JAVA

Applications interactives

-

Modèle Vue-Contrôleur (MVC)

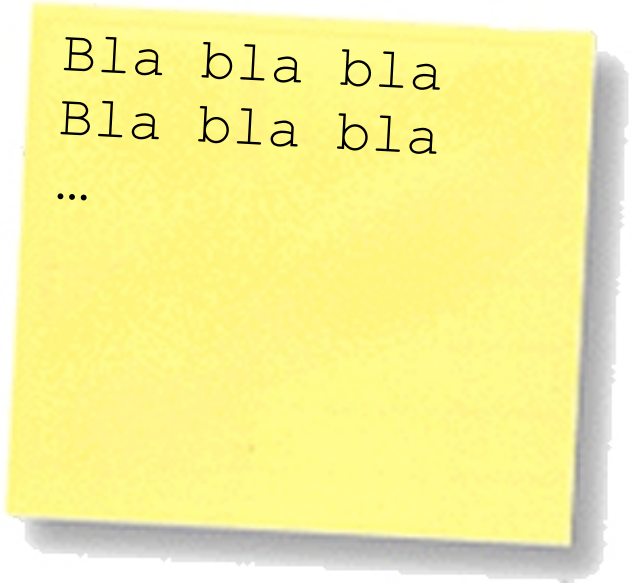
Stéphane HUOT
Dpt. Informatique
iUT ORSAY

Introduction

- 6h00 de cours
  - Programmation d'applications interactives en Java
    - Structure d'une application interactive: le modèle **MVC** *(1-2h00)*
    - Les interfaces graphiques en Java: *AWT* et ***Swing*** *(4-5h00)*

Introduction

- Notions à retenir:



- Point important:

Introduction

- Pas de distribution des transparents du cours

Mais

- Mise en ligne du ‘pdf’ après le cours
- Prenez des notes, je dis des choses intéressantes et utiles parfois…

Plan du cours

1. Structure d'une application interactive
   - Ce que l'on « fait » et que l'on « voit »
   - Ce qu'il se passe

2. Réalisation d'applications interactives: **Modèle-Vue-Contrôleur (MVC)**
   - Le modèle
   - La vue
   - Le contrôleur

3. Utilisation et réalisation de MVC
   - Analyse en terme de MVC
   - Réalisation en Java: 2 exemples pour démarrer

Plan du cours

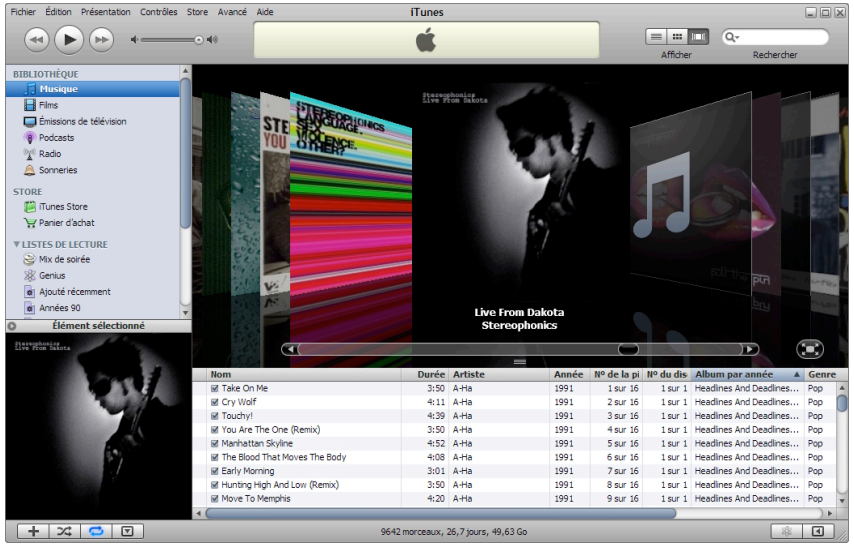
1. Structure d'une application interactive
   - Ce que l'on « fait » et que l'on « voit »
   - Ce qu'il se passe

2. Réalisation d'applications interactives: **Modèle-Vue-Contrôleur (MVC)**
   - Le modèle
   - La vue
   - Le contrôleur

3. Utilisation et réalisation de MVC
   - Analyse en terme de MVC
   - Réalisation en Java: 2 exemples pour démarrer

Application interactive

- Une application avec laquelle l'utilisateur peut **interagir**:
  - L'application effectue des **opérations en réponse aux actions de l'utilisateur**
  - Coopération entre le programme et l'utilisateur, **commandée par l'utilisateur**



Ex: iTunes - ©Apple

  - Est-ce qu'un *shell* (ligne de commandes) est une application interactive ?

Programmer des applications interactives

- Tâche pouvant se révéler complexe:
  - Parce que la tâche que doit accomplir l'utilisateur peut être complexe
  - Parce qu'il faut prévoir les **scenarios d'interaction** (et donc les réactions de l'application)
  - Parce qu'il faut pouvoir **maintenir** et **réutiliser**

⇒ Passage à l'échelle (application importante)

Programmer des applications interactives

- Heureusement, il y a:
  - **Des concepts** (structures et modèles d'applications interactives)
    - MVC
    - PAC
    - Nombreux 'Design patterns' (patrons de conception)
    - …
  - **Des outils** (pour concrétiser ces concepts)
    - Des langages/environnements de programmation adaptés
    - Librairies et leurs APIs (Application Programming Interfaces): boîtes à outils
    - …

Structure d'une application interactive

- La partie 'visible' *(**front office**)*:
  ce que l'on fait et ce que l'on voit
  - Interface Homme-Machine (IHM)

- La partie 'invisible' *(**back office**):*
  ce qu'il se passe
  - Traitements
  - Données (stockage et accès)
  - Communications

Structure d'une application interactive:
Ce que l'on fait et ce que l'on « voit »

- La tâche de l'utilisateur et l'IHM de l'application
  - Interface non graphique:
    - *Ex:* Ligne de commande, tableau de commandes et indicateurs, …
  - Interface graphique (GUI):
    - *Ex:* Application standard Windows, page web, …

- L'utilisateur **'commande'** l'application (programmation événementielle)
  - L'IHM doit être adaptée à la tâche (cf PIG en S2)
  - L'application doit être **'réactive'** (pas de traitements trop longs ou alors les notifier à l'utilisateur)

Structure d'une application interactive:
Ce que l'on fait et ce que l'on « voit »

- Les **entrées** (ce que l'on fait)

et

- Les **sorties** (ce que l'on voit)

Les entrées

. . .

Les sorties

. . .

Exemple: Jeu d'échecs

- La tâche globale: jouer aux échecs!
  - Sous-tâche: déplacer les pièces
- Interfaces utilisateur:

```
Please enter move: e2e4


My move is:  E7-E5     0
The Board :
     a b c d e f g h
   +-----------------+
8  | r n b q k b n r |  8
7  | p p p p . p p p |  7
6  | . . . . s . . . |  6
5  | . . . . p . . . |  5
4  | . . . . P . . . |  4
3  | . . . . . . . . |  3
2  | P P P P . P P P |  2
1  | R N B Q K B N R |  1
   +-----------------+
     a b c d e f g h
Please enter move:
```

Ligne de commande

ou

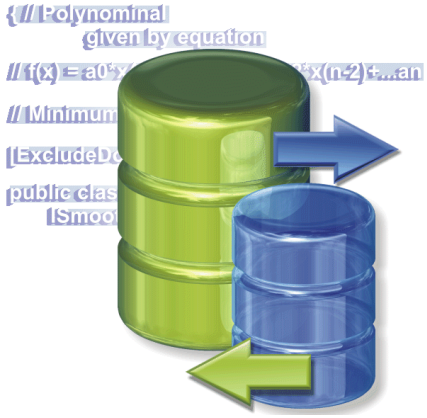
Graphique

Structure d'une application interactive:
Ce qu'il se passe

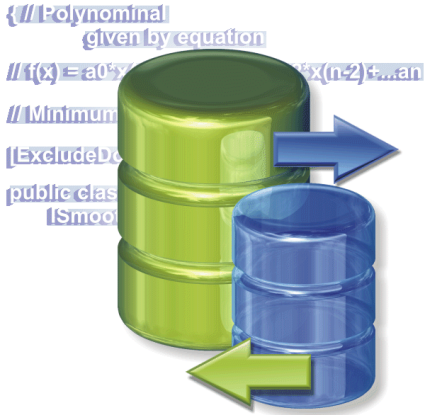
- Le **'noyau'** de l'application:
  - Fonctionnalités
  - Accès aux données
  - Traitement des données
- Les données
- Produit les résultats aux actions de l'utilisateur

Exemple: Jeu d'échecs

- Fonctionnalités:
  - *Jouer* une partie
    - *Déplacer* des pièces
    - *Gérer* les tours de jeu
    - *Joueur virtuel*
    - …
  - *Enregistrer* une partie
  - *Charger* une partie
- Données:
  - État de la partie en cours
  - Parties sauvegardées
  - Catalogues d'ouvertures



Liaison entre ces 2 parties d'une application ?



Liaison entre ces 2 parties d'une application ?
Programmation 'en vrac et comme on peut'...



