Université Nice Sophia Antipolis

Patrons de conception: décorateur

F. Mallet

miage.m1@gmail.com

http://deptinfo.unice.fr/~fmallet/

Université Nice Sophia Antipolis

Motivation

- Les besoins pour une bonne conception et du bon code :
  - Extensibilité
  - Flexibilité
  - Facilité à maintenir
  - Réutilisabilité
  - Les qualités internes
  - Meilleure spécification, construction, documentation

Université Nice Sophia Antipolis

Historique

- MVC
- Gang of Four : Erich Gamma, Richard Helm,Ralph Johnson, John Vlissides
  - Définition de 23 patterns
- Design Patterns – Elements of Reusable Object-Oriented Software, Addison Wesley, 1994
- Un Design Pattern nomme, abstrait et identifie les aspects essentiels d'une structuration récurrente, ce qui permet de créer une modélisation orientée objet réutilisable

Université Nice Sophia Antipolis

Classification

- Création
  - Comment un objet peut être créé
  - Indépendance entre la manière de créer et la manière d'utiliser
- Structure
  - Comment les objets peuvent être combinés
  - Indépendance entre les objets et les connexions
- Comportement
  - Comment les objets communiquent
  - Encapsulation de processus (ex : observer/observable)

Université Nice Sophia Antipolis

Patrons de structure

- **Adapter pattern**
  - adapter une interface à une autre
- **Bridge pattern**
  - conserver l'interface d'un programme client tandis que le comportement du programme serveur peut être changé.
- **Composite pattern**
  - composer des objets ensembles (structure d'arbre)
- **Decorator pattern**
  - encapsuler dynamiquement des objets et leur fournir de nouvelles fonctions
- **Façade pattern**
  - regrouper une hiérarchie complexe en une interface simple depuis l'extérieur
- **Flyweight pattern**
  - limiter la prolifération d'instances petites, simples et similaires en factorisant hors des classes des données passées en paramètre lors d'appel de méthode
- **Proxy pattern**
  - remplacer un objet le temps de le créer

Université Nice Sophia Antipolis

Décorateur / Decorator

- Intention
  - Attacher dynamiquement des capacités additionnelles à un objet
  - Fournir ainsi une alternative flexible à l'héritage pour étendre les fonctionnalités
- Exemple
  - Ajout de capacités pour objet individuellement et dynamiquement
  - Englober l'objet existant dans un autre objet qui ajoute les capacités (plus que d'hériter)
  - ex : JScrollPane
- *Synonyme : Wrapper (attention !)*

