La modélisation avec UML: introduction aux patrons de conception (suite)

ACDA – CPOO (M3105)

Mathieu Sassolas

IUT de Sénart Fontainebleau
Département Informatique

Année 2015-2016
Cours 5

UPEC UNIVERSITÉ PARIS-EST CRÉTEIL — IUT SENART FONTAINEBLEAU

Plan de la séance

Plan de la séance

La bonne (?) nouvelle

- Le cahier des charges peut être déposé/modifié jusqu'à lundi matin, 8h.
- Ensuite, c'est **terminé** $\rightsquigarrow 0$.
- Donc n'hésitez pas à déposer des versions non finales avant.
- Vous n'êtes pas obligés d'y passer votre week-end, si vous le rendez ce soir c'est très bien aussi.

La mauvaise (?) nouvelle

- Le contrôle aura lieu **mercredi 25 novembre à 8h45**, en salle 140.
- Le programme est : tous les diagramme (re)vus cette année, les scénarios ; s'aider des patrons de conception.
- Les documents sont autorisés ; ils ne sont vraiment utiles que pour les patrons de conception.

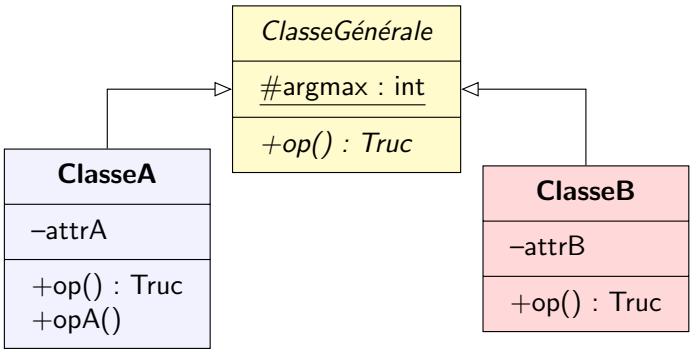
Pas de tiers-temps déclaré ; si vous êtes dans ce cas dépêchez-vous de vous arranger avec l'administration.

Plan de la séance

Plan de la séance

Problème

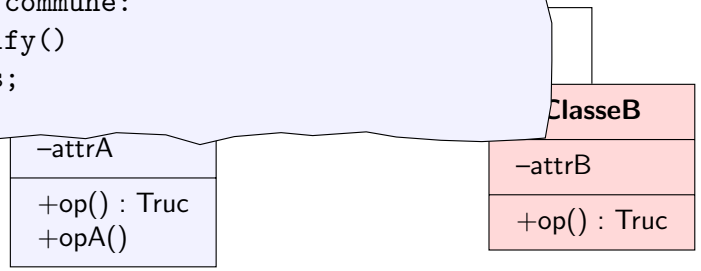
- Une méthode qui a la même structure dans toutes ses versions polymorphes.
- Comment factoriser en permettant des variations ?



```
public Truc op () {
  // Partie spécifique:
  opA();
  attrA.do();
  // Partie commune:
  Truc res = new Truc();
  for (int i=0; i < argmax; i++) {
    // Sous partie spécifique
    res = res.append(attrA.result(argmax-i));
  }
  // Partie commune:
  res.beautify()
  return res;
}
```

...s toutes ses

...riations ?




```
public Truc op () {
  // Partie spécifique:
  opA();
  attrA.do();
  // Partie commune:
  Truc res = new Truc();
  for (int i=0; i
    // Sous parti
    res = res.app
  }
  // Partie commu
  res.beautify()
  return res;
}
```

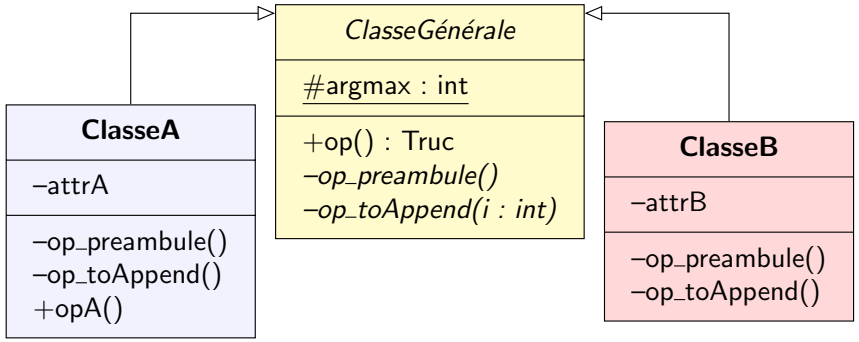
...s toutes ses

```
public Truc op () {
  // Partie spécifique:
  attrB.read();
  // Partie commune:
  Truc res = new Truc();
  for (int i=0; i < argmax; i++) {
    // Sous partie spécifique
    res = res.append(attrB.getValue(i+1));
  }
  // Partie commune:
  res.beautify()
  return res;
}
```


Solution

Idée

- La partie commune de l'opération est dans la classe mère.
- Seules les parties spécifiques seront implémentées (par polymorphisme) dans les classes filles.

