M1 info

GINF41B2 (Conception et Programmation Orientée Objet)

Cours #9
Quelques éléments sur la programmation événementielle

Pierre Tchounikine

Plan

- Le schéma observeur / observé

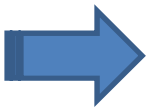
  (à travers un exemple)

- Lien avec les interfaces graphiques (Listener)

Idée générale

- Problématique : maintenir des objets synchronisés
  - sans que l'objet intéressé ait besoin d'aller regarder toutes les x milli-secondes si l'objet auquel il s'intéresse a changé
  - sans que l'objet qui évolue ait besoin de savoir qui est intéressé par son état et/ou ses changements d'état
- Approche : une autre façon de communiquer

des objets qui envoient des messages à d'autres objets



des objets qui s'intéressent aux changements de certains objets

+

des objets qui signalent qu'ils ont changé d'état

objets synchronisés : données / données, représentations / données, etc.

Exemple

Exemple

des objets qui s'intéressent aux changements de certains objets

des objets qui signalent qu'ils ont changé d'état

quelque chose qui fait le lien

Les hommes

```
import java.util.*;
public class Homme implements Runnable {
String nom;

Homme (String nom){this.nom = nom;}

public void drague(Femme f){
System.out.println("Hello " + f.getNom() + " je m'appelle "+
nom + " vous habitez chez vos parents ?");    }

public void run(){
while (true)
{
System.out.println("je bois une bière");
{ try {Thread.sleep(7000);}
  catch (InterruptedException e) {}}
}
}
}
```

activité de fond

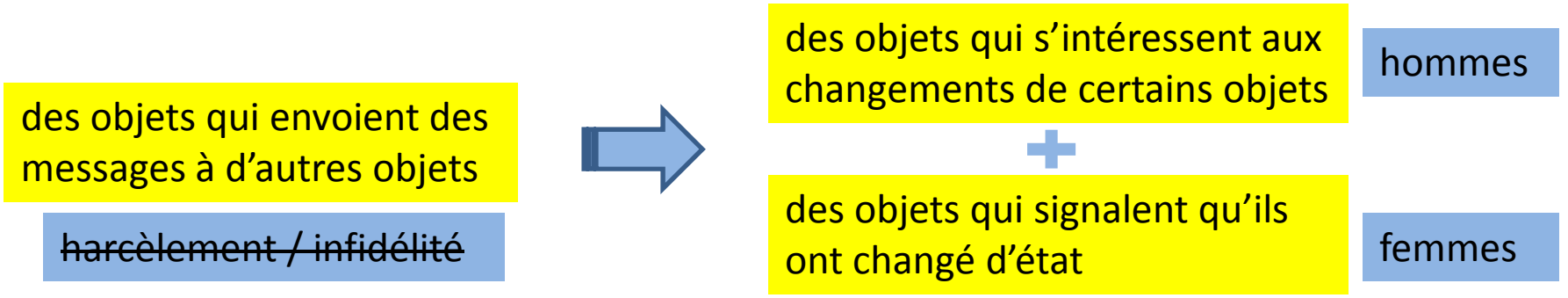
Les femmes

```
import java.util.*;
import java.util.Random;
public class Femme implements Runnable {
private String nom;
private Boolean amoureuse = false;

Femme (String nom){this.nom = nom;}
String getNom(){return nom;}
Boolean amoureuse (){return amoureuse==true;}

void tombeAmoureuse (){amoureuse=true;}
void rompt (){amoureuse=false;}

public void run(){
while (true) {
int a = new Random().nextInt(100);
if (a>95) { if (amoureuse()) {rompt();}
            else             {tombeAmoureuse();}}}
}}
```

activité de fond

Idée générale

- Problématique : maintenir des objets synchronisés
  - sans que l'objet intéressé ait besoin d'aller regarder toutes les x milli-secondes si l'objet auquel il s'intéresse a changé
  - sans que l'objet qui évolue ait besoin de savoir qui est intéressé par son état et/ou ses changements d'état
- Approche : une autre façon de communiquer



objets synchronisés : données / données, représentations / données, etc.