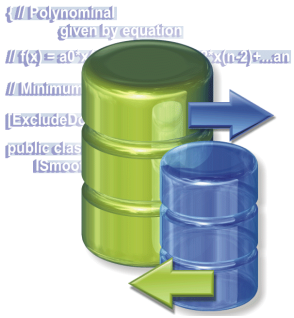




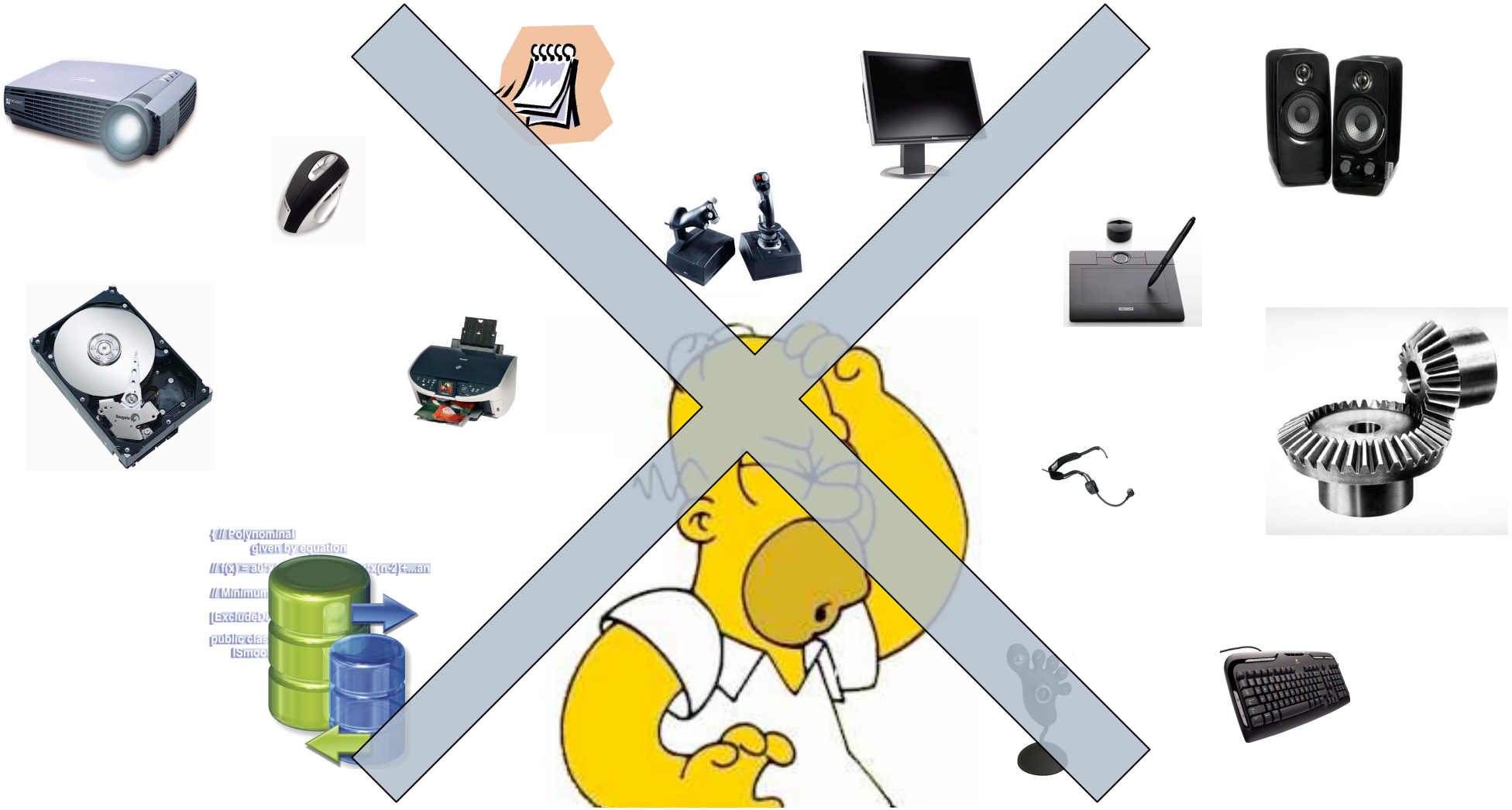








Liaison entre ces 2 parties d'une application ?
Programmation 'en vrac et comme on peut'…



Liaison entre ces 2 parties d'une application ?
… programmation selon un modèle organisé.

Ce qu'il faut retenir

- Application interactive ?
- L'utilisateur à le contrôle
- Développement complexe
- Séparation **Front Office / Back Office**

Plan du cours

1. Structure d'une application interactive
   - Ce que l'on « fait » et que l'on « voit »
   - Ce qu'il se passe

2. Réalisation d'applications interactives: **Modèle-Vue-Contrôleur (MVC)**
   - Le modèle
   - La vue
   - Le contrôleur

3. Utilisation et réalisation de MVC
   - Analyse en terme de MVC
   - Réalisation en Java: 2 exemples pour démarrer

Le modèle 'Modèle-Vue-Contrôleur' (MVC)

- **MVC** est:
  - Un *patron de conception (une solution standardisée à un problème, indépendante des langages de programmation)*,
  - Une *architecture logicielle (une manière de structurer une application ou un ensemble de logiciels)*.
- Introduit en 1979 par *Trygve Reenskaug*,
- Très fortement lié aux concepts de la programmation objet (*Smalltalk*).

MVC: structure du modèle

- Organiser, structurer une application interactive en séparant:
  - Les données et leurs traitements:

    Le Modèle

  - La représentation des données

    La Vue

  - Le comportement de l'application

    Le Contrôleur

Le modèle 'Modèle-Vue-Contrôleur' (MVC)

MODELE

- fonctionnalités de l'application
- accès aux données et traitements

VUE

- présentation des données à l'utilisateur

CONTROLEUR

- gestion des entrées de l'utilisateur
- définit le comportement de l'application

Le modèle 'Modèle-Vue-Contrôleur' (MVC)

MODELE

- fonctionnalités de l'application
- accès aux données et traitements

1-entrées utilisateur

VUE

- présentation des données à l'utilisateur

CONTROLEUR

- gestion des entrées de l'utilisateur
- définit le comportement de l'application

Le modèle 'Modèle-Vue-Contrôleur' (MVC)

MODELE

- fonctionnalités de l'application
- accès aux données et traitements

2-notification de changement d'état

1-entrées utilisateur

VUE

- présentation des données à l'utilisateur

CONTROLEUR

- gestion des entrées de l'utilisateur
- définit le comportement de l'application

Le modèle 'Modèle-Vue-Contrôleur' (MVC)

3-opérations internes

MODELE

- fonctionnalités de l'application
- accès aux données et traitements

2-notification de changement d'état

1-entrées utilisateur

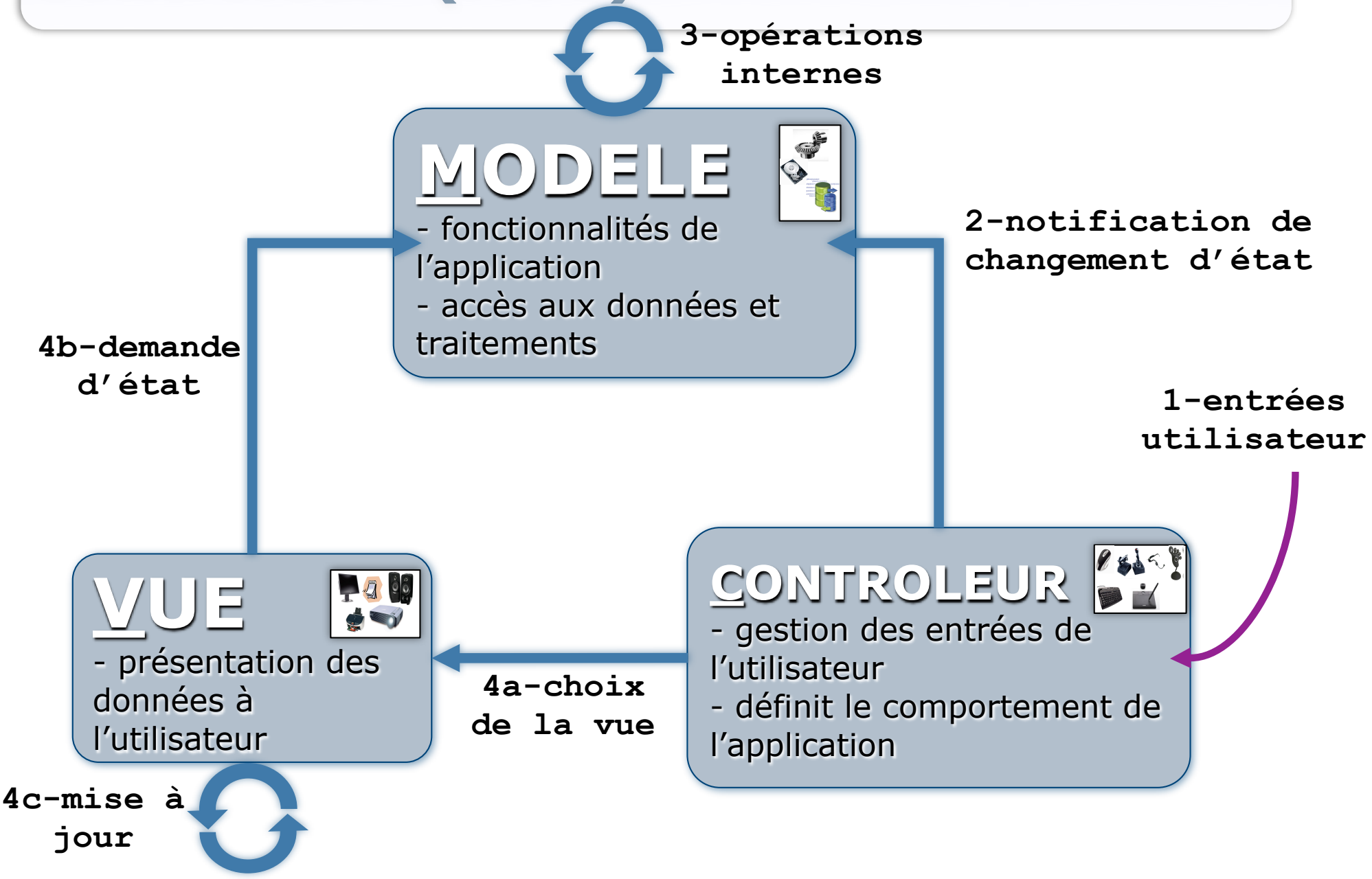
VUE

- présentation des données à l'utilisateur

CONTROLEUR

- gestion des entrées de l'utilisateur
- définit le comportement de l'application

Le modèle 'Modèle-Vue-Contrôleur' (MVC)



Le modèle 'Modèle-Vue-Contrôleur' (MVC) Vision plus concrète

MODELE
- fonctionnalités de l'application
- accès aux données et traitements

demande d'état

notification de changement

notification de changement d'état

VUE
- présentation des données à l'utilisateur

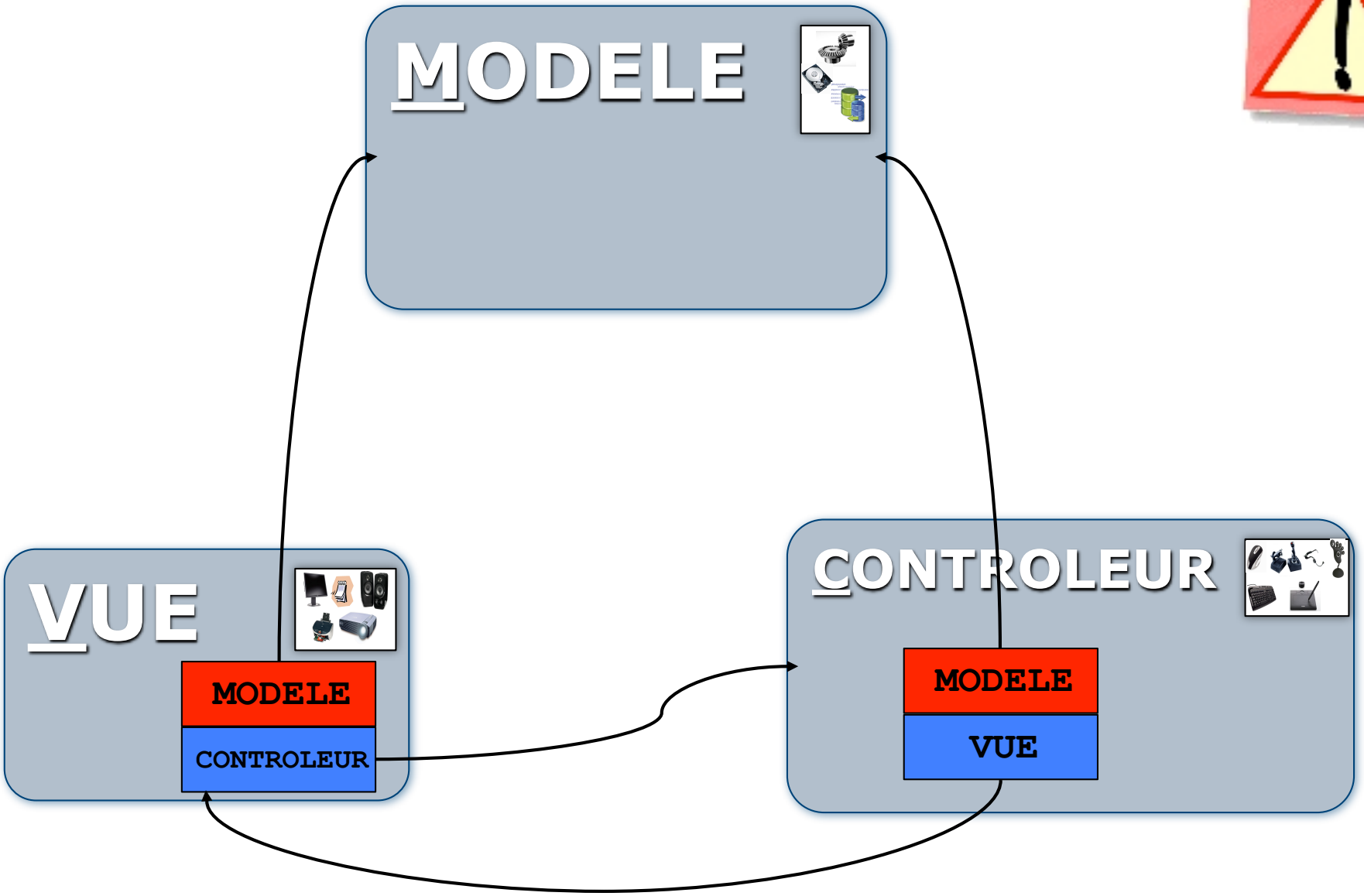
sélection de la vue

CONTROLEUR
- gestion des entrées de l'utilisateur
- définit le comportement de l'application

notification d'entrées

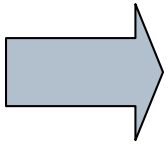
entrées utilisateur

Le modèle 'Modèle-Vue-Contrôleur' (MVC)
Références entre composants



MVC: le modèle

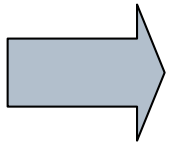
- Le modèle:
  - Représente les données
  - Fournit les accès aux données
  - Fournit les traitements applicables aux données
  - Expose les fonctionnalités de l'application