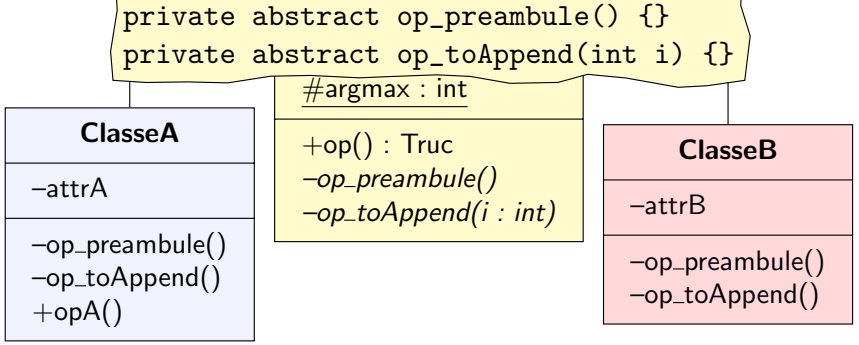


Solutio

Idée

- La ...mère.
- Se... par
  po...

```
public Truc op () {
  op_preambule();
  Truc res = new Truc();
  for (int i=0; i < argmax; i++) {
    res = res.append(toAppend(i));
  }
  res.beautify()
  return res;
}
private abstract op_preambule() {}
private abstract op_toAppend(int i) {}
```



Solutio

Idée

- La
- Se

po

mère.

ar

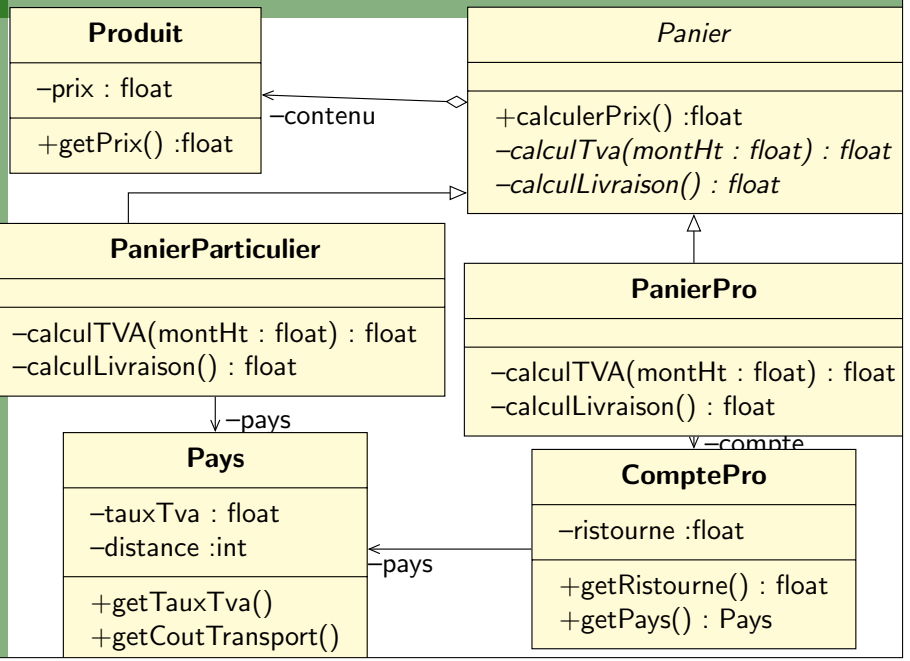
```
public Truc op () {
  op_preambule();
  Truc res = new Truc();
  for (int i=0; i < argmax; i++) {
    res = res.append(toAppend(i));
  }
  res.beautify()
  return res;
}
private abstract op_preambule() {}
private abstract op_toAppend(int i) {}
```

```
private op_preambule () {
  opA();
  attrA.do();
}
private op_toAppend(int i) {
  return attrA.result(argmax-i);
}
```

```
private op_preambule () {
  attrB.read();
}
private op_toAppend(int i) {
  return attrB.getValue(i+1));
}
```

#argmax : int

Exemple



Exemple

```
public abstract class Panier {
  ...
  public float calculerPrix() {
    float res = 0.;
    for (Produit prod: contenu) {
      res = res + prod.getPrix();
    };
    res = res + calculTva(res);
    res = res + calculLivraison();
    return res;
  }
  private abstract float calculTva(float montHt) {}
  private abstract float calculLivraison() {}
  ...
}
```

–pays

+getTauxTva()
+getCoutTransport()

+getRistourne() : float
+getPays() : Pays

Exemple

```
public class PanierParticulier extends Panier {
  ...
  private float calculTva(float montHt) {
    return (montHt * pays.getTauxTva());
  }
  privatefloat calculLivraison() {
    return pays.getCoutTransport();
  }
```

PanierPro

```
public class PanierPro extends Panier {
  ...
  private float calculTva(float montHt) {
    return 0.;
  }
  private float calculLivraison() {
    return (compte.getPays().getCoutTransport()
            * compte.getRistourne());
  }
```

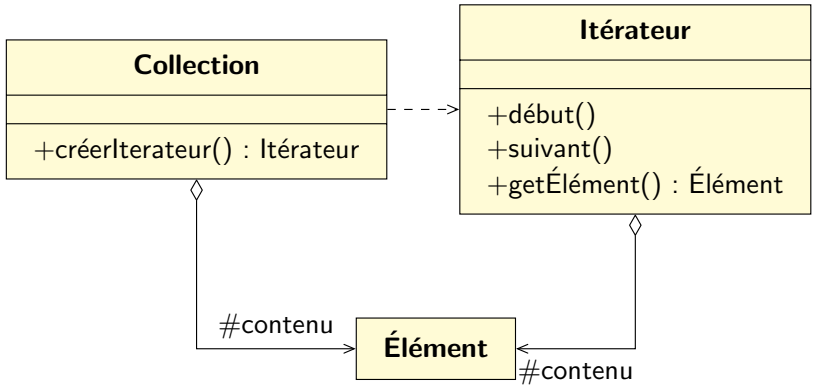
Plan de la séance

Problème

- Parcourir une collection en abstrayant la structure de donnée sous-jacente.
- Pouvoir faire plusieurs parcours en même temps ; par exemple : chaque instance d'un client parcourt les données.

