

Objets

Olivier Cailloux

LAMSADE, Université Paris-Dauphine

Version du 8 février 2021

Rôle des classes

Deux rôles des classes ?

Rôle des classes

Deux rôles des classes ? Classe : conteneur à méthodes statiques

- `Math.random`, `Math.abs`...
- Pour organiser

Classe : représente une idée

- L'idée de radiateur \neq un radiateur concret
- L'idée de radiateur est associée à des propriétés
- Décrivent un radiateur : taille, couleur, allumé / éteint...
- Idée de radiateur : pas de valeur pour ces propriétés
- Radiateur concret : valeur pour chaque propriété
- Classe `Heater` : idée, modèle abstrait, non instancié
- Objet `heater` : instance de radiateur

État et comportement

- Un objet a un état : l'ensemble des valeurs de ses propriétés
- Exemple : taille = 60 cm ; couleur = blanc ; allumé
- Un objet a un comportement : l'ensemble de ses méthodes
- Exemple : `isOn(): bool`; `getEnergySpent(double seconds): double`; `powerOn()`
- Classe définit : états possibles + comportement
- Toutes les instances ont le même comportement sauf héritage
- Mais toutes n'ont pas le même état

Construction

- Une classe a *généralement* (au moins) un constructeur
- Construit, initialise les instances de la classe
- Exemple : un nouveau radiateur est toujours allumé \Rightarrow initialiser la propriété correspondante dans le constructeur
- Attention à initialiser toutes vos propriétés (valeurs par défaut)
- Ou fournir un constructeur avec paramètres

Conception

- Réfléchir à l'interface *publique* de votre classe
- C-à-d la façon dont les programmeurs (vous ou d'autres) vont l'utiliser
- À distinguer de son implémentation, détail inutile pour l'utilisateur
- Votre objet doit être une boîte noire pour l'utilisateur, sauf concernant son interface publique
- Exemple : ajouter `getPower()` ? Ou utile seulement en interne ?
- Plus grande interface : plus de documentation à apporter, plus de méthodes à maintenir, plus de complexité
- Bonne pratique : interdire aux utilisateurs de changer l'état de votre objet directement
- Exemple : faire utiliser `heater.powerOn()` plutôt que `heater.on ← true`

En Java

- Distinguer méthodes `private` et `public` et `package`
- Champs (propriétés) : privilégier `private`
- Initialiser : utiliser `new` avec un constructeur (`new Heater(true);`)
- Implémenter méthode `toString(): String`

Variables

- 4 attributs d'une variable ?
- Type primitif (`int`, `double`, ...) : valeur dans sa case mémoire
- Type objet : valeur est une référence vers un objet (sur le tas)
- Deux variables réfèrent à un même objet \neq deux variables réfèrent à deux objets dans le même état
- Destruction : par GC (?)

Variables

- 4 attributs d'une variable ? Nom, type, espace mémoire sur la pile, valeur
- Type primitif (`int`, `double`, ...) : valeur dans sa case mémoire
- Type objet : valeur est une référence vers un objet (sur le tas)
- Deux variables réfèrent à un même objet \neq deux variables réfèrent à deux objets dans le même état
- Destruction : par GC (?)

Variables

- 4 attributs d'une variable ? Nom, type, espace mémoire sur la pile, valeur
- Type primitif (`int`, `double`, ...) : valeur dans sa case mémoire
- Type objet : valeur est une référence vers un objet (sur le tas)
- Deux variables réfèrent à un même objet \neq deux variables réfèrent à deux objets dans le même état
- Destruction : par GC (? Garbage Collector)
- Quand plus aucune référence vers l'objet

Interfaces explicites

- Java distingue Class et Interface
- Le deuxième : seulement des en-têtes de méthodes ou default methods
- Exemple : classe Heater et interface IHeater
- Rend explicite \neq entre interface et implémentation
- Indiquez que votre classe implémente votre interface
- L'utilisateur de votre classe ne dépend que de l'interface
- Il ne doit pas avoir votre classe pour compiler son code
- Vous pouvez fournir la classe plus tard
- Remplacement possible par une autre classe sans recompilation par l'utilisateur

Licence

Cette présentation, et le code LaTeX associé, sont sous licence MIT. Vous êtes libres de réutiliser des éléments de cette présentation, sous réserve de citer l'auteur.

Le travail réutilisé est à attribuer à Olivier Cailloux, Université Paris-Dauphine.