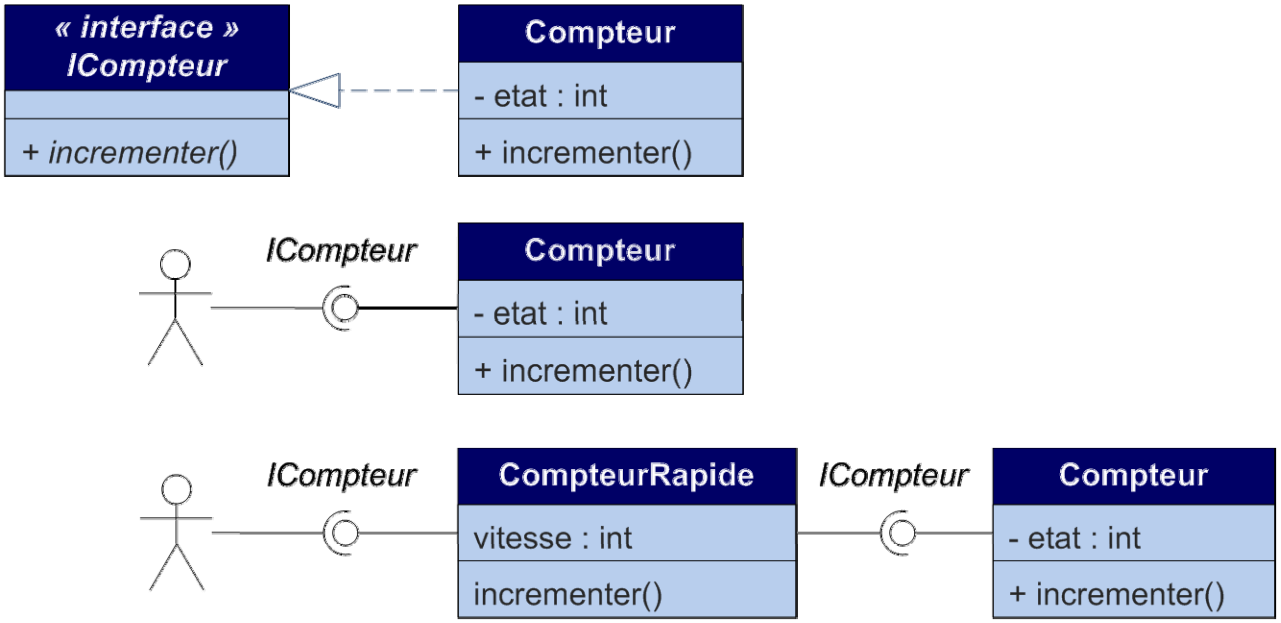
Champs d'application

Champs d'application

- Pour ajouter des capacités de manière transparente
- Pour des capacités qui peuvent être retirées
- Quand l'extension par héritage produirait un nombre trop important d'extensions indépendantes
- Quand l'héritage est interdit

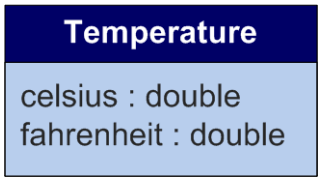
Example1

- Décorateur : **`Compteur`**
  - Ajout de propriétés sans perturber l'utilisateur

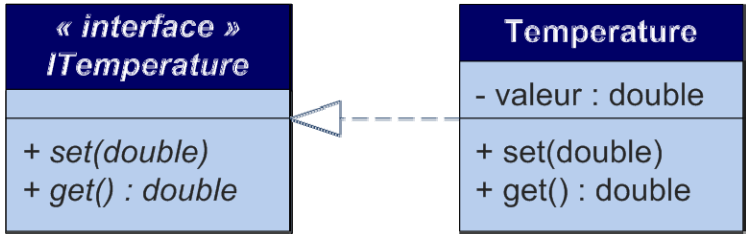


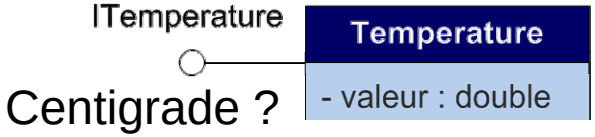
Example2

- Décorateur : **`Temperature`**
  - Ajout de propriétés sans perturber l'utilisateur



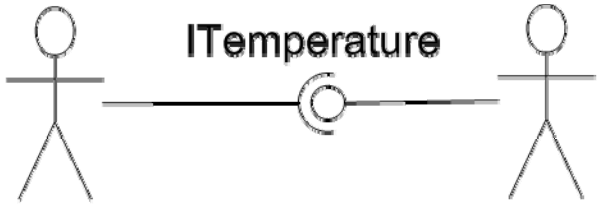
Consistance ?
Kelvin ? Centigrade ? Rankine ? Réaumur ?
Delisle ? Newton ?





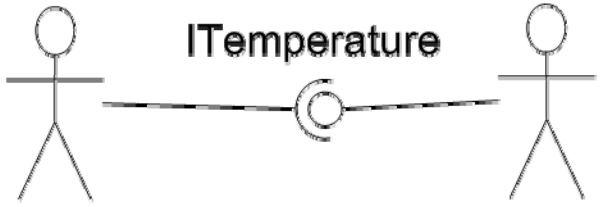
Example2

- Décorateur : **`Temperature`**
  - Ajout de propriétés sans perturber l'utilisateur



Français **Météo France2**

Celsius ?
Fahrenheit ?