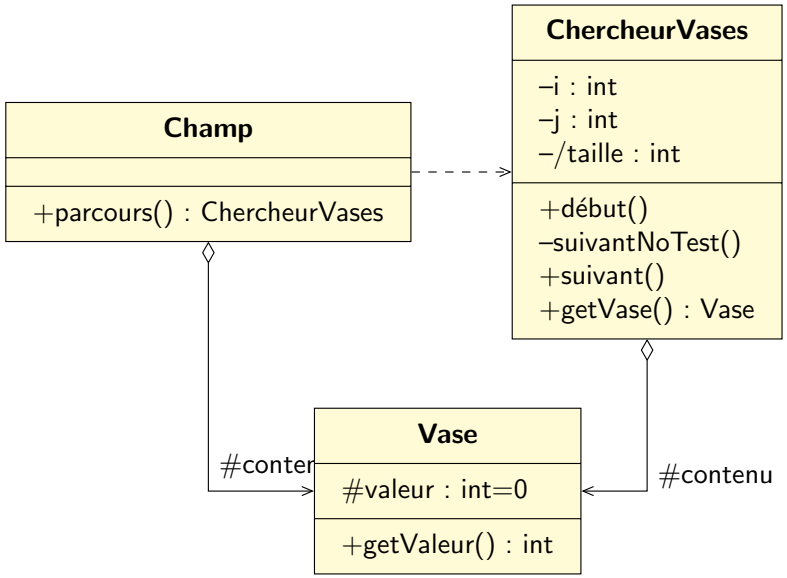
Idée de la solution

Version non générique

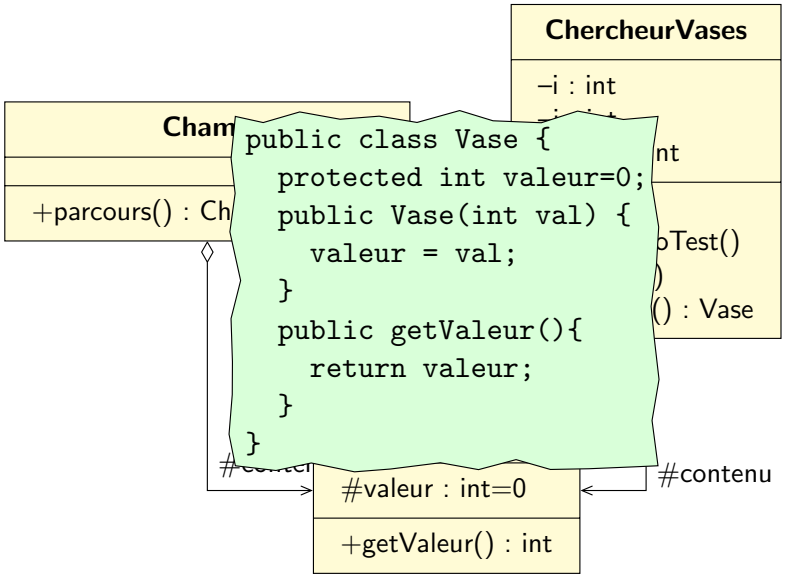


- C'est l'opération `suivant` qui fait tout le travail.
- L'itérateur peut en même temps filtrer les éléments à renvoyer.

Exemple : parcours d'un champ



Exemple : parcours d'un champ




```
public class Vase {
  protected int valeur=0;
  public Vase(int val) {
    valeur = val;
  }
  public getValeur(){
    return valeur;
  }
}
```


Exemple : parcours d'un champ


```
public class Champ {
  protected Vase[][] contenu;
  public Champ(int taille){
    contenu = new Vase[taille][taille];
    for (int i=0; i < taille; i++) {
      for (int j=0; j < taille; j++) {
        if (Random.bool()) {
          contenu[i][j] = new Vase(0);
        } else {
          contenu[i][j] = new Vase(Random.int(taille));
        }}}
  };
  public ChercheurVases parcours() {
    return new ChercheurVases(contenu);
  }
}
```

