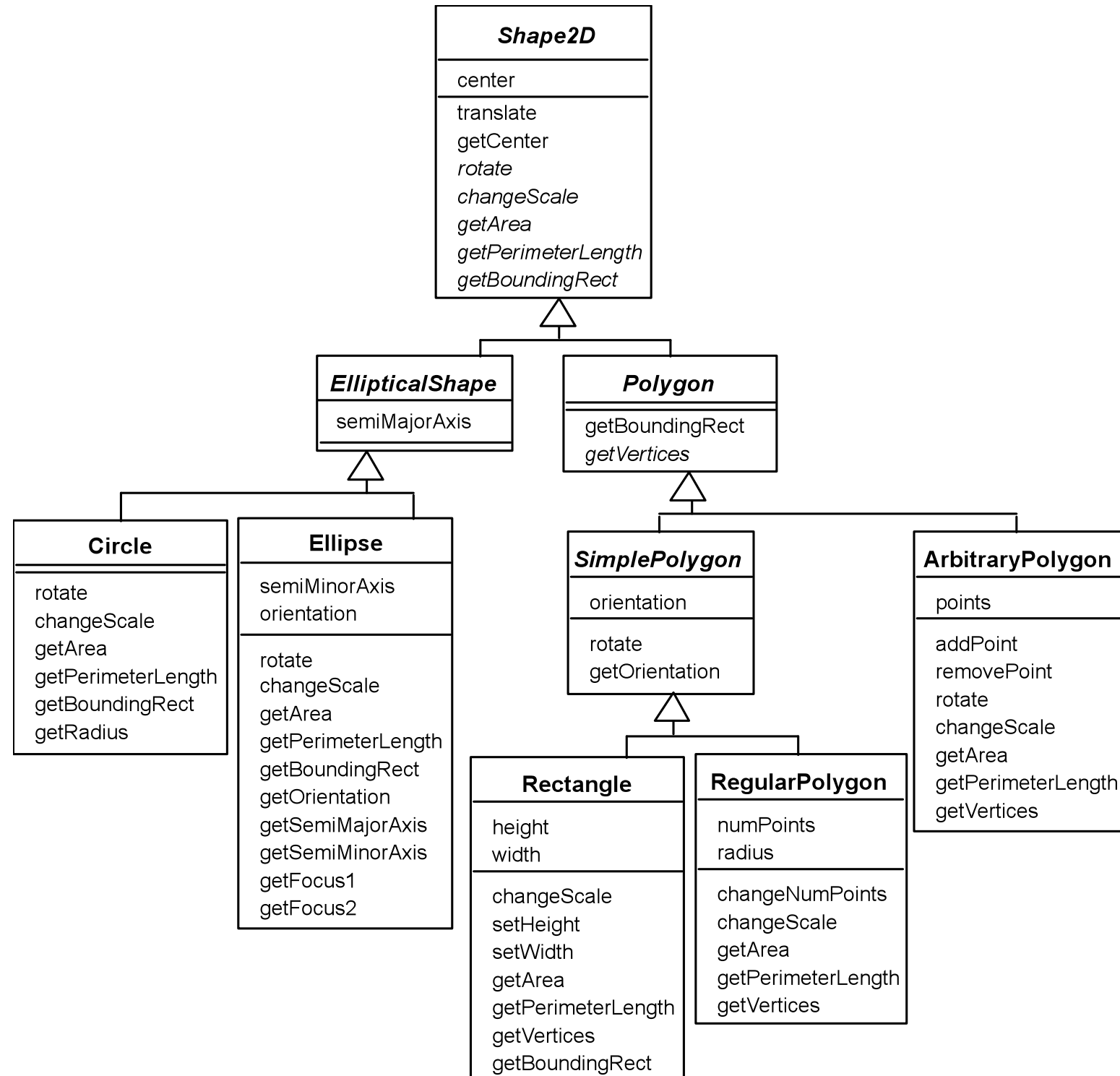


Polymorphisme

Une même opération s'applique de différentes façons dans différentes classes

- ▶ Exige l'existence de plusieurs méthodes ayant le même nom
- ▶ La méthode qui sera exécutée par un objet dépend de la classe d'appartenance de cet objet
- ▶ Réduit grandement la nécessité d'insérer des énoncés de contrôle tels que le `if-else` ou le `switch`