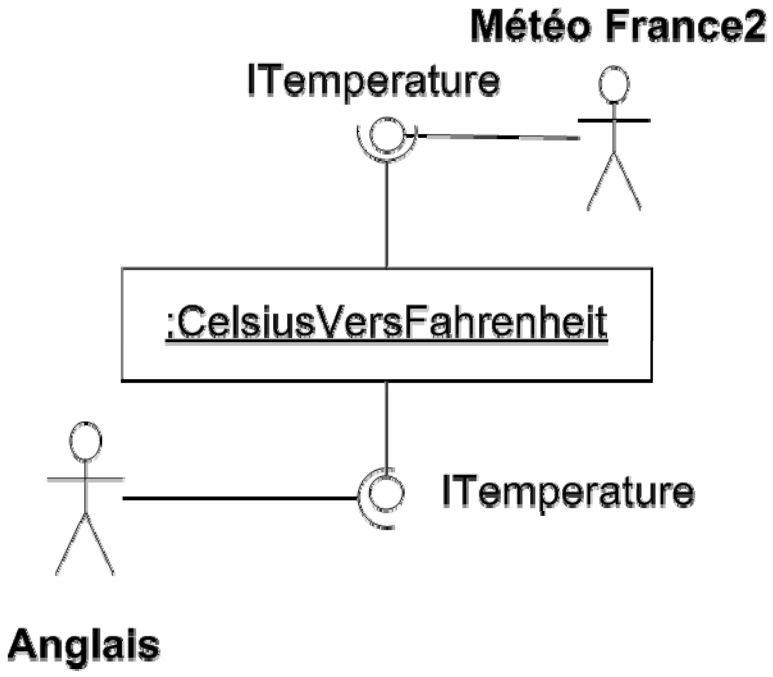


Anglais **NZ Weather Angels**

Celsius ?
Fahrenheit ?

Example2

- Décorateur : **`Temperature`**
  - Ajout de propriétés sans perturber l'utilisateur



Université Nice Sophia Antipolis

En Java

Temperature

```
class Temperature
            implements ITemperature {
  private double valeur;
  Temperature(double v) {
    this.valeur = v;
  }

  public double get() {
    return valeur;
  }
  public void set(double v) {
    this.valeur = v;
  }
}
```

CelsiusVersFahrenheit

```
class CelsiusVersFahrenheit
            implements ITemperature {
  private ITemperature celsius;
  CelsiusVersFahrenheit(ITemperature c){
    this.celsius = c;
  }

  public double get() {
    return celsius.get()*9/5+32;
  }
  public void set(double fahr) {
    celsius.set((fahr-32)*9/5);
  }
}
```

Example 3

- Flot de caractères

```
interface FlotCaractere {
  void print(String s);
}
class VersWriter implements
    FlotCaractere {
  PrintWriter pw;

  VersFichier(Writer w) {
    PrintWriter pw =
                  new PrintWriter(w);
  }
  public void print(String s) {
    pw.println(s);
  }
}
```

Adaptateur !

- Codage de César

```
class CodeCesar implements
    FlotCaractere {
  int cle;
  FlotCaractere flot;
  CodeCesar(int cle,
                  FlotCaractere flot) {
    this.cle = cle;
    this.flot = flot
  }
  public void print(String s) {
    char[] b = s.toCharArray();
    for(int i=0; i<b.length;i++){
      int c=((b[i]-'A')+cle)%25;
      b[i] = c+'A';
    }
    flot.print(new String(b));
  }
}
```

Décorateur !

Example 3

- Flot de caractères

```
interface FlotCaractere {
  void print(String s);
}
class VersWriter implements
    FlotCaractere {
  PrintWriter pw;

  VersFichier(Writer w) {
    PrintWriter pw =
                  new PrintWriter(w);
  }
  public void print(String s) {
    pw.println(s);
  }
}
```

Adaptateur !

- Décodage de César

```
class DecodeCesar implements
    FlotCaractere {
  int cle;
  FlotCaractere flot;
  DecodeCesar(int cle,
                  FlotCaractere flot) {
    this.cle = cle;
    this.flot = flot
  }
  public void print(String s) {
    char[] b = s.toCharArray();
    for(int i=0; i<b.length;i++){
      int c=((b[i]-'A')-cle+25)%25;
      b[i] = c+'A';
    }
    flot.print(new String(b));
  }
}
```

Décorateur !

Université Nice Sophia Antipolis

Résumé

- Implémentation
  - Java : utilisation d'interfaces pour la conformité
  - Pas forcément besoin d'un décorateur abstrait
  - Maintenir une classe de base légère
  - **Decorator** est fait pour le changement d'aspect, **Strategy** est fait pour le changement radical d'approche
- Conséquences
  - Personnalisation d'objets sans héritage
  - Perte de type (perte de la relation « est-un »)
  - Multiplication des classes (les décorations)