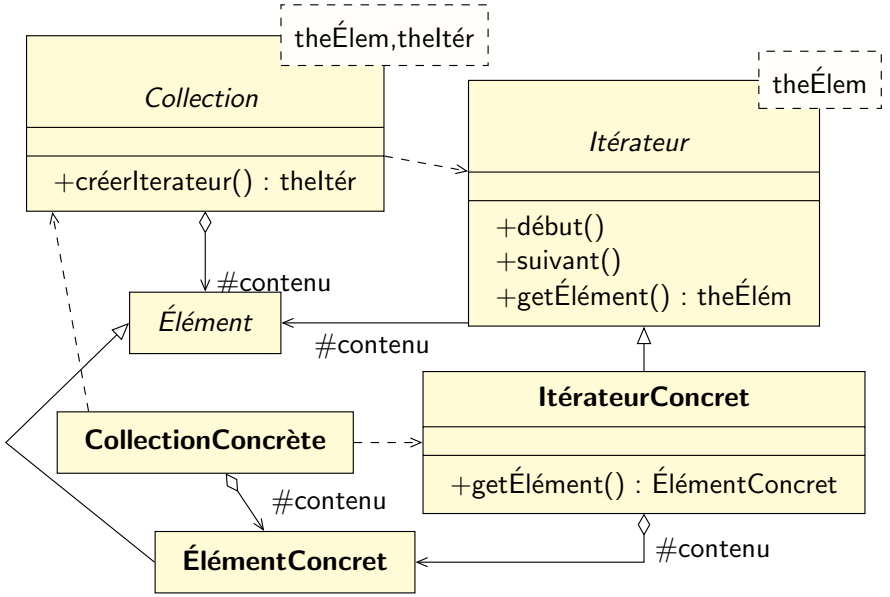

Exemple : parcours d'un champ


```
public class ChercheurVases {
  private int i=0;
  private int j=0;
  private int taille;
  protected Vase[][] contenu;
  public ChercheurVases(Vase[][] champ) {
    contenu = champ;
    taille = length(contenu);
  }
  private suivNoTest() {
    if (j == taille - 1) {
      i = i+1;
      j=0;
    } else {
      j = j+1;
  }
```

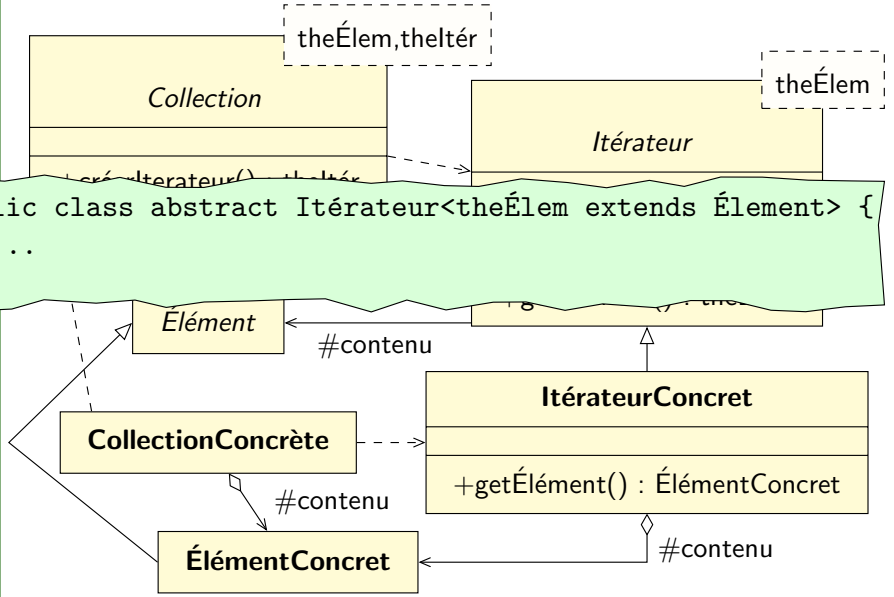

Exemple : parcours d'un champ

```
  public Vase getVase() {
    if (i < taille && j < taille) {
      return contenu[i][j];
    } else {
      return null;
    }
  }
  public début() {
    i=0;
    j=0;
    suivant();
  }
  public suivant() {
    while (getVase() != null && getVase().getValeur() == 0) {
      suivNoTest();
    };
  }
}
```

Idée de la solution
Version générique



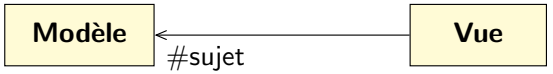
Idée de la solution
Version générique



Plan de la séance

Problème

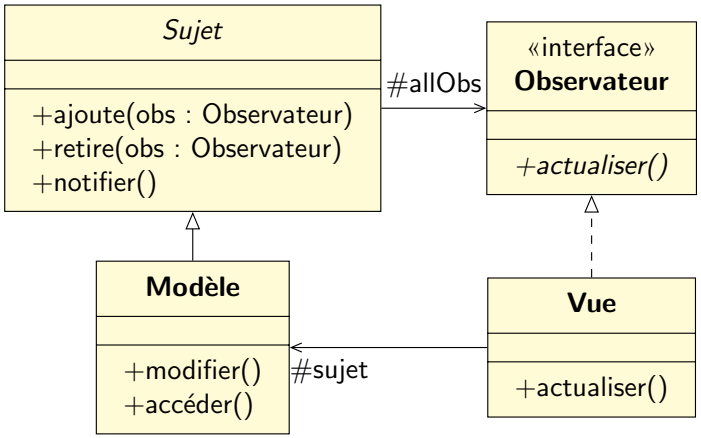
Propager les modifications du modèle vers les modifications des vues.



Solution

Idée

- Le modèle maintient une liste de ses observateurs.
- À chaque modification de son état ils sont notifiés.