Noyau Fonctionnel de l'application

MVC: la vue

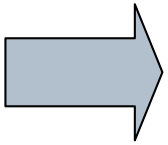
- La vue:
  - Représente la (ou une) représentation des données du modèle
  - Assure la consistance entre la représentation qu'elle donne et l'état du modèle/le contexte de l'application



Sorties de l'application

MVC: le contrôleur

- Le contrôleur:
  - Représente le comportement de l'application face aux actions de l'utilisateur
  - Fournit la traduction des actions de l'utilisateur en actions sur le modèle
  - Fournit la vue appropriée par rapport aux actions de l'utilisateur et des réactions du modèle



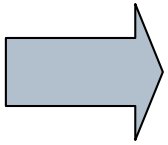
Comportement et gestion des entrées de l'application

Avantages de MVC

- Structure 'propre' de l'application
- **Indépendance** 'données' – 'représentation' – 'comportements'
- Adapté aux concepts de la programmation 0-0
- **Modulaire** et **réutilisable**
  - Vues interchangeables
  - Contrôleurs interchangeables
  - Changement de *'Look & Feel'*
- Facilite les vues et contrôleurs multiples
  - Synchronisation 'quasi-implicite'

Inconvénients de MVC

- Mise en place complexe dans le cas d'applications importantes
- Mises à jour potentiellement trop nombreuses
  - 'Spaghettis' dans le code
  - Temps d'exécution
- Contrôleur et Vue restent souvent fortement liés au Modèle



Adapter la réalisation au problème

Ce qu'il faut retenir

- Structure **modulaire** de MVC

- **Rôle** des composants

- **Liens** entre les composants

Plan du cours

1. Structure d'une application interactive
   - Ce que l'on « fait » et que l'on « voit »
   - Ce qu'il se passe

2. Réalisation d'applications interactives: **Modèle-Vue-Contrôleur (MVC)**
   - Le modèle
   - La vue
   - Le contrôleur

3. Utilisation et réalisation de MVC
   - Analyse en terme de MVC
   - Réalisation en Java: 2 exemples pour démarrer

MVC: utilisation et réalisation

- Comment réaliser une application interactive selon le modèle MVC ?

- Ce qui a déjà été vu: **le modèle**
  - Implantation de modèles
  - Tests unitaires

Conventions de nommage

- Paquetages:
  - Contrôleurs: `package application.controleurs;`
  - Vues: `package application.vues;`
  - Modèle: `package application.modele;`

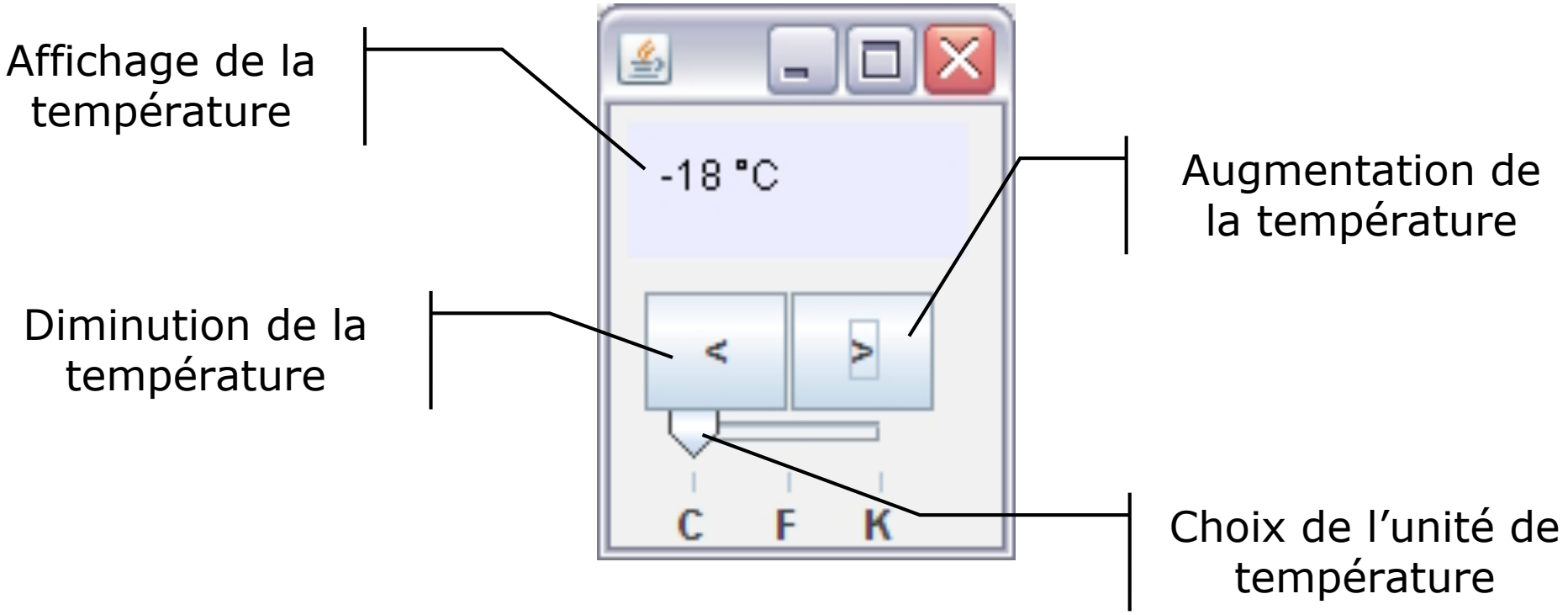
- Classes:
  - Contrôleurs: `ControleurNomClasse.java`
  - Vues: `VueNomClasse.java`
  - Modèle: `ModeleNomClasse.java`

Exemple 1: le thermomètre v1

- Réaliser une application interactive simulant un *thermomètre*, sur laquelle l'utilisateur peut agir pour contrôler la température

- L'application fournira:
  - Un **affichage textuel** de la température courante mesurée par le thermomètre en °C ou en °K ou en F
  - Des **contrôles** permettant à l'utilisateur de **diminuer** et **augmenter** la température courante du thermomètre
  - Un **contrôle** permettant de **choisir l'unité d'affichage** de la température

Thermomètre v1

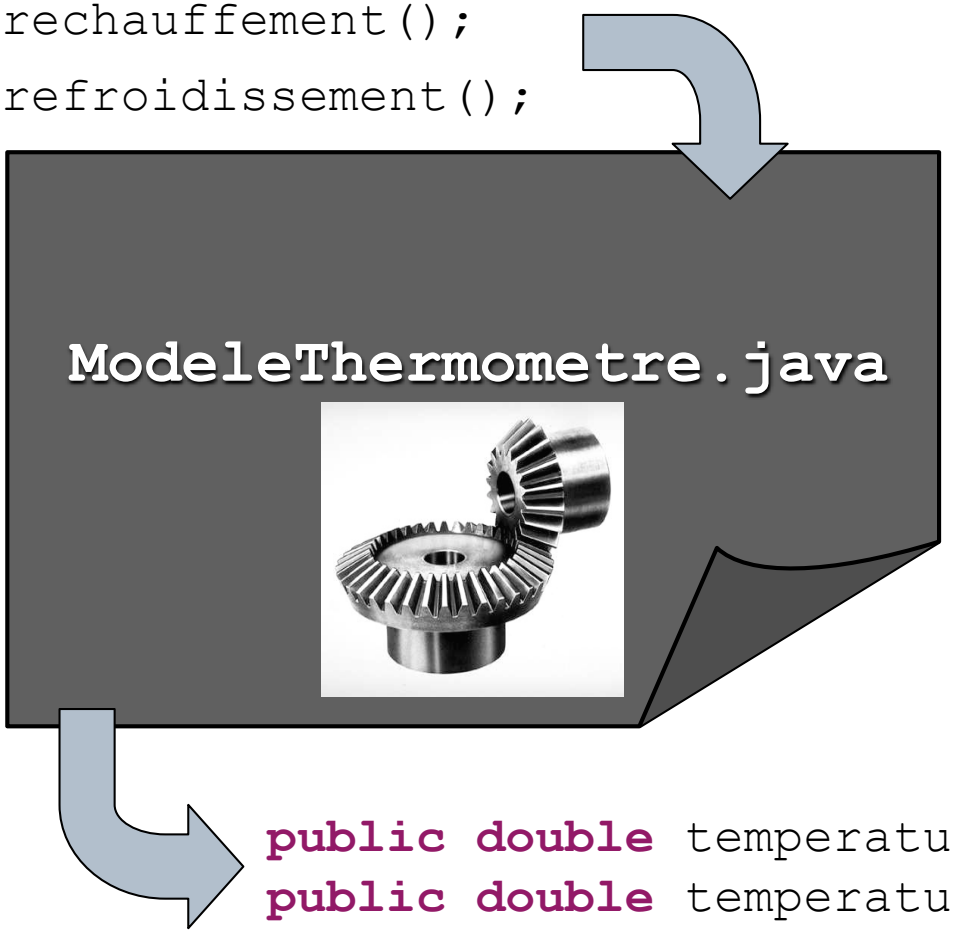


Thermomètre v1: analyse

- Le modèle (cf TP1b)
  - Données et traitements réalisés:
    - *Température courante*
    - *Maintient de l'état* de la température courante
    - *Conversions* de la température en différentes unités
  - Fonctionnalités exposées:
    - **Augmenter** la température de 1° (C ou K)
    - **Diminuer** la température de 1° (C ou K)
    - **Donner** la température en °C, °K ou F