les hommes draguent les femmes qui ne sont pas (déjà) amoureuses

Modélisation générale

des objets qui s'intéressent aux changements de certains objets

« observateurs »

des objets qui signalent qu'ils ont changé d'état

« observables »

quelque chose qui fait le lien

« qui observe qui »

Outils Java : la classe observable

Overview Package **Class** Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS

SUMMARY: NESTED | FIELD | CONSTR | METHOD

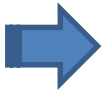
java.util

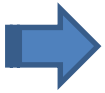
Class Observable

java.lang.Object
  └ java.util.Observable

public class Observable
extends Object

Outils Java : la classe observable





Method Summary	
void	addObserver(Observer o) Adds an observer to the set of observers for this object, provided that it is not the same as some observer already in the set.
protected void	clearChanged() Indicates that this object has no longer changed, or that it has already notified all of its observers of its most recent change, so that the hasChanged method will now return false.
int	countObservers() Returns the number of observers of this Observable object.
void	deleteObserver(Observer o) Deletes an observer from the set of observers of this object.
void	deleteObservers() Clears the observer list so that this object no longer has any observers.
boolean	hasChanged() Tests if this object has changed.
void	notifyObservers() If this object has changed, as indicated by the hasChanged method, then notify all of its observers and then call the clearChanged method to indicate that this object has no longer changed.
void	notifyObservers(Object arg) If this object has changed, as indicated by the hasChanged method, then notify all of its observers and then call the clearChanged method to indicate that this object has no longer changed.
protected void	setChanged() Marks this Observable object as having been changed; the hasChanged method will now return true.

Les femmes

```
public class Femme extends Observable implements Runnable {
...

void tombeAmoureuse (){
amoureuse=true;
System.out.println(nom + " : je deviens amoureuse");
setChanged();
notifyObservers();
}

void rompt (){
amoureuse=false;
System.out.println(nom + " : je rompt");
setChanged();
notifyObservers();
}
```

Outils Java : l'interface observer

java.util

Interface Observer

`public interface Observer`

A class can implement the `Observer` interface when it wants to be informed of changes in observable objects.

Since:

JDK1.0

See Also:

`Observable`

Method Summary	
void	update(Observable o, Object arg) This method is called whenever the observed object is changed.