IUT SENART FONTAINEBLEAU

Solution

Design Patterns II
M. Sassolas
M3105
Cours 5
Annonces
Comportement
Patron de méthode
Itérateur
Observateur
Étude de cas

Idée

- Le modèle maintient une liste de ses observateurs.
- À chaque modification de son état ils sont notifiés.

```
public class Modèle extends Sujet {
  ...
  public modifier() {
    ...
    notifier();
  }
  ...
}
```

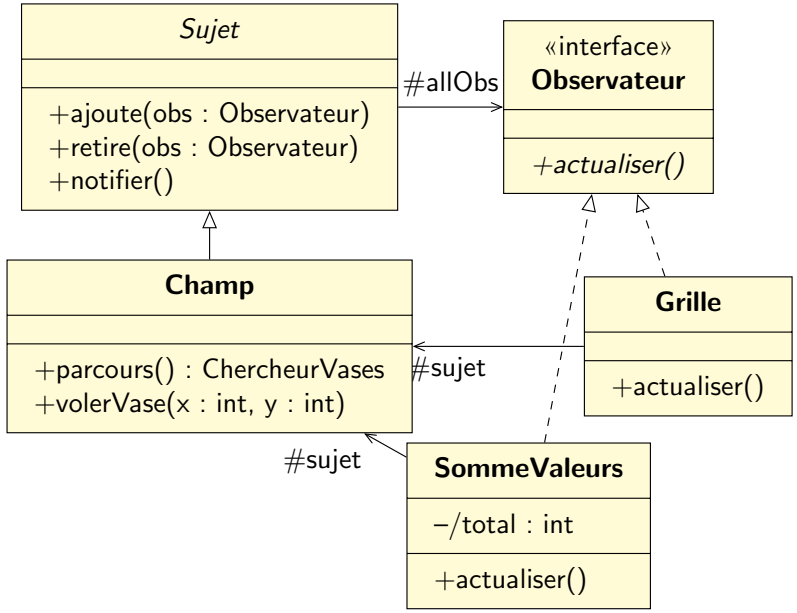
```
public class Sujet {
  ...
  public notifier() {
    for (Observateur o: allObs) {
      o.actualiser();
    };
  }
  ...
```

«interface»
Observateur

IUT SENART FONTAINEBLEAU

Exemple

Design Patterns II
M. Sassolas
M3105
Cours 5
Annonces
Comportement
Patron de méthode
Itérateur
Observateur
Étude de cas



IUT SENART FONTAINEBLEAU

Exemple

```
public class Champ extends Sujet{
  ...
  public VolerVase(int x, int y) {
    if (x < length(contenu) && y < length(contenu)) {
      contenu[x][y] = 0;
      notifier();
    };
  ...
  }
```

+parcours() : ChercheurVases
+volerVase(x : int, y : int)
#sujet
+actualiser()
#sujet
SommeValeurs
−/total : int
+actualiser()

Exemple

```
public class Champ extends Sujet{
  ...
  publ
    if
    };
  ...
  }
```

```
public class SommeValeurs implements Observateur {
  ...
  public actualiser() {
    ChercheurVases champIter = sujet.parcours();
    int res = 0;
    champIter.début():
    while (champIter.getVase() != null) {
      res = res + champIter.getVase().getValeur();
      champIter.suivant();
    };
    total = res;
  }
}
```

+actualiser()

Exemple

```
public class Champ extends Sujet{
  ...
  publ
    if
    };
  ...
  }
```

```
public class SommeValeurs implements Observateur {
  ...
  public actualiser() {
    ChercheurVases champIter = sujet.parcours();
    int res = 0;
    champIter.début():
    while (champIter.getVase() != null) {
      re...
      ch
    };
    tota
  }
}
```

```
Champ field = new Champ(42);
Grille grid = new Grille(field);
field.ajoute(grid);
SommeValeurs tot = new SommeValeurs(field);
field.ajoute(tot);
...
```


Plan de la séance

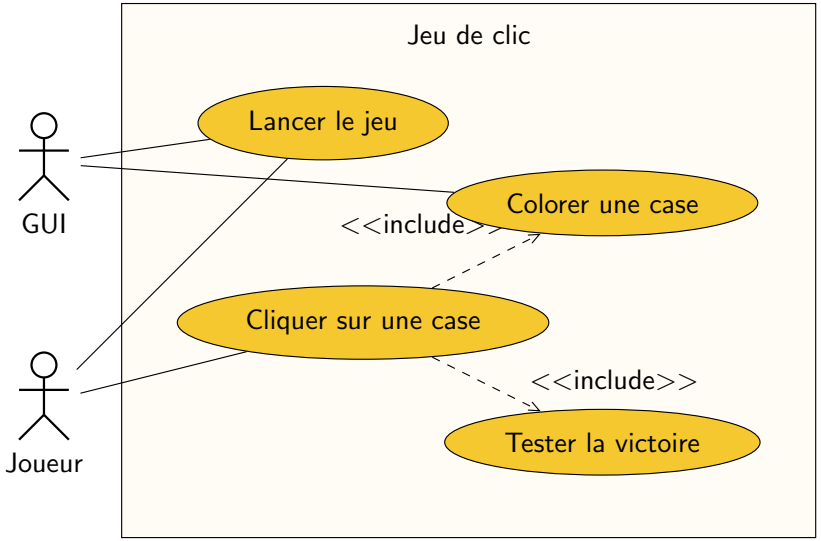
Plan de la séance

Un jeu de clics

On souhaite implémenter un petit jeu qui fonctionne de la manière suivante. Le jeu est lancé en ligne de commande avec deux nombres n et p en argument. Le système crée alors via un système d'interface graphique un carré de $n \times n$ fenêtres remplies d'une couleur RGB aléatoire. Lorsque l'utilisateur clique sur une case, celle-ci change de couleur. Le jeu se termine lorsque toutes les cases ont la même couleur. Pour faciliter la chose, on n'utilisera qu'un nombre p de couleurs qu'on choisira le plus éloignés possible dans la roue chromatique.