Thermomètre v1: analyse

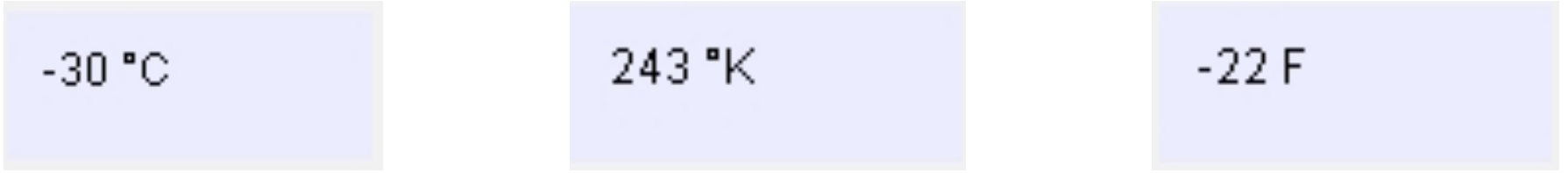
```
public void rechauffement();
public void refroidissement();
```




```
public double temperatureEnKelvin();
public double temperatureEnCelsius();
public double temperatureEnFarenheit();
```

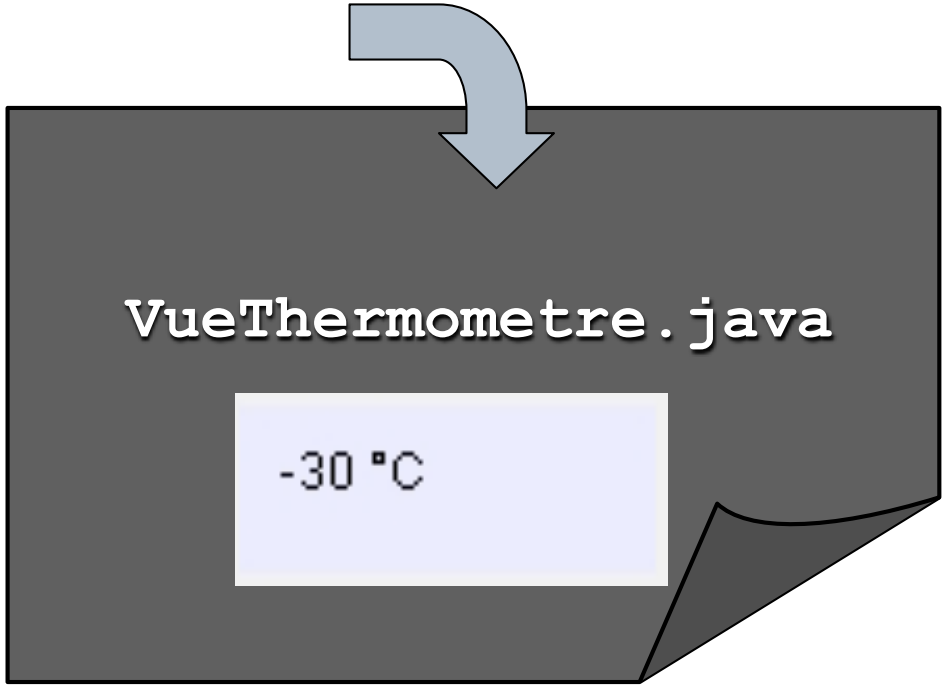
Thermomètre v1: analyse

- La vue
  - **Affiche** la température courante sous forme de texte
  - **Adapte** son affichage à l'unité courante



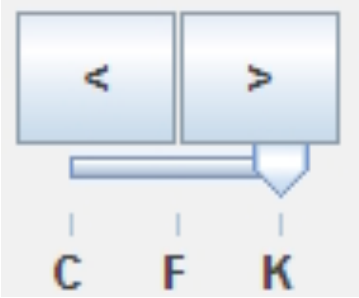
Thermomètre v1: analyse

```
public void redessiner();
public void reglerUnite(Unite unite);
```

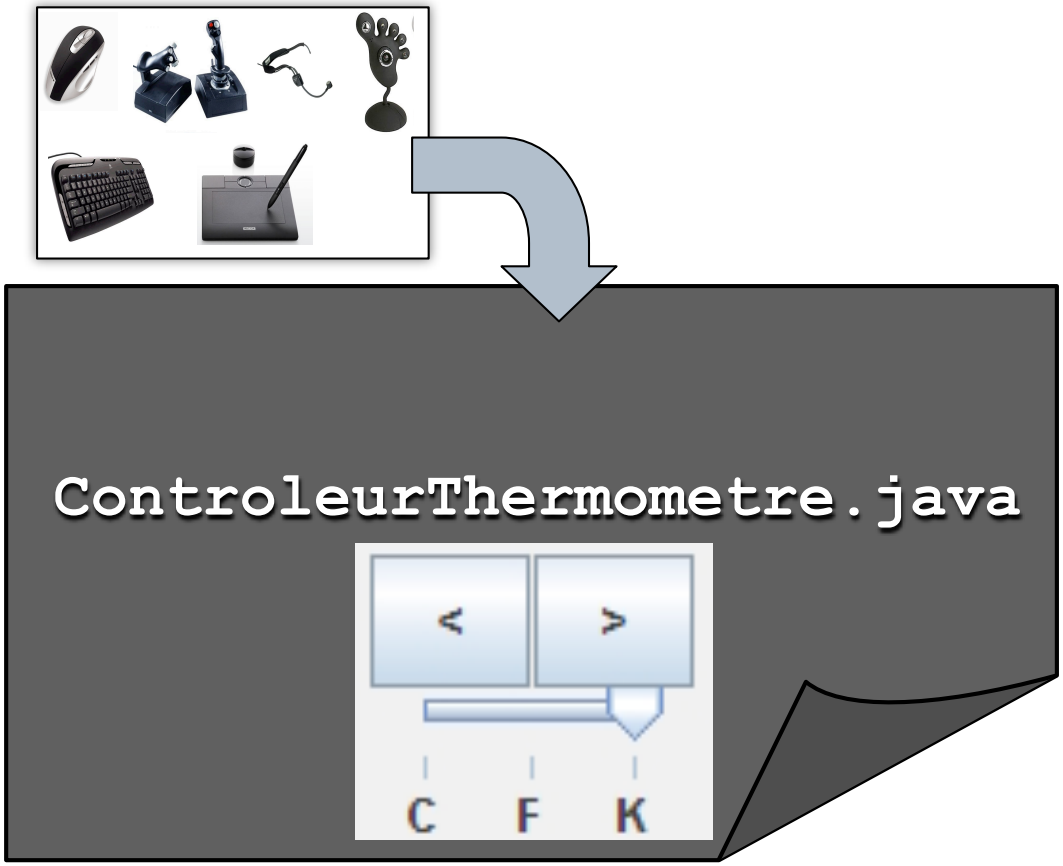


Thermomètre v1: analyse

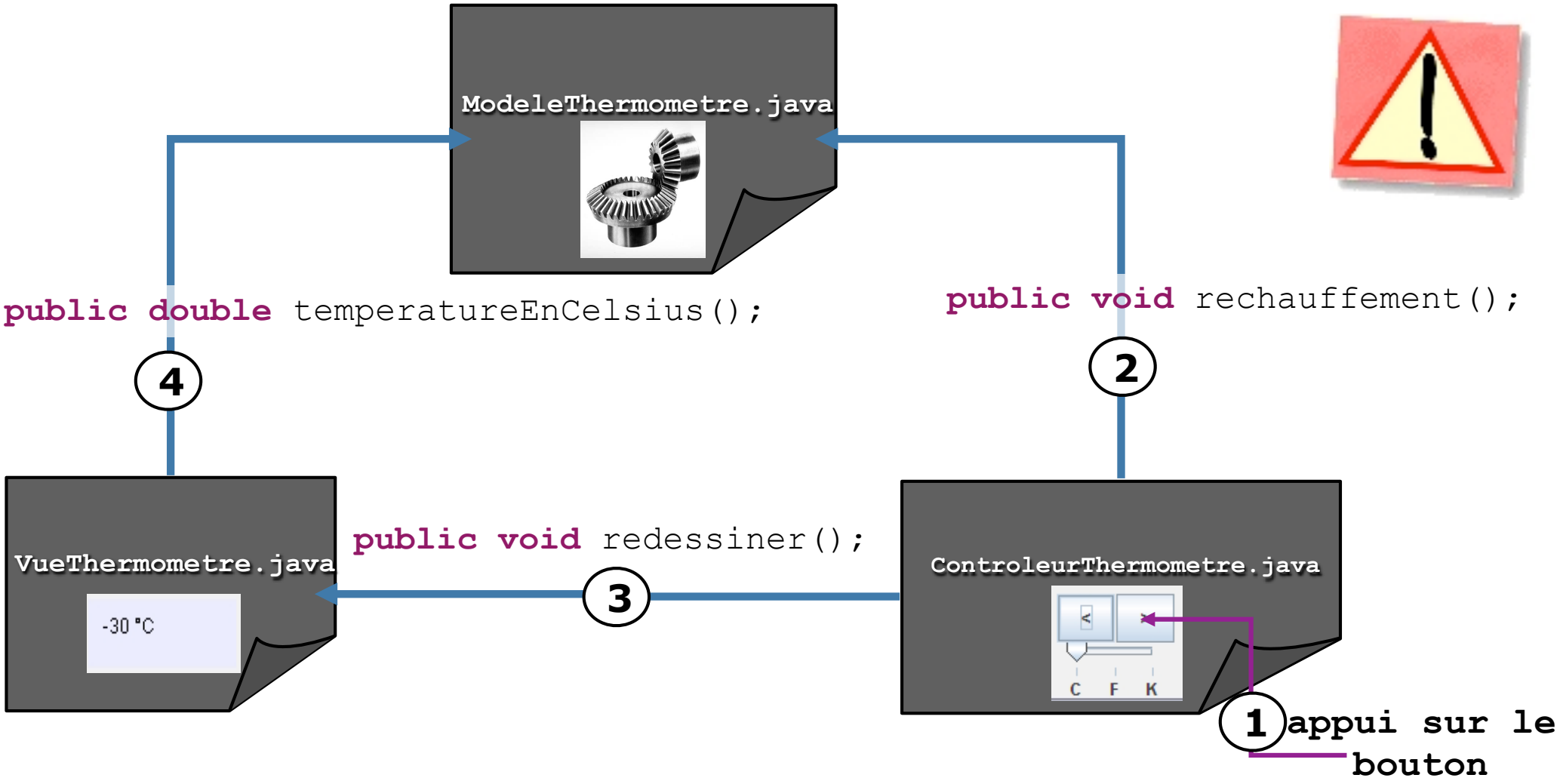
- Le contrôleur
  - **Fournit** à l'utilisateur les **contrôles** sur le modèle: **augmenter** ou **diminuer** la température
  - **Traduit** les actions de l'utilisateur en opération sur le modèle: déclenche les traitements par des appels de méthodes sur le modèle
  - **Sélectionne** et **mets à jour** la vue