Les hommes

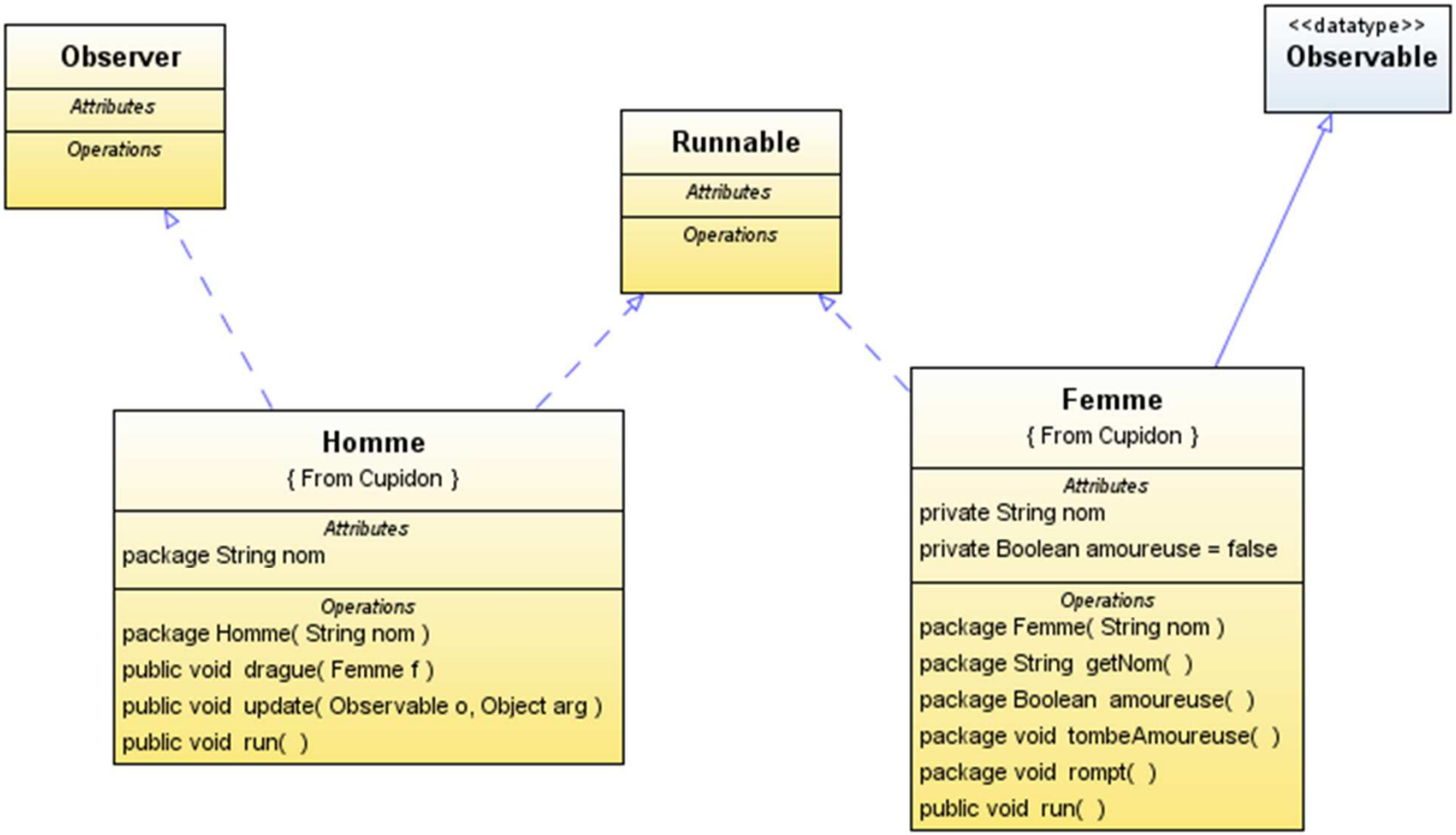
```
public class Homme implements Observer,Runnable {
...

public void drague(Femme f){
System.out.println("Hello " + f.getNom() + " je m'appelle "+
nom + " vous habitez chez vos parents ?");    }

public void update(Observable o, Object arg){
Femme f = (Femme)o;
if (!f.amoureuse()) {drague(f);}
}

public void run(){  }   // inchangé
}
```

Cupidon fait le lien

```
public class Main {
    public static void main(String[] args) {
      Femme f1 = new Femme("Julie");
      Femme f2 = new Femme("Isabelle");
      Homme h1 = new Homme ("Paul");
      Homme h2 = new Homme ("Mathieu");
      Thread tf1 = new Thread (f1);
      Thread tf2 = new Thread (f2);
      Thread th1 = new Thread (h1);
      Thread th2 = new Thread (h2);
      f1.addObserver (h1);
      f2.addObserver (h1);
      f2.addObserver (h2);
      tf1.start();
      tf2.start();
      th1.start();
      th2.start(); }

}
```