Plan de la séance

Cas d'utilisation



Plan de la séance

Scénario nominal d'une partie

Descriptif

1. Le joueur lance la partie.
2. Le GUI affiche les cases.
3. Le joueur clique sur une case.
4. Le GUI change la couleur de la case.
5. La condition de victoire est atteinte.
6. Le jeu se termine.

Flux alternatif :

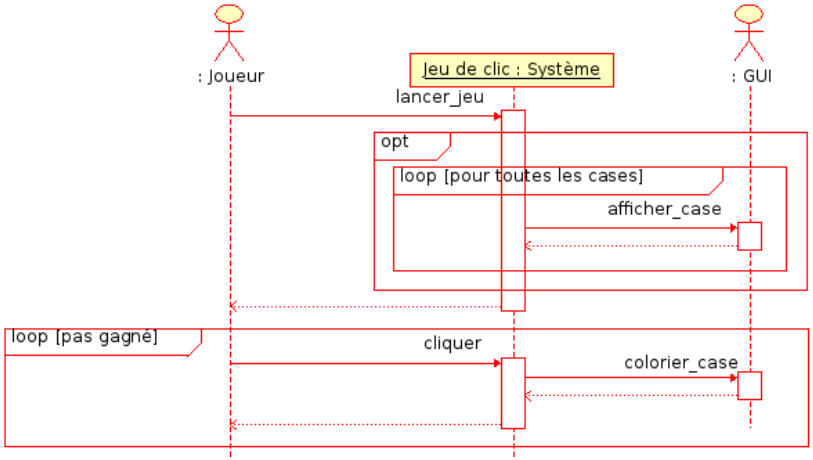
5. La condition de victoire n'est pas atteinte. Le scénario continue à l'étape 3.

Scénarios d'erreur

- Les paramètres donnés en entrée ne sont pas corrects.
- Le système ne dispose pas d'interface graphique.

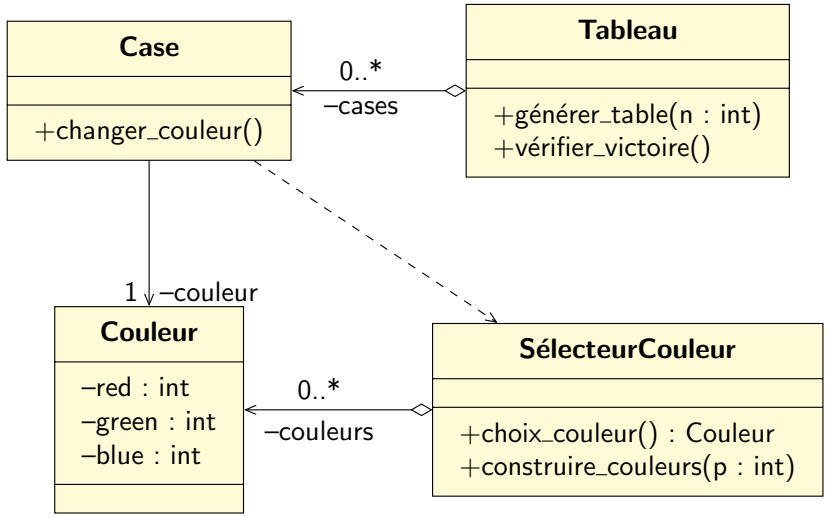
Plan de la séance

Une partie



Plan de la séance

Diagramme de classes « haut niveau »



Remarques

- Pas de classe `Système` ou `Jeu` car tout est le système.
- Pas de classe `Joueur` ni `GUI`.
- On a une classe couleur car on donne un rôle particulier.
- On ne se préoccupe pas (encore) des clics ni de l'affichage.

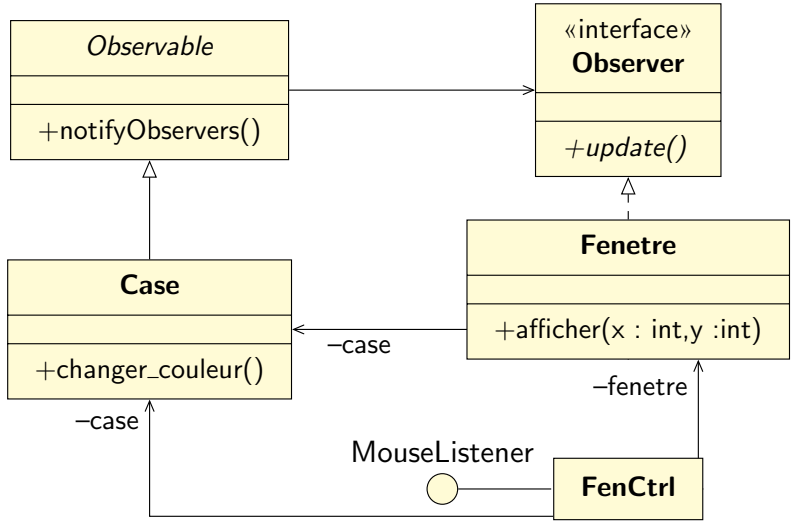
Plan de la séance

Liste des problèmes à gérer

- On doit afficher des cases : vue graphique
- On doit pouvoir cliquer sur une case, c'est à dire qu'un clic doit être détecté (par un contrôleur).
- Lorsque l'on modifie la couleur, les vues doivent être averties.
- Le tableau doit également être averti d'une modification des cases car il doit tester la victoire (en un sens, c'est également une vue).
- Il n'y aura qu'une seule instance de la classe `Tableau`.
- Il n'y aura qu'une seule instance de la classe `SélecteurCouleur` qui recense toute les couleurs définies globalement.