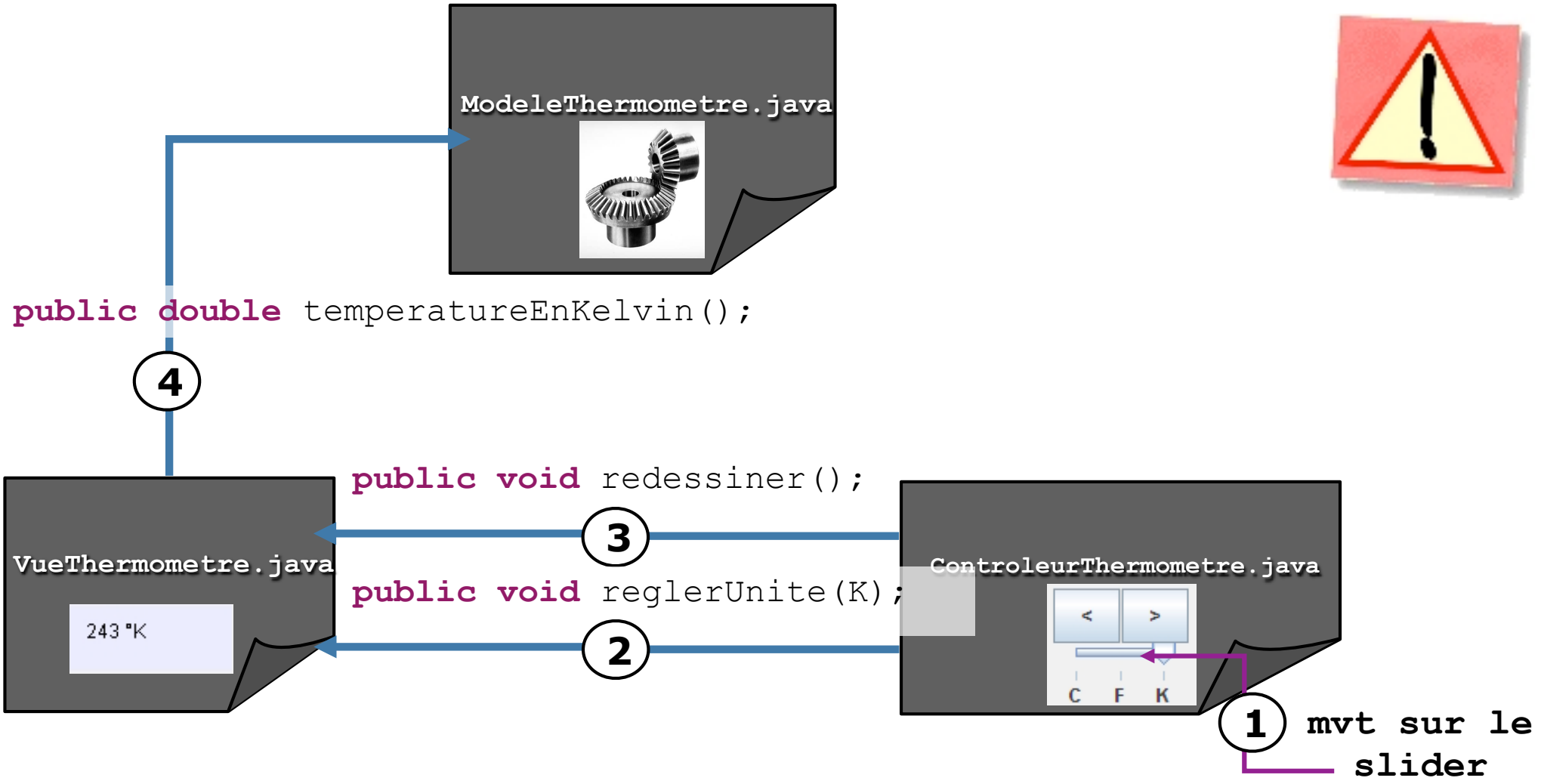
Thermomètre v1: analyse



Thermomètre v1: analyse



Thermomètre v1: analyse



Thermomètre v1: réalisation

- Le modèle (cf TP1b)

```
package thermometre.modele;
public class ModeleThermometre {
 private double _temperature;//la temperature en Kelvin
//… (constructeurs)
 public double temperatureEnKelvin(){
        return _temperature;
 }
  public double temperatureEnCelsius(){
        return _temperature - 273.15;
 }
 public double temperatureEnFarenheit(){
        return (9 / 5d) * _temperature - 459.67;
 }
 public void rechauffement(){
         _temperature++;
 }
 public void refroidissement(){
        if (_temperature > 1) {
                        _temperature--;
        }
 }
}
```

Thermomètre v1: réalisation

- La vue

```
package thermometre.vues;

public class VueThermometre {
 private ModeleThermometre _modele;//le modèle à représenter
 private Unite _unite; //l'unité d'affichage courante

 public VueThermometre (ModeleThermometre modele){
       _modele = modele;
 }

 public void reglerUnite (Unite unite){
     _unite = unite;
 }

//suite au prochain transparent...
```

Thermomètre v1: réalisation

- La vue (suite)

```
public void redessiner (){
    double tempCourante = 0;
    switch (_unite) {
                case CELSIUS: tempCourante = _modele.temperatureEnCelsius();
                        break;
                case FAHRENHEIT: tempCourante = _modele.temperatureEnFarenheit();
                        break;
                case KELVIN: tempCourante = _modele.temperatureEnKelvin();
    }

    //opérations de dessin de la vue avec la valeur tempCourante
    //...
 }

}
```

Thermomètre v1: réalisation

- Le contrôleur

```
package thermometre.controleurs;
public class ControleurThermometre {
 private ModeleThermometre _modele;//le modèle à contrôler
 private VueThermometre _vue;//la vue pour représenter le modèle

 JButton _pButton = new JButton(">");//le bouton pour augmenter la température
 JButton _mButton = new JButton("<");//le bouton pour diminuer la température
 JSlider _mSlider = new JSlider(0, 20);//le slider pour choisir l'unité

 public ControleurThermometre (ModeleThermometre modele, VueThermometre vue){
     _modele = modele;
     _vue = vue;
     //placement des contrôles de l'IHM
     //...
 }

//Suite au prochain transparent...
```

Thermomètre v1: réalisation

- Le contrôleur (suite)

```
//Lors d'une action sur le bouton « UP » (les mécanismes d'événements seront détaillés plus tard
      dans ce cours)
 public void boutonUpActivé (){
     _modele.rechauffement();
     _vue.redessiner();
 }


//Lors d'une action sur le bouton « UP » (les mécanismes d'événements seront détaillés plus tard
      dans ce cours)
 public void boutonDownActivé (){
     _modele.refroidissement();
     _vue.redessiner();
 }


//Lors d'une action sur le bouton « UP » (les mécanismes d'événements seront détaillés plus tard
      dans ce cours)
 public void sliderActivé (){
     _vue.reglerUnite(uniteDuSlider);
     _vue.redessiner();
 }
}
```

Thermomètre v1: réalisation

- L'application

```
package thermometre;
import java.awt.GridLayout;
import javax.swing.JFrame;
import thermometre.controleurs.ControleurThermometre;
import thermometre.modele.ModeleThermometre;
import thermometre.vues.VueThermometre;

public class AppliThermometreSimple {

        public static void main(String[] args) {
                        //Creation d'une fenêtre pour l'application
                        JFrame frame = new JFrame();

                        //Creation d'un modèle de thermomètre
                        ModeleThermometre modele = new ModeleThermometre(243.15);

                        //Creation de la vue et du contrôleur
                        VueThermometre vue = new VueThermometre (modele);
                        final ControleurThermometre pt1 = new ControleurThermometre(modele, vue);

                        //Ajout des panneaux à la fenêtre et affichage...
                        //...
        }
}
```

Commentaires sur cette réalisation de MVC ?

- Validité par rapport au modèle
- Problèmes

Commentaires sur cette réalisation de MVC

- Mécanismes de gestion des événements => *seront traités dans la suite du cours*
- Modularité => *Oui*
- Indépendance Données-Vue-Interaction ? => *Oui*

Commentaires sur cette réalisation de MVC

- **Réutilisabilité** et **généricité** ?
  - Si l'on veut utiliser un autre modèle, une autre vue ou un autre contrôleur ?

- Par exemple, si l'on veut maintenant:
  - Utiliser un autre modèle de thermomètre avec cette vue et ce contrôleur ?
  - Utiliser une autre vue avec ce modèle et ce contrôleur ?
  - Utiliser un autre contrôleur avec ce modèle et cette vue ?
  - Utiliser une autre vue et un autre contrôleur simultanément avec cette vue, ce modèle et ce contrôleur ?
  - *Etc.*

Commentaires sur cette réalisation de MVC

- **Réutilisabilité** et **généricité** ? => *Non!*
  - Si l'on veut utiliser un autre modèle, une autre vue ou un autre contrôleur ?

- Par exemple, si l'on veut maintenant:
  - Utiliser un autre modèle de thermomètre avec cette vue et ce contrôleur ?
  - Utiliser une autre vue avec ce modèle et ce contrôleur ?
  - Utiliser un autre contrôleur avec ce modèle et cette vue ?
  - Utiliser une autre vue et un autre contrôleur simultanément avec cette vue, ce modèle et ce contrôleur ? (synchronisation)
  - *Etc.*

- Impossible en l'état car les références entre les objets de chaque module sont typées par leur classe concrète. **On doit modifier des parties du code de chaque module.**

Valide par rapport au modèle, mais n'en n'exploite pas le pouvoir d'abstraction (indépendance entre données/représentation/contrôle)

Réalisation 'propre' de MVC

- Utiliser des *Interfaces* et des *classes abstraites*
  - **Abstraction des implantations** de chaque module
  - Interfaces: contrats que doivent respecter les modules pour assurer leur **interopérabilité**

- Remarque:

Interface Utilisateur ≠ Interface en O-O-P

Exemple 2: le thermomètre v2

- Réaliser une application interactive simulant **2 thermomètres** sur lesquels l'utilisateur peut agir pour contrôler la température
- L'application fournira:
  - Un affichage de la température courante mesurée par les thermomètre en °C ou en °K ou en F sous la forme:
    - D'un **thermomètre à mercure** pour le 1er thermomètre
    - D'un **cadran à aiguille** pour le 2nd thermomètre
  - Des contrôles permettant à l'utilisateur de diminuer et augmenter la température courante **de chaque** thermomètre
  - Un contrôle permettant de choisir l'unité d'affichage de la température pour **chaque vue**

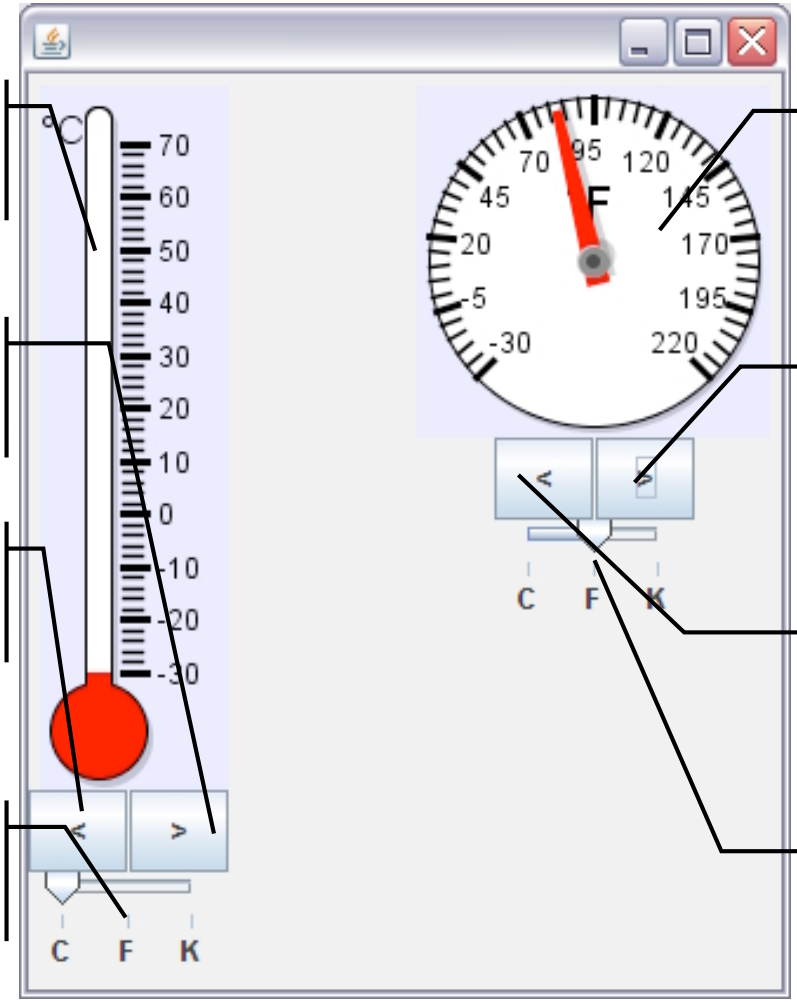
Thermomètre v2

Affichage de la température thermo 1

Augmentation de la température thermo 1

Diminution de la température thermo 1

Choix de l'unité de température thermo 1



Affichage de la température thermo 2

Augmentation de la température thermo 2

Diminution de la température thermo 2

Choix de l'unité de température thermo 2

Thermomètre v2: analyse

- Le modèle
  - Identique au précédent
- Le contrôleur
  - Identique au précédent
  - MAIS doit opérer sur des vues potentiellement différentes
- La vue
  - Doit fournir différentes vue d'un même type de modèle (mais pas forcément la même instance)

