Patrons de comportement: Observer/Observable

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

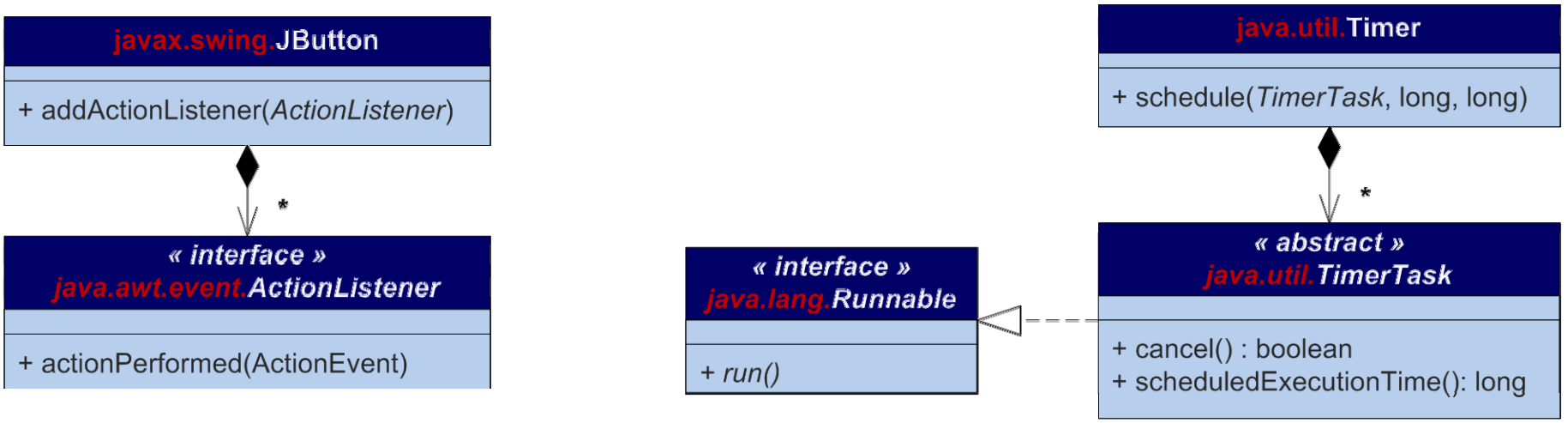
Classification

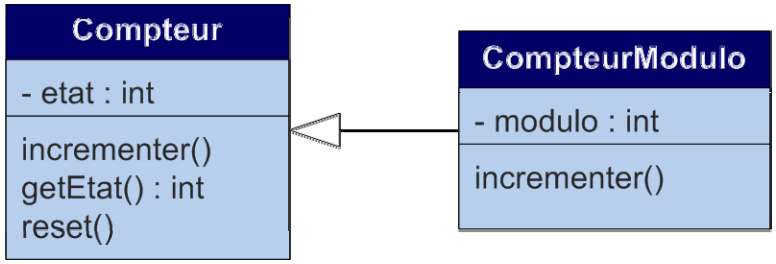
- Création
  - Comment un objet peut être créé
  - Indépendance entre la manière de créer et la manière d'utiliser
- Structure
  - Comment les objets peuvent être combinés
  - Indépendance entre les objets et les connexions
- Comportement
  - Comment les objets communiquent
  - Encapsulation de processus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

Université Nice Sophia Antipolis

Ecouteur

- Notifier un événement **à qui veut**





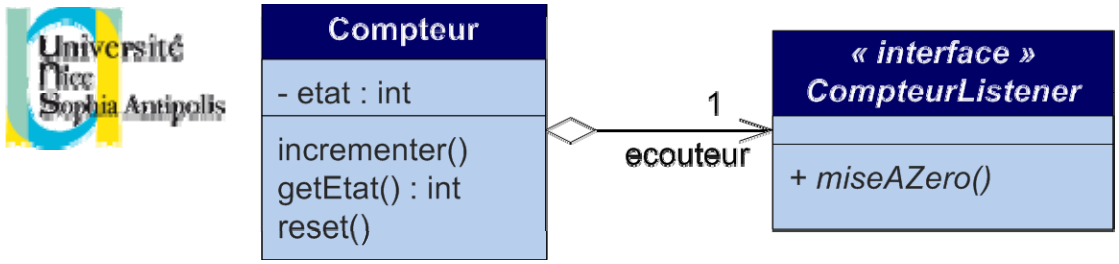
Compteur

Gestion de l'état

```
class Compteur {
  private int etat = 0;

  void incrementer(){
    this.etat += 1;
  }
  int getEtat() {
    return this.etat;
  }
  void reset() {
    this.etat = 0;
  }
}
```