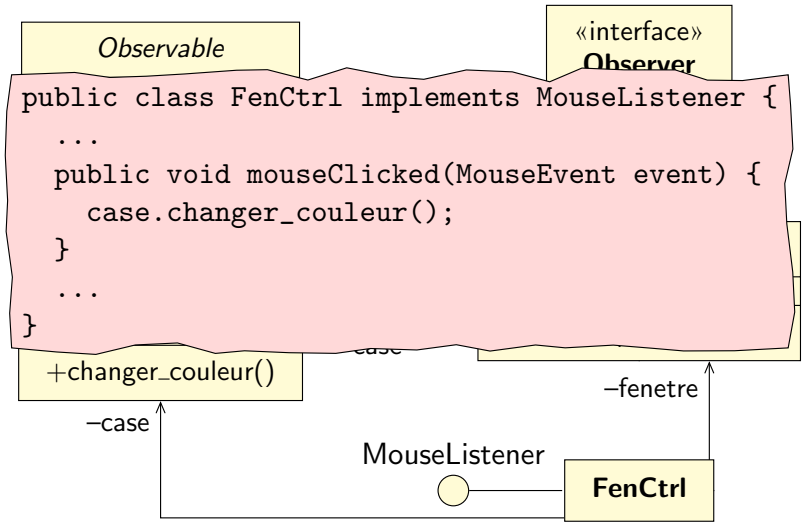
On crée les diagramme de séquences en même temps que les diagrammes de classes de conception.

Les cases

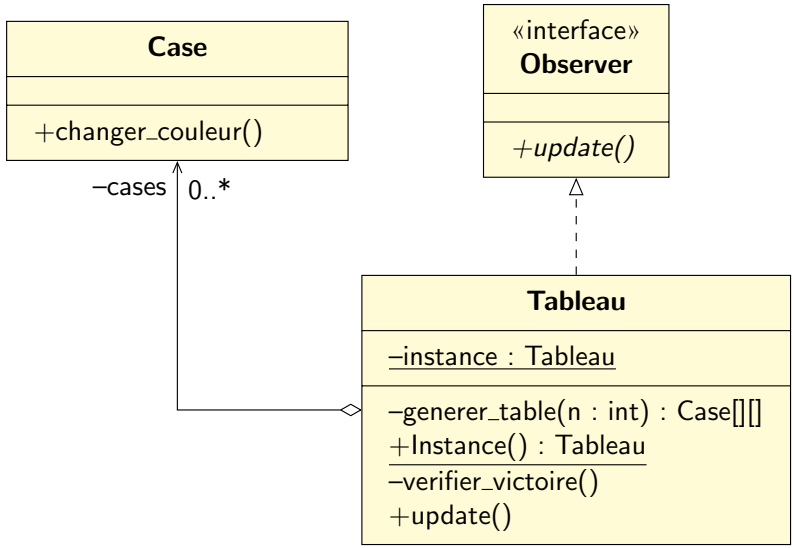


Les cases

```
public class FenCtrl implements MouseListener {
  ...
  public void mouseClicked(MouseEvent event) {
    case.changer_couleur();
  }
  ...
}
```