cf. code et démo

Cupidon fait le lien



Pour aller plus loin …

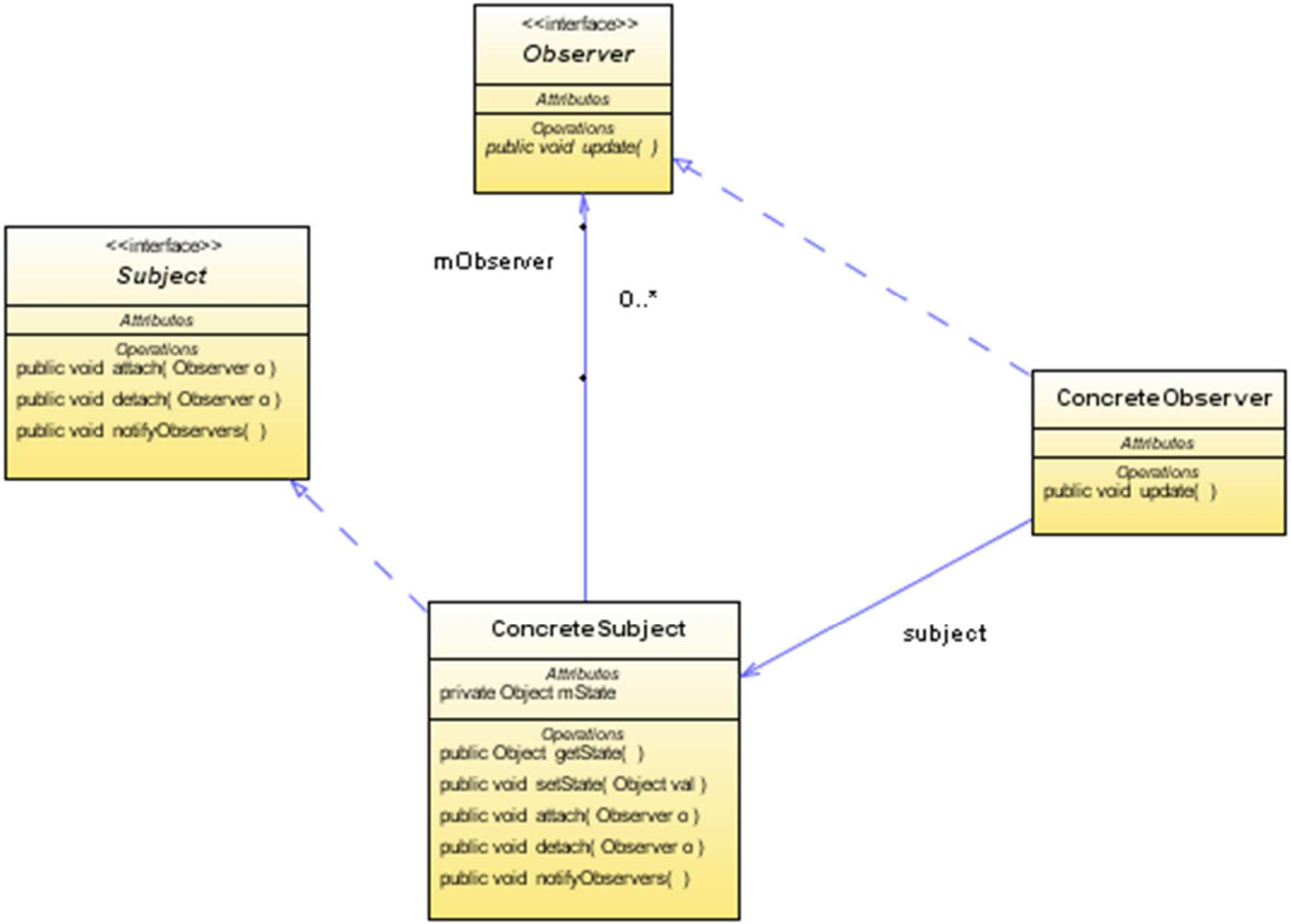
Le Pattern Observer

avertir différents objets d'un changement d'état

Observer	A one-to-many dependency is defined between objects. When one object changes state, all its dependents are automatically notified and updated.	Subject	Defines an abstract interface used to attach various Observers to this Subject.
		Observer	Defines an abstract interface used when a Subject the Observer is observing is updating.
		ConcreteSubject	Implements the operations declared in the Subject interface.
		ConcreteObserver	Implements the operations declared in the Observer interface.