Gestion de n

```
class CompteurModulo
            extends Compteur {
  private int modulo;

  void incrementer() {
    super.incrementer();
    if (getEtat() == modulo){
      super.reset();
    }
  }
}
```



Compteur

Gestion de l'état

```
class Compteur {
  private int etat = 0;
  private CompteurListener ecouteur = null;
  void incrementer(){
    this.etat += 1;
  }
  void reset() {
    if(ecouteur != null)
      ecouteur.miseAZero();
    this.etat = 0;
  }
}
```

Ecouteur

```
interface CompteurListener {
  void miseAZero();
}
```

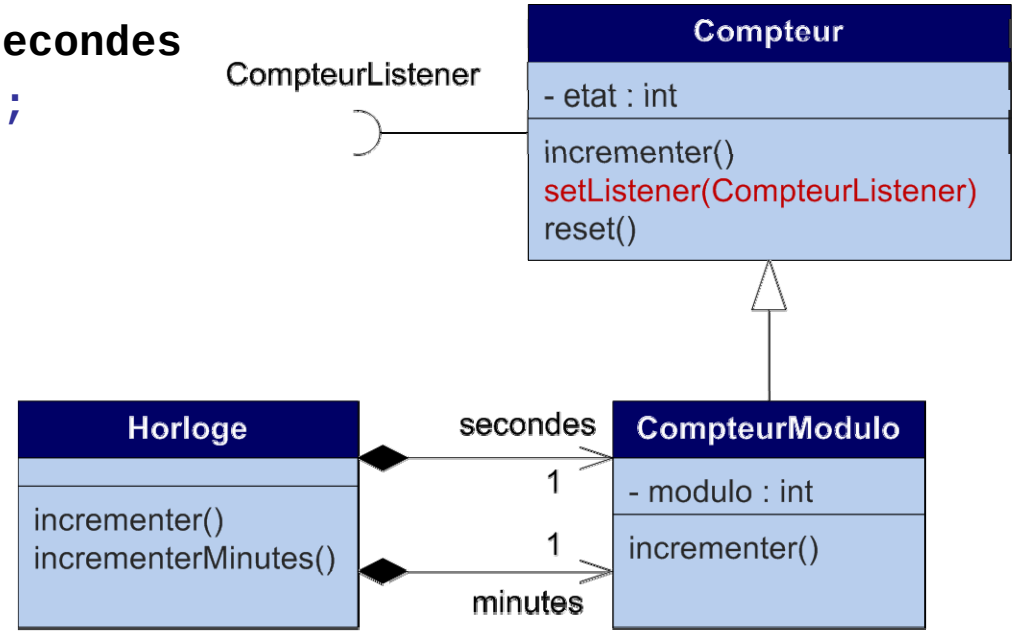
Université Nice Sophia Antipolis

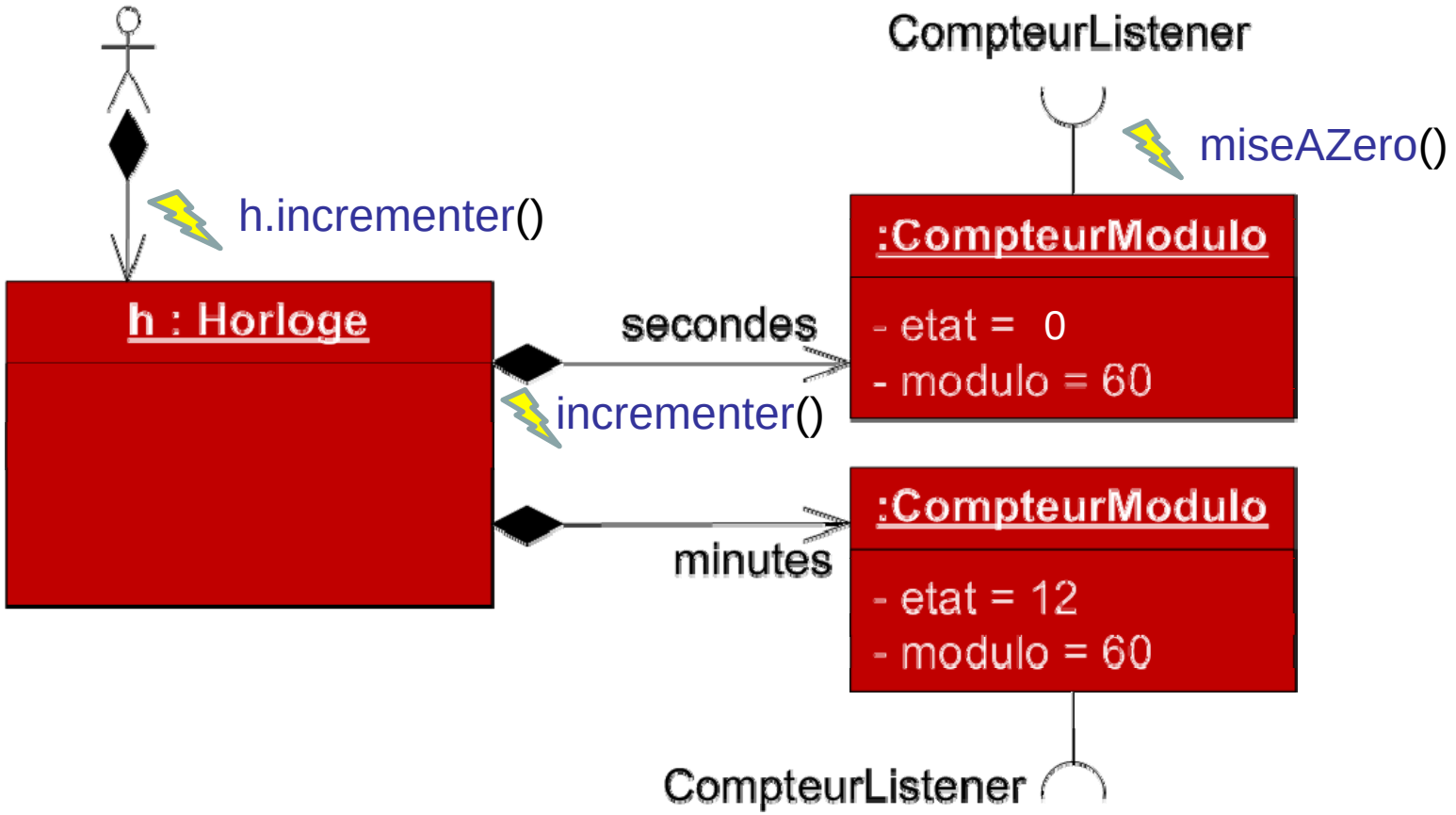
Compteur

```
class Compteur {
  private int etat = 0;
  private CompteurListener ecouteur = null;
  void incrementer(){
    this.etat += 1;
  }
  void setListener(CompteurListener l){
    ecouteur = l;
  }
  void reset() {
    if(ecouteur != null)
      ecouteur.misAZero();
    this.etat = 0;
  }
}
```

Horloge

```
class Horloge{
  private CompteurModulo secondes, minutes;
  Horloge(int sec, int min) {
    this.secondes = new CompteurModulo(60, sec);
    this.minutes = new CompteurModulo(60, min);

  }
  void incrementer() { // les secondes
    this.secondes.incrementer();
    if(this.secondes.etat == 0)
      incrementerMinutes();
  }
  void incrementerMinutes() {
    this.minutes.incrementer();
  }
}
```



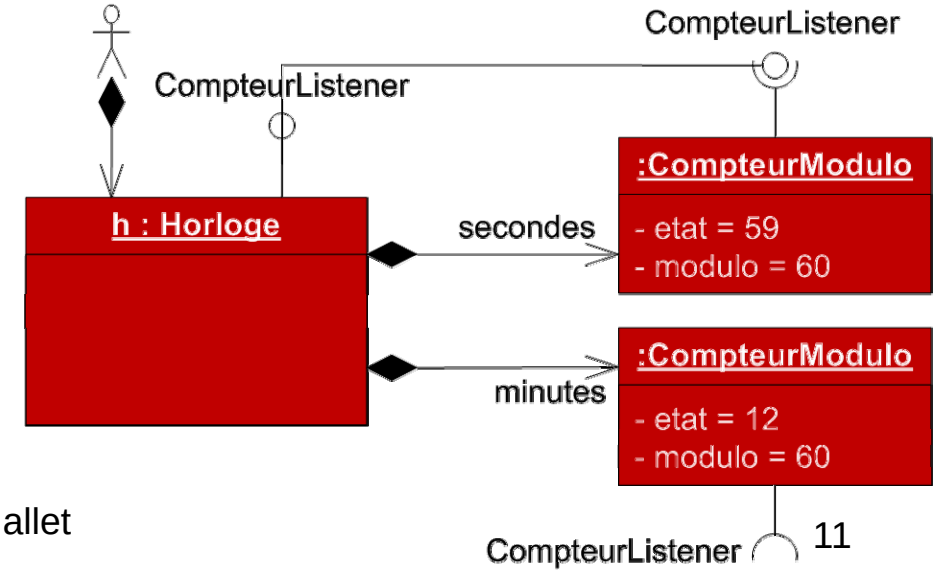