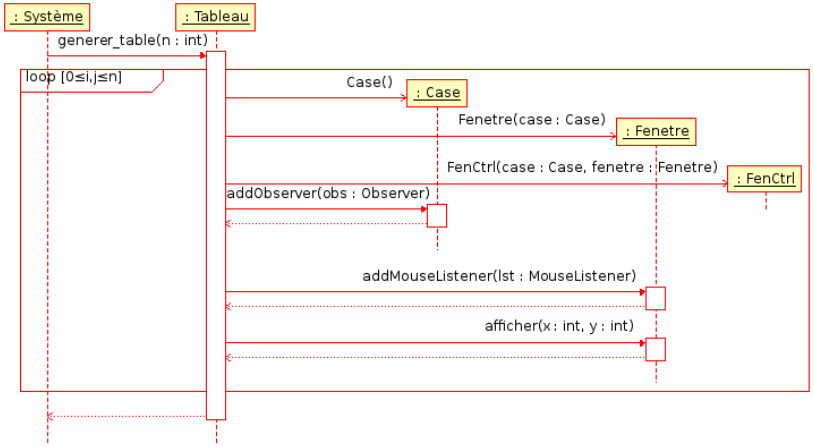


Le tableau



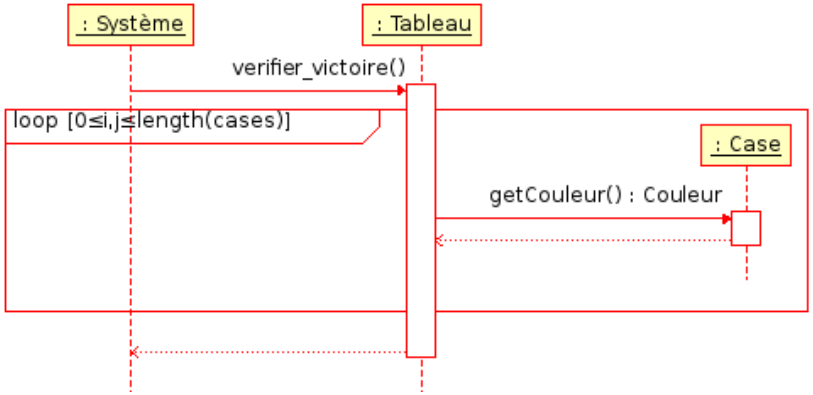
```
public class Tableau implements Observer {
  private Case[][] cases;
  private static Tableau instance = null;
  private Tableau (int n) {
    cases = generer_table(n);
  }
  public Tableau Instance(int n){
    if (instance == null) {
      instance = new Tableau(n);
    };
    return instance;
  }
  public static Tableau Instance() {
    if (instance == null) {
      instance = new Tableau(0);
    };
    return instance;
  }
  public update() {verifier_victoire()} // Adaptateur
  ...
```


Génération du tableau

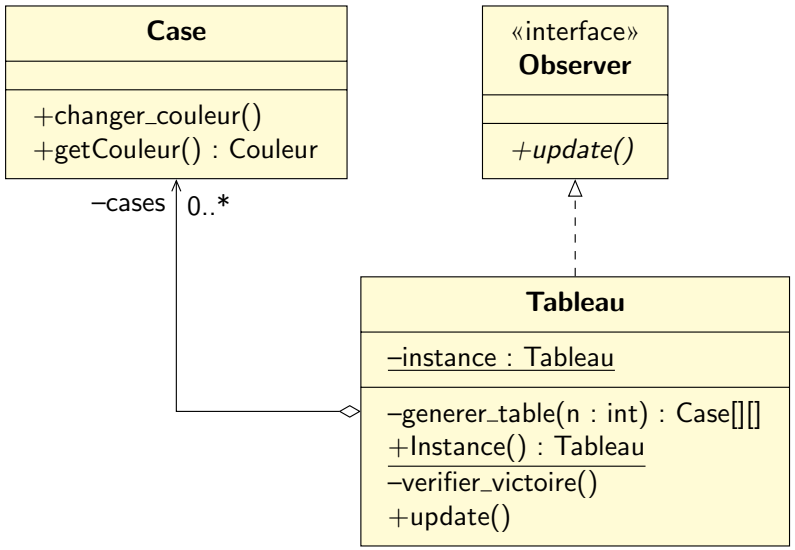



```
  ...
  private static Case[][] generer_table(int n) {
    Case[][] res = new Case[n][n];
    for (int i=0; i < n; i++) {
      for (int j=0; j < n; j++) {
        cases[i][j] = new Case(); // Modèle
        fenij = new Fenetre(cases[i][j]); // Vue
        ctrlij = new FenCtrl(cases[i][j],fenij);
          // Contrôleur
        cases[i][j].addObserver(fenij);
        fenij.addMouseListener(ctrlij);
        fenij.afficher(i,j);
      };
    };
    return res;
  }
  private verifier_victoire() {
    // Test de l'égalité de toutes les couleurs
    // des cases du tableau. À implémenter.
}
```

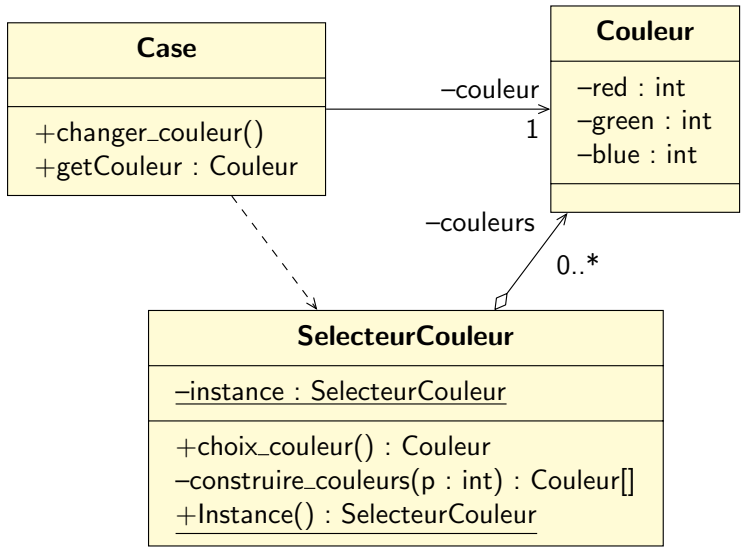

Test de la victoire



Le tableau



Couleurs



Couleurs

Case

Couleur

```
public class Case extends Observable {
  private Couleur;
  public Case() {
    couleur = SelecteurCouleur.Instance().choix_couleur();
  }
  public changerCouleur() {
    couleur = SelecteurCouleur.Instance().choix_couleur();
    notifyObservers();
  }
}
```

–construire_couleurs(p : int) : Couleur[]
+Instance() : SelecteurCouleur

Remarques

- On a découpé le système en plusieurs diagrammes.
- Tout n'est pas fini, il manque :
  - L'affichage proprement dit (fonction `afficher`).
  - La fonction `construire_couleur`.
  - Le `SelecteurCouleur`, qui est un singleton.
  - Le `main` qui s'interface avec la ligne de commande.
- Extension possible : lorsqu'on clique sur une case, on affiche le nombre de case de la même couleur via un observateur commun à toutes les cases qui fait appel à un itérateur sur le tableau.

Plus d'études de cas de ce genre avec Luc Hernandez dans la suite de CPOO.