Polymorphisme



Classes abstraites et Méthodes abstraites

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

- à ce niveau, l'opération peut être **abstraite**
- dans ce cas la classe devient aussi **abstraite**
 - ▶ aucun instance directe peut être créée
- Si une classe possède une opération abstraite, l'une de ses sous-classe doit définir concrètement cette opération

Redéfinition

Redéfinition

Une méthode héritée peut être redéfinie

Redéfinition

Une méthode héritée peut être redéfinie

- Dans un but de restriction
 - ▶ E.g. `scale(x,y)` ne fonctionne pas dans `Circle`

Redéfinition

Une méthode héritée peut être redéfinie

- Dans un but de restriction
 - ▶ E.g. `scale(x, y)` ne fonctionne pas dans `Circle`
- Dans un but d'extension
 - ▶ `SavingsAccount` peut inclure des frais additionnels dans le cas d'un retrait

Redéfinition

Une méthode héritée peut être redéfinie

- Dans un but de restriction
 - ▶ E.g. `scale(x, y)` ne fonctionne pas dans `Circle`
- Dans un but d'extension
 - ▶ `SavingsAccount` peut inclure des frais additionnels dans le cas d'un retrait
- Dans un but d'optimisation
 - ▶ `getPerimeterLength` dans `Circle` est beaucoup plus simple que celle de `Ellipse`

Méthode exécutée

1. Vérifier si il existe une définition pour cette méthode dans la classe de l'objet de référence
2. Sinon, vérifier dans la super-classe de niveau immédiatement supérieur
3. Répéter l'étape 2, en remontant les niveaux jusqu'à ce qu'une méthode concrète soit trouvée
4. Si aucune méthode concrète n'est trouvée, il y a erreur

Liaison dynamique

- Se produit lorsque la décision concernant la méthode à exécuter se prend lors de l'exécution du programme
 - ▶ lorsqu'une variable référence un objet dont la classe d'appartenance fait partie d'une hiérarchie
 - ▶ et lorsque plus d'une méthode existe pour une même opération (polymorphique)

Concepts clé POO

- **Objets** (ou Identités)
 - ▶ chaque objet a une existence distincte
- **Classes**
 - ▶ décrire des ensembles d'objets
- **Héritage**
 - ▶ mécanisme permettant d'hériter des propriétés et comportements dans une hiérarchie de classes
- **Polymorphisme**
 - ▶ mécanisme suivant lequel il peut exister plusieurs méthodes pour la même opération

Concepts POO

- **Abstraction**

- ▶ Objet ➔ du monde réel
- ▶ Classe ➔ objets
- ▶ Opérations ➔ méthodes
- ▶ Attributs et associations ➔ variables d'instances

Concepts POO

Concepts POO

- **Modularité**
 - ▶ le programme se construit entièrement à l'intérieur de classes

Concepts POO

- **Modularité**

- ▶ le programme se construit entièrement à l'intérieur de classes

- **Encapsulation**

- ▶ tous les détails sont cachés à l'intérieur des classes
- ▶ principe de masquage de l'information : un programmeur n'a pas besoin de connaître l'intérieur d'une classe pour en faire usage

Style de programmation

- Un programme est fait pour être lu
 - ▶ toujours retenir l'alternative la plus simple
 - ▶ écarter toute approche aussi futée soit elle, mais difficile à saisir
 - ▶ un programme plus court n'est pas nécessairement meilleur
- Choisir des noms appropriés
 - ▶ ils doivent être très descriptifs

Style de programmation

- Commenter
 - ▶ tout ce qui pourrait ne pas être évident
 - ▶ les commentaires peuvent représenter de 25% à 50% du programme
- Organiser les classes de façon consistante
 - ▶ variables, constructeurs, méthodes publiques et méthodes privées
- Structurer le code de façon consistante

Style de programmation

- Éviter toute duplication de code
- Ne pas cloner le code
- Créer plutôt une méthode (privée) regroupant le code commun

Style de programmation

- Se conformer aux bons principes de l'orienté objet
- Minimiser le nombre de méthodes publiques
- Bien séparer les interfaces et l'application