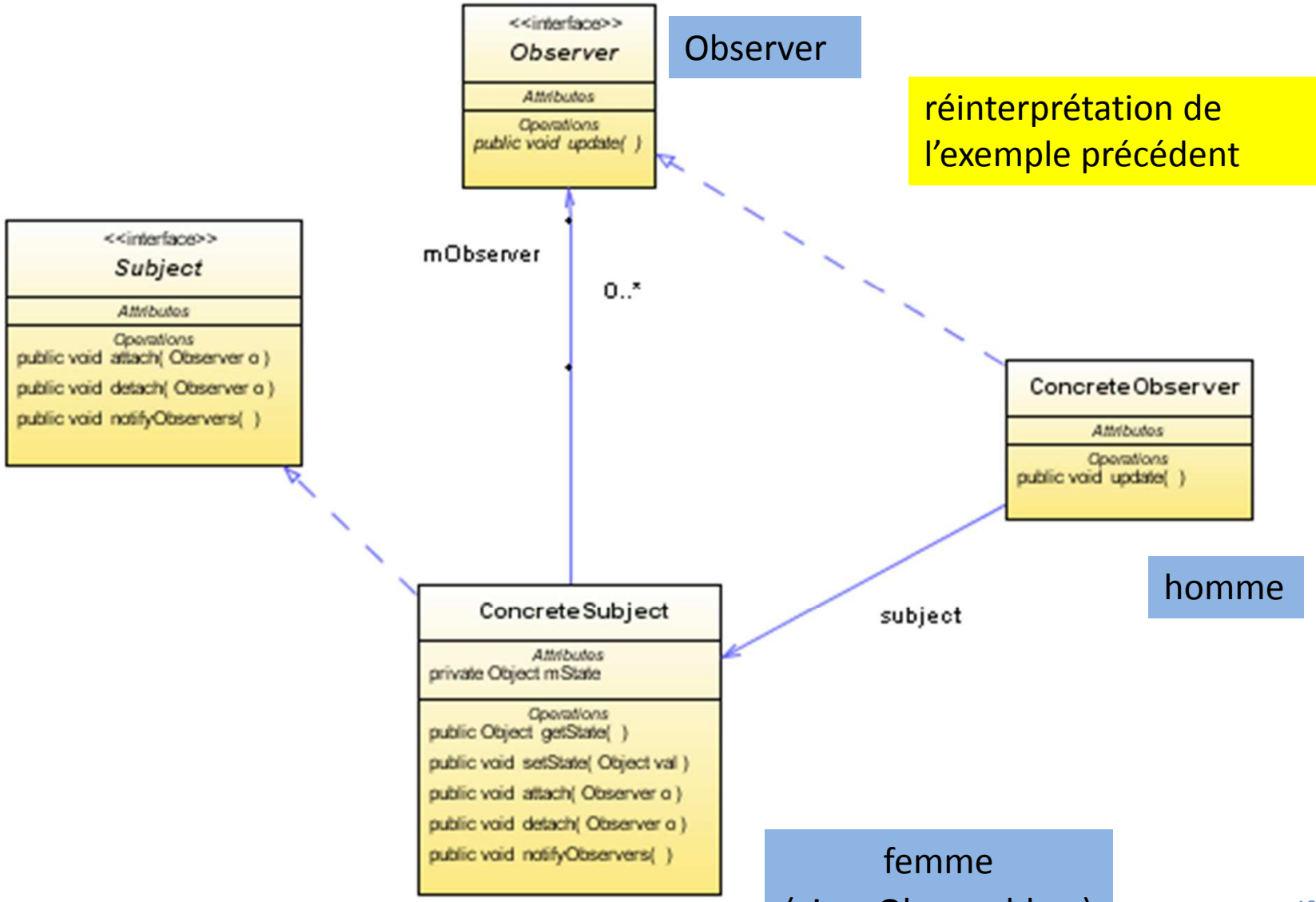
Pour aller plus loin …

Le Pattern Observer

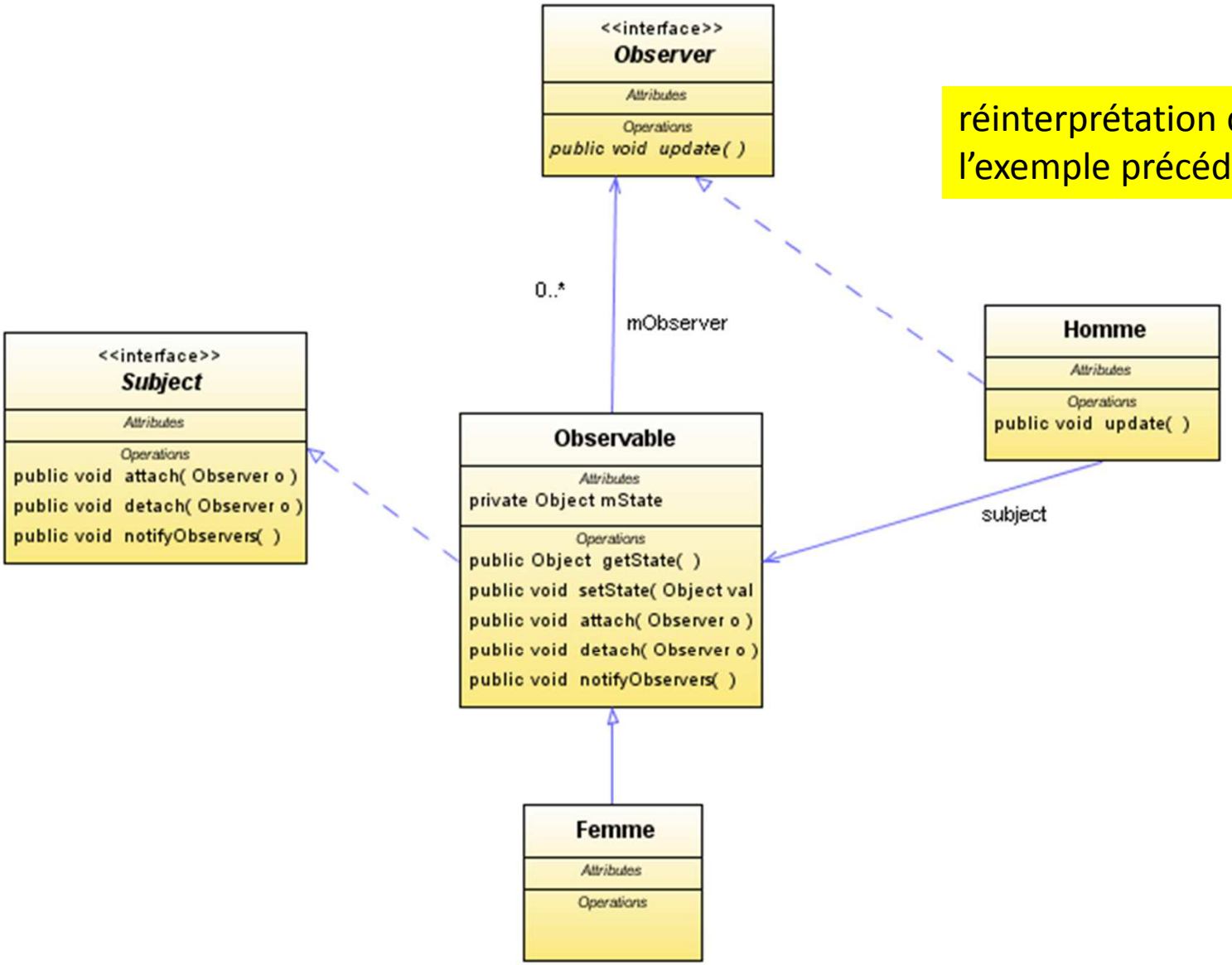


Pour aller plus loin …

Le Pattern Observer



Le Pattern Observer



Résumé

- des observés / des observateurs
  - 1 observé / n observateurs
  - n observés / 1 observateur
- notification observés → observateurs (<<événement>>)
- les observés ne connaissent rien des observateurs
- possibilités d'ajouter / retirer des observateurs dynamiquement

Lien avec les interfaces graphiques (Listener)

Une fenêtre

```
import javax.swing.*;
public class Fenêtre extends JFrame{
Fenêtre(int x, int y, int dimx, int dimy, String nom){
    this.setBounds(x,y,dimx,dimy);
    this.setTitle(nom);
    this.setVisible(true);
}
}
```

```
Fenêtre f = new Fenêtre(150,300,600,400,"Cliques'y si tu l'oses");
```

Fenêtre = interface

- Il se passe des choses dans la fenêtre
  - entrée dans la fenêtre
  - click souris
  - …
- Pour savoir ce qu'il se passe, il faut « écouter » la fenêtre
- Deux façons de faire :
  - « Listener »
  - « Adapter »

MouseListener

```
public interface MouseListener
extends EventListener
```

The listener interface for receiving "interesting" mouse events (press, release, click, enter, and exit) on a component. (To track mouse moves and mouse drags, use the `MouseMotionListener`.)

The class that is interested in processing a mouse event either implements this interface (and all the methods it contains) or extends the abstract `MouseAdapter` class (overriding only the methods of interest).