Thermomètre v2: analyse

- Solutions pour la vue:
  1. Implanter des vues différentes dans la même classe `VueThermometre.java`

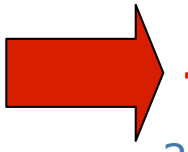
     Solution lourde et peu flexible
  2. Implanter des classes de vues différentes `VueThermometreMercure.java` et `VueThermometreCompteur.java`

     Il faut alors 2 contrôleurs (lourd et peu flexible)
  3. Fournir une **interface** `VueThermometre.java` spécifiant une vue de thérmomètre et spécifiant les prototypes de ces méthodes et l'implanter selon les besoins

     Bonne solution

Thermomètre v2: analyse

- Solutions pour la vue:
  1. Implanter des vues différentes dans la même classe `VueThermometre.java`



~~*Solution lourde et peu flexible*~~

  2. Implanter des classes de vues différentes `VueThermometreMercure.java` **et** `VueThermometreCompteur.java`

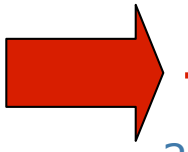
Il faut alors 2 contrôleurs (lourd et peu flexible)

  3. Fournir une **interface** `VueThermometre.java` spécifiant une vue de thérmomètre et spécifiant les prototypes de ces méthodes et l'implanter selon les besoins

Bonne solution

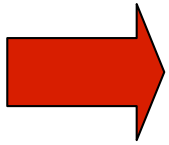
Thermomètre v2: analyse

- Solutions pour la vue:
  1. Implanter des vues différentes dans la même classe `VueThermometre.java`



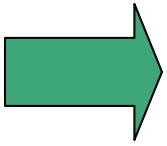
~~*Solution lourde et peu flexible*~~

  2. Implanter des classes de vues différentes `VueThermometreMercure.java` et `VueThermometreCompteur.java`



~~*Il faut alors 2 contrôleurs (lourd et peu flexible)*~~

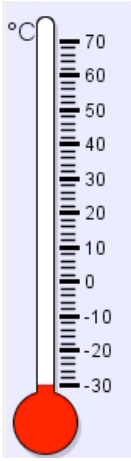
  3. Fournir une **interface** `VueThermometre.java` spécifiant une vue de thermomètre et spécifiant les prototypes de ces méthodes et l'implanter selon les besoins



Bonne solution

Thermomètre v2: analyse

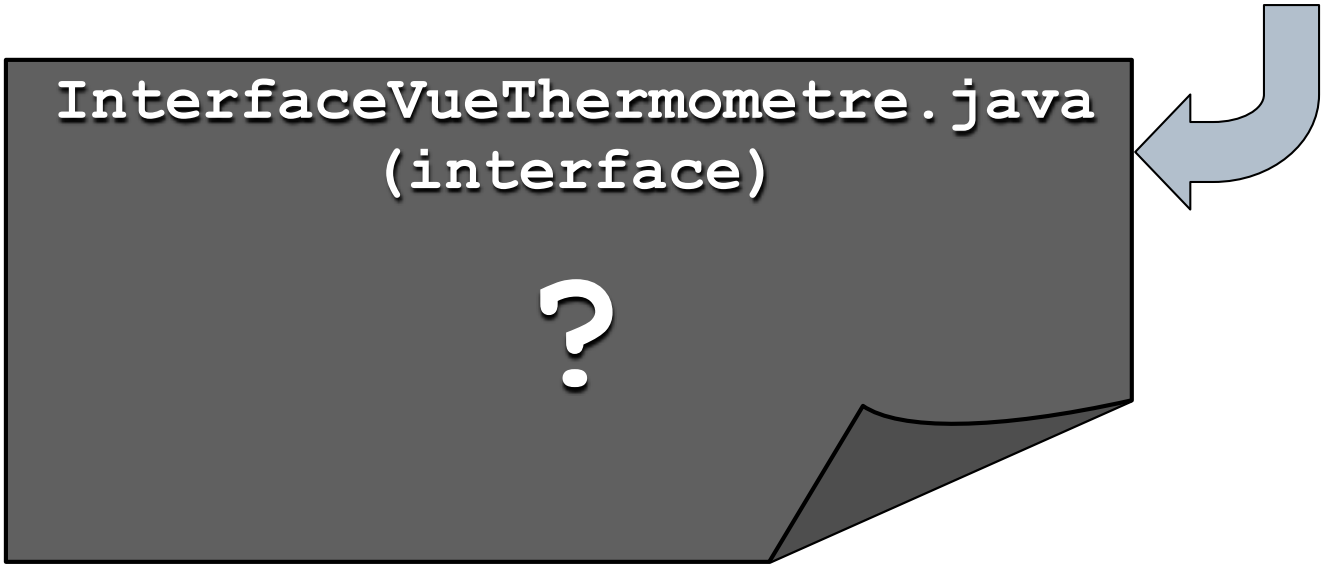
- Tous les types de vues
  - **Affichent** la température courante sous une forme indéterminée
  - **Adaptent** leur affichage à l'unité courante



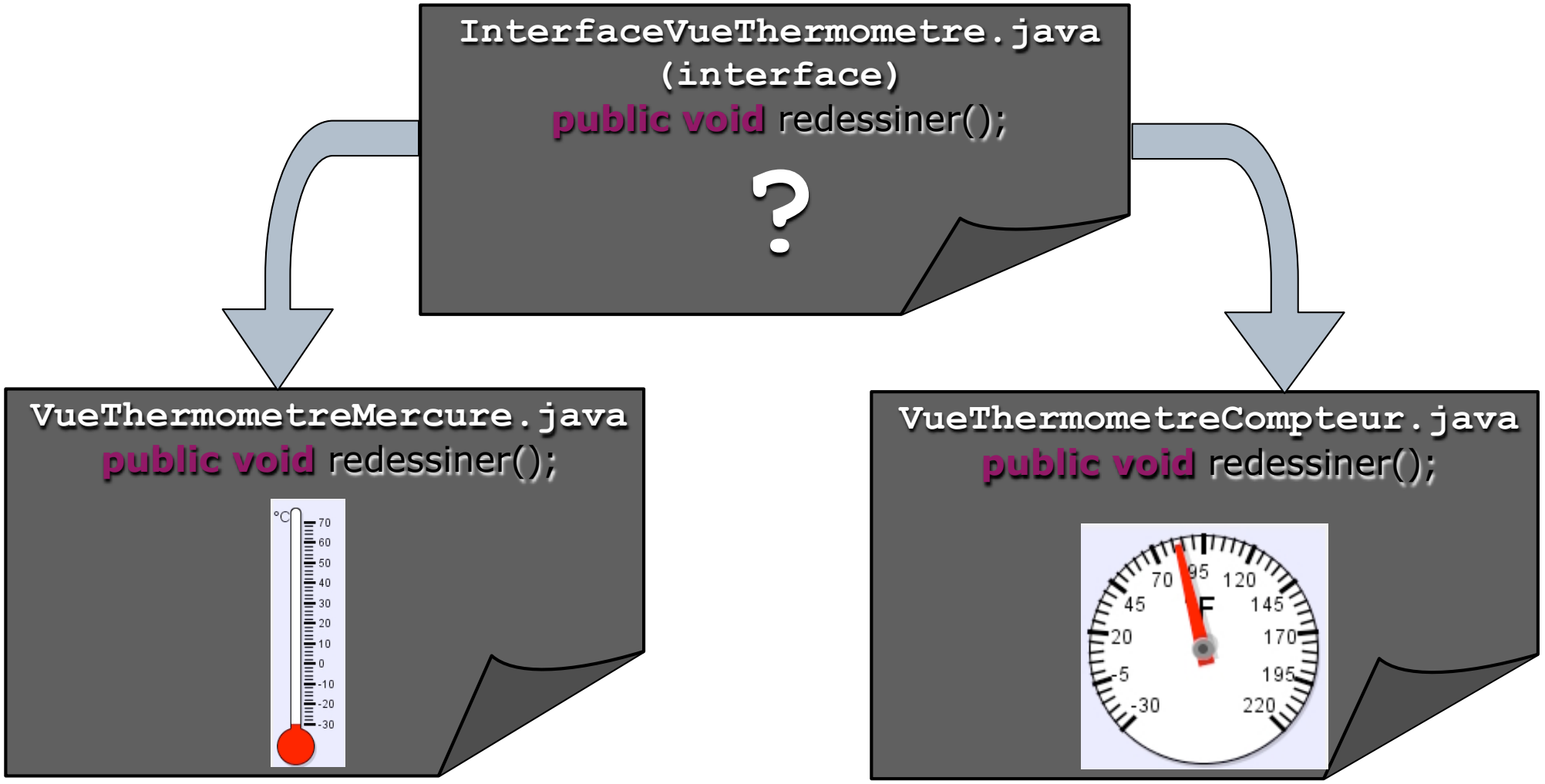
Thermomètre v2: analyse

Methodes requises:

```
public void redessiner();
public void reglerUnite(Unite unite);
```



Thermomètre v2: analyse



Thermomètre v2: réalisation

- Les vue(s)

```
package thermometre.vues;

public interface InterfaceVueThermometre {

 //...

 /***
 * Régle l'unité d'affichage du thermometre.
 *
 * @param unite l'unité désirée.
 */
 public void reglerUnite(Unite unite);

 /**
 * Methode pour notifier à la vue de se redessiner
 */
 public void redessiner();

}
```

Thermomètre v2: réalisation

- La vue 'Mercure' `VueThermometreMercure.java`

```
package thermometre.vues;
public class VueThermometreMercure implements InterfaceVueThermometre {
 //...
 protected ModeleThermometre _modele = null;
 protected Unite _unite = Unite.CELSIUS;

 public VueThermometreMercure(ModeleThermometre modele) {
  this._modele = modele;
 }

 public void reglerUnite(Unite unite) {
   this._unite = unite;
 }

 public void redessiner (){
        double tempCourante = 0;
        switch (_unite) {
                        case CELSIUS: tempCourante = _modele.temperatureEnCelsius();
                                      break;
                        case FAHRENHEIT: tempCourante = _modele.temperatureEnFarenheit();
                                      break;
                        case KELVIN: tempCourante = _modele.temperatureEnKelvin();
        }
        //Opérations spécifiques de dessin de la vue Mercure...
}
```

Thermomètre v2: réalisation

- La vue 'Compteur' `VueThermometreCompteur.java`

```
package thermometre.vues;
public class VueThermometreCompteur implements InterfaceVueThermometre {
 //...
 protected ModeleThermometre _modele = null;
 protected Unite _unite = Unite.CELSIUS;

 public VueThermometreCompteur(ModeleThermometre modele) {
  this._modele = modele;
 }

 public void reglerUnite(Unite unite) {
   this._unite = unite;
 }

 public void redessiner (){
        double tempCourante = 0;
        switch (_unite) {
                        case CELSIUS: tempCourante = _modele.temperatureEnCelsius();
                                      break;
                        case FAHRENHEIT: tempCourante = _modele.temperatureEnFarenheit();
                                      break;
                        case KELVIN: tempCourante = _modele.temperatureEnKelvin();
        }
        //Opérations spécifiques de dessin de la vue Compteur...
}
```

Thermomètre v2: réalisation

- Le modèle: identique au précédent

- Le contrôleur: identique au précédent mais référence vers `InterfaceVueThermometre`

Thermomètre v2: réalisation

- Le contrôleur

```
package thermometre.controleurs;

public class ControleurThermometre {
 private ModeleThermometre _modele;//le modèle à contrôler
 private InterfaceVueThermometre _vue;//la vue pour représenter le modèle

 JButton _pButton = new JButton(">");//le bouton pour augmenter la température
 JButton _mButton = new JButton("<");//le bouton pour diminuer la température
 JSlider _mSlider = new JSlider(0, 20);//le slider pour choisir l'unité

 public ControleurThermometre (ModeleThermometre modele, InterfaceVueThermometre vue){
     _modele = modele;
     _vue = vue;
     //placement des contrôles de l'IHM
     //...
 }

//Suite idem contrôleur précédent...
```

Thermomètre v2: réalisation

- L'application

```
package thermometre;
import java.awt.GridLayout;
import javax.swing.JFrame;
import thermometre.controleurs.ControleurThermometre;
import thermometre.modele.ModeleThermometre;
import thermometre.vues.*;

public class AppliThermometreSimple {

        public static void main(String[] args) {
                //Creation d'une fenêtre pour l'application
                JFrame frame = new JFrame();

                //Creation des modèles de thermomètre
                ModeleThermometre modele1 = new ModeleThermometre(243.15);
                ModeleThermometre modele2 = new ModeleThermometre();

                //Creation de la vue 'mercure' et de son contrôleur
                InterfaceVueThermometre vue1 = new VueThermometreMercure (modele1);
                ControleurThermometre pt1 = new ControleurThermometre(modele1, vue1);

                //Creation de la vue 'compteur' et de son contrôleur
                InterfaceVueThermometre vue2 = new VueThermometreCompteur (modele2);
                ControleurThermometre pt2 = new ControleurThermometre(modele2, vue2);

                //Ajout des panneaux à la fenêtre et affichage...
                //...
        }
}
```

Commentaires sur cette réalisation de MVC ?

- Validité par rapport au modèle
- Problèmes

Commentaires sur cette réalisation de MVC

- Mécanismes de gestion des événements => *seront traités dans la suite du cours*
- Mécanismes de dessin des vues => *seront traités dans la suite du cours*
- Modularité => *Oui*
- Indépendance Données-Vue-Interaction ? => *Oui*
- Code redondant dans les 2 types de vues => Utilisation d'une classe abstraite intermédiaire.

Commentaires sur cette 2nde réalisation de MVC

- **Réutilisabilité** et **généricité** ? => *Incomplète!*
  - Si l'on veut utiliser un autre modèle ou un autre contrôleur ?

- Par exemple, si l'on veut maintenant:
  - Utiliser un autre modèle de thermomètre avec ce(s) vue(s) et ce contrôleur ?
  - Utiliser une autre vue avec ce modèle et ce contrôleur ?
  - Utiliser un autre contrôleur avec ce modèle et ces vue ?
  - Utiliser une autre vue et un autre contrôleur simultanément avec cette vue, ce modèle et ce contrôleur ? (synchronisation)
  - *Etc.*

- Impossible en l'état car des références entre des objets de chaque module sont typées par leur classe concrète. **On doit encore modifier des parties du code de chaque module.**

Valide par rapport au modèle, mais n'en n'exploite pas le pouvoir d'abstraction (indépendance entre données/représentation/contrôle)

Réalisation de MVC

- Principes des interfaces à étendre à la réalisation du modèle et des contrôleurs:
  - Utiliser une interface `InterfaceModeleThermometre.java` (indépendance de la représentation et du contrôle avec l'implantation du modèle)
  - Idem pour le(s) contrôleur(s) (souvent moins utile car le contrôleur est relativement indépendant du modèle et de la vue)
- Chaque module référence les autres par leur **type apparent (Interface)** et les liens entre modules sont donc indépendants de leurs implantations

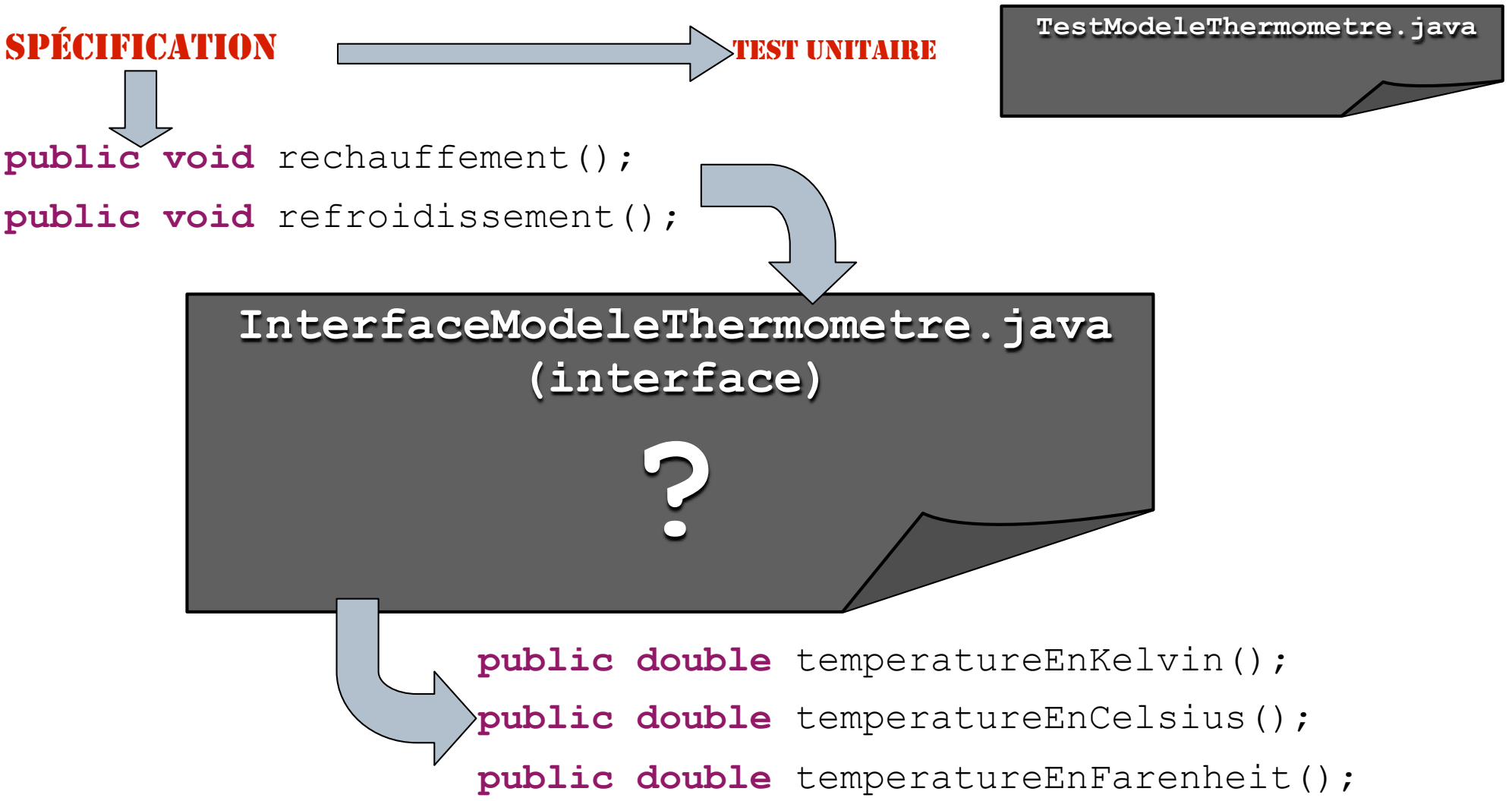
Exemple 3: le thermomètre MVC 'parfait'

- Réaliser une application interactive simulant **2 thermomètres** sur lesquels l'utilisateur peut agir pour contrôler la température
- Les thermomètres devront être de 2 types:
  - Le **thermomètre basique**, précis à 1°C près
  - Le **thermomètre « spatial »**, précis à 0.0001°C près
- L'application fournira:
  - Un affichage de la température courante mesurée par les thermomètre en °C ou en °K ou en F sous la forme:
    - D'un **thermomètre à mercure** pour le 1er thermomètre
    - D'un **cadran à aiguille** pour le 2nd thermomètre
  - Des contrôles permettant à l'utilisateur de diminuer et augmenter la température courante **de chaque** thermomètre
  - Un contrôle permettant de choisir l'unité d'affichage de la température pour **chaque vue**

Thermomètre MVC: analyse

- Le modèle (cf TP1b)
  - Données et traitements réalisés:
    - *Température courante*
    - *Maintient de l'état* de la température courante
    - *Conversions* de la température en différentes unités
  - Fonctionnalités exposées:
    - **Augmenter** la température de 1° (C ou K)
    - **Diminuer** la température de 1° (C ou K)
    - **Donner** la température en °C, °K ou F

Thermomètre MVC: analyse



Thermomètre MVC: analyse

InterfaceModeleThermometre.java

(interface)

```
public void rechauffement();
public void refroidissement();
public double temperatureEnKelvin();
public double temperatureEnCelsius();
public double temperatureEnFarenheit();
```

?

ModeleThermometreBasique.java

```
public void rechauffement();
public void refroidissement();
public double temperatureEnKelvin();
public double temperatureEnCelsius();
public double temperatureEnFarenheit();
```

ModeleThermometreSpatial.java

```
public void rechauffement();
public void refroidissement();
public double temperatureEnKelvin();
public double temperatureEnCelsius();
public double temperatureEnFarenheit();
```

TestModeleThermometre.java

Thermomètre MVC: analyse

- Le contrôleur: identique au précédent mais référence vers `InterfaceModeleThermometre`

- L(es) vue(s): identique(s) au(x) précédente(s) mais référence vers `InterfaceModeleThermometre`

Thermomètre MVC: réalisation

- Le modèle

```
package thermometre.modele;
public interface InterfaceModeleThermometre {
  /**
  * indique la temperature en Kelvin
  * @return la temperature en Kelvin courante que le thermomètre indique
  */
  public double temperatureEnKelvin();
  /**
  * indique la temperature en Celsius
  * @return la temperature en Celsius courante que le thermomètre indique
  */
  public double temperatureEnCelsius();
  /**
  * indique la temperature en Farenheit
  * @return la temperature en Farenheit courante que le thermomètre indique
  */
  public double temperatureEnFarenheit();
  /**
  * permet au capteur d'envoyer l'information d'un rechauffement ambiant d'un degré de la température
  *
  */
  public void rechauffement();
  /**
  * permet au capteur d'envoyer l'information d'un refroidissement ambiant d'un degré de la température
  *
  */
  public void refroidissement();
}
```

Thermomètre MVC: réalisation

- Le modèle thermomètre basique

```
package thermometre.modele;

public class ModeleThermometreBasique implements InterfaceModeleThermometre {
  final static protected double zeroAbsoluCelcius=-273.15;
  private double _temperature;//la temperature en Kelvin


  /**
  * Construit un thermometre avec comme temperature initale 0 degre celcius
  */
  public ModeleThermometreBasique(){
     _temperature=0-zeroAbsoluCelcius;
  }

  public double temperatureEnKelvin(){
     //opérations propres à ce modèle...
  }
  //Implantation des autres méthodes définies dans l'interface...
}
```

Thermomètre MVC: réalisation

- Le modèle thermomètre spatial

```
package thermometre.modele;

public class ModeleThermometreSpatial implements InterfaceModeleThermometre {
  final static protected double zeroAbsoluCelcius=-273.15;
  private double _temperature;//la temperature en Kelvin


  /**
  * Construit un thermometre avec comme temperature initale O degre celcius
  */
  public ModeleThermometreSpatial(){
     _temperature=0-zeroAbsoluCelcius;
  }

  public double temperatureEnKelvin(){
     //opérations propres à ce modèle...
  }
  //Implantation des autres méthodes définies dans l'interface...
}
```

Thermomètre MVC: réalisation

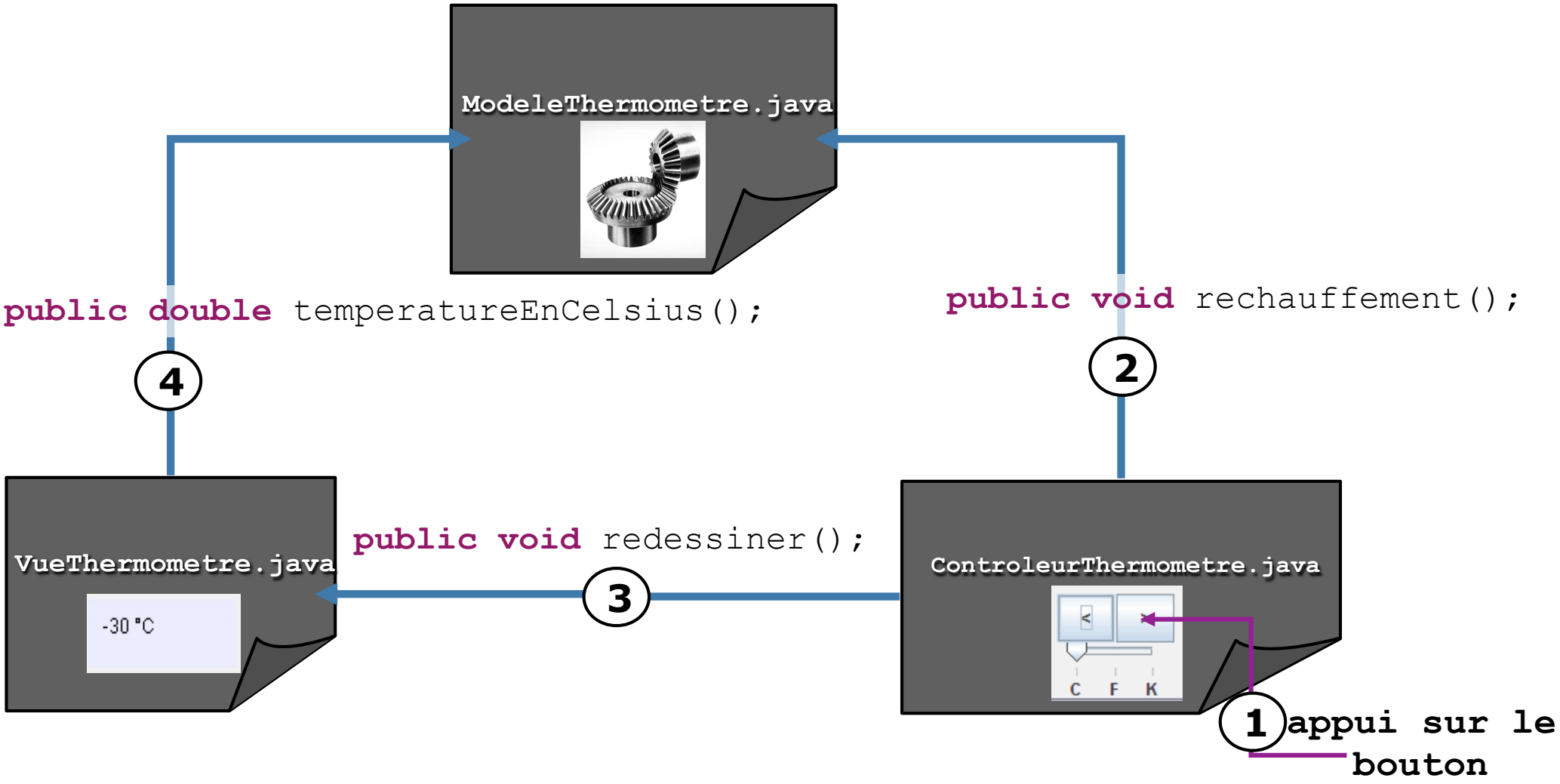
- L'application

```
package thermometre;
import java.awt.GridLayout;
import javax.swing.JFrame;
import thermometre.controleurs.ControleurThermometre;
import thermometre.modele.ModeleThermometre;
import thermometre.vues.*;

public class AppliThermometreSimple {

        public static void main(String[] args) {
                //Creation d'une fenêtre pour l'application
                JFrame frame = new JFrame();

                //Creation des modèles de thermomètre
                InterfaceModeleThermometre modele1 = new ModeleThermometreBasique(243.15);
                InterfaceModeleThermometre modele2 = new ModeleThermometreSpatial();

                //Creation de la vue 'mercure' et de son contrôleur
                InterfaceVueThermometre vue1 = new VueThermometreMercure (modele1);
                ControleurThermometre pt1 = new ControleurThermometre(modele1, vue1);

                //Creation de la vue 'compteur' et de son contrôleur
                InterfaceVueThermometre vue2 = new VueThermometreCompteur (modele2);
                ControleurThermometre pt2 = new ControleurThermometre(modele2, vue2);

                //Ajout des panneaux à la fenêtre et affichage...
                //...
        }
}
```

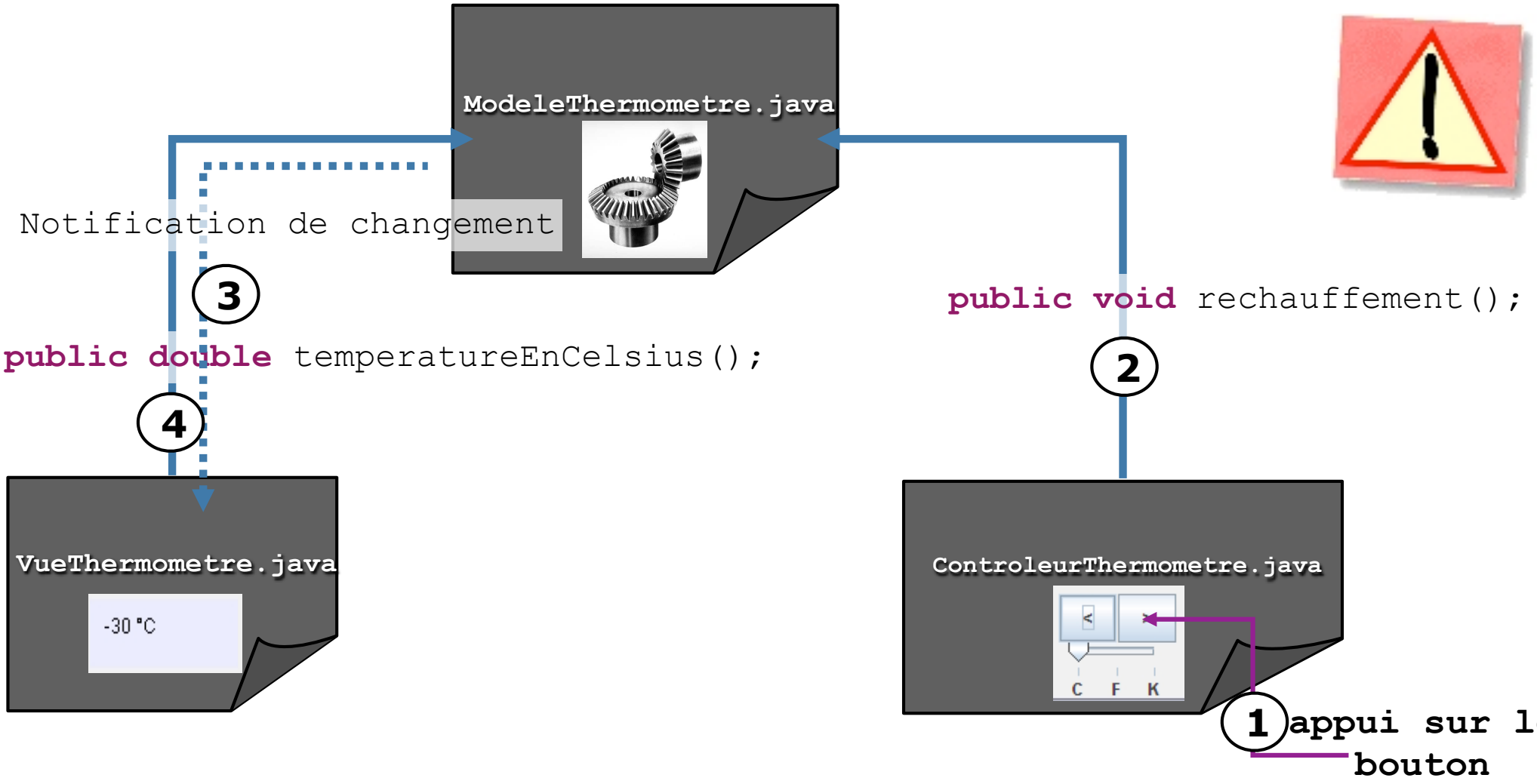
Thermomètre MVC: pour aller plus loin…

- Pour les vues:
  - Permettre à la vue d'être notifiée directement par le modèle de ses mises à jour (observateur)
    - Simplifie la réalisation de MVC
    - Réduit les messages et les réaffichages
    - Permet la synchronisation de plusieurs vues
  - Détails d'implantation dans la suite du cours (*listeners*)

Thermomètre v1: analyse (rappel)



Thermomètre



Ce qu'il faut retenir

- Importance de l'**analyse** du problème en « **pensant MVC** »

- **Abstraire** le schéma de l'application (interfaces et classes abstraites)

-**Indépendance** des modules

MVC: Conclusion et bilan

- Un modèle pour:
  - **Analyser** un « problème »
  - **Structurer** une application interactive
  - **Implanter** un système de manière modulaire, flexible et réutilisable
- Garantit et facilite:
  - L'indépendance **front-office** (IHM) – **back-office** (données et traitements)
  - La **maintenance** et la **réutilisation** de modules
- Mais ce n'est pas une solution universelle...