Method Summary	
void	**mouseClicked**(MouseEvent e) Invoked when the mouse button has been clicked (pressed and released) on a component.
void	**mouseEntered**(MouseEvent e) Invoked when the mouse enters a component.
void	**mouseExited**(MouseEvent e) Invoked when the mouse exits a component.
void	**mousePressed**(MouseEvent e) Invoked when a mouse button has been pressed on a component.
void	**mouseReleased**(MouseEvent e) Invoked when a mouse button has been released on a component.

Une implémentation de MouseListener

```
import javax.swing.*;
import java.awt.event.*;
public class EcouteurSouris implements MouseListener {

    public void mouseClicked (MouseEvent e)
                              {System.out.println("Ouille");}
    public void mousePressed (MouseEvent e) {  }
    public void mouseReleased (MouseEvent e) {  }
    public void mouseEntered (MouseEvent e) {  }
    public void mouseExited (MouseEvent e) {  }
}
```

Une utilisation de MouseAdapter

```
import javax.swing.*;
import java.awt.event.*;
public class AdapteurSouris extends MouseAdapter{
// on redéfinit juste les méthodes utiles

    public void mouseClicked (MouseEvent e)
    {System.out.println("Aie, touché au point " + e.getX()
                        + " " +  + e.getY());
    }
}
```

Lien fenêtre / écouteurs

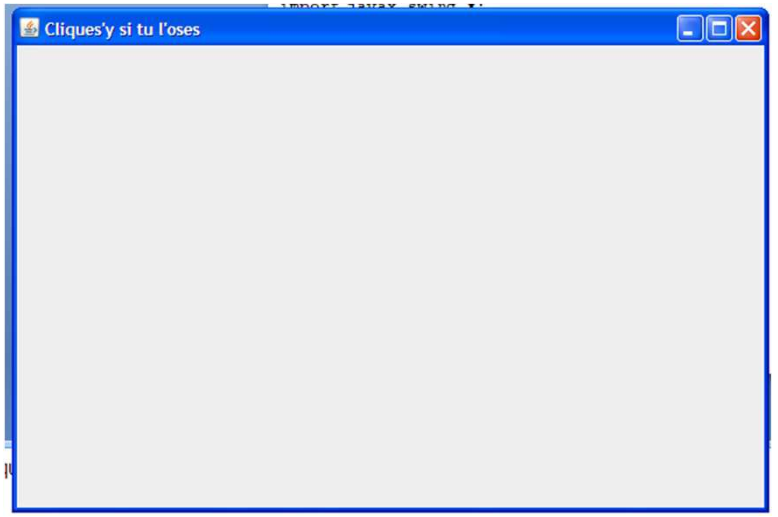
```
public class Main {
    public static void main(String[] args) {
Fenêtre f = new Fenêtre(150,300,600,400,
                    "Cliques'y si tu l'oses");
EcouteurSouris e = new EcouteurSouris();
AdapteurSouris a = new AdapteurSouris();
f.addMouseListener(e);
f.addMouseListener(a);
}
}
```

NB

ici, création d'objets écouteurs spécifiques
la fenêtre (l'objet source) pourrait être son propre écouteur

Résultat

```
run:
Ouille
Aie, touché au point 158 101
Ouille
Aie, touché au point 104 308
Ouille
Aie, touché au point 500 215
```



Généralisation

à une catégorie d'objet Xxx — `Mouse`

pour laquelle existent différents événements — `MouseClicked, etc.`

on associe un écouteur d'évènement XxxEvent — `MouseEvent`

par une méthode addXxxListener — `add MouseListener`

et

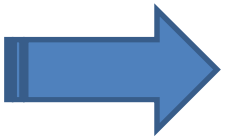
- soit on implémente toutes les méthodes de l'interface correspondante à XxxListener — `MouseListener`
- soit on utilise la classe XxxAdaptateur et on redéfinit juste les classes nécessaires — `MouseAdapter`

cf. cours/livres/sites Web spécialisés pour
- les notions (containers, layout, etc.)
- la syntaxe
- **les bonnes pratiques**

Pour aller plus loin …

Exploitation /conception d'interfaces

- Des outils pour séparer
  - le **Modèle** : données, traitements sur les données, interactions avec la base de données
  - Les **Vues** : interface utilisateur, représentation(s) manipulable(s) des données du modèle
  - Le **Contrôle** : synchronisation entre le modèle et les vues



cf. cours d'IHM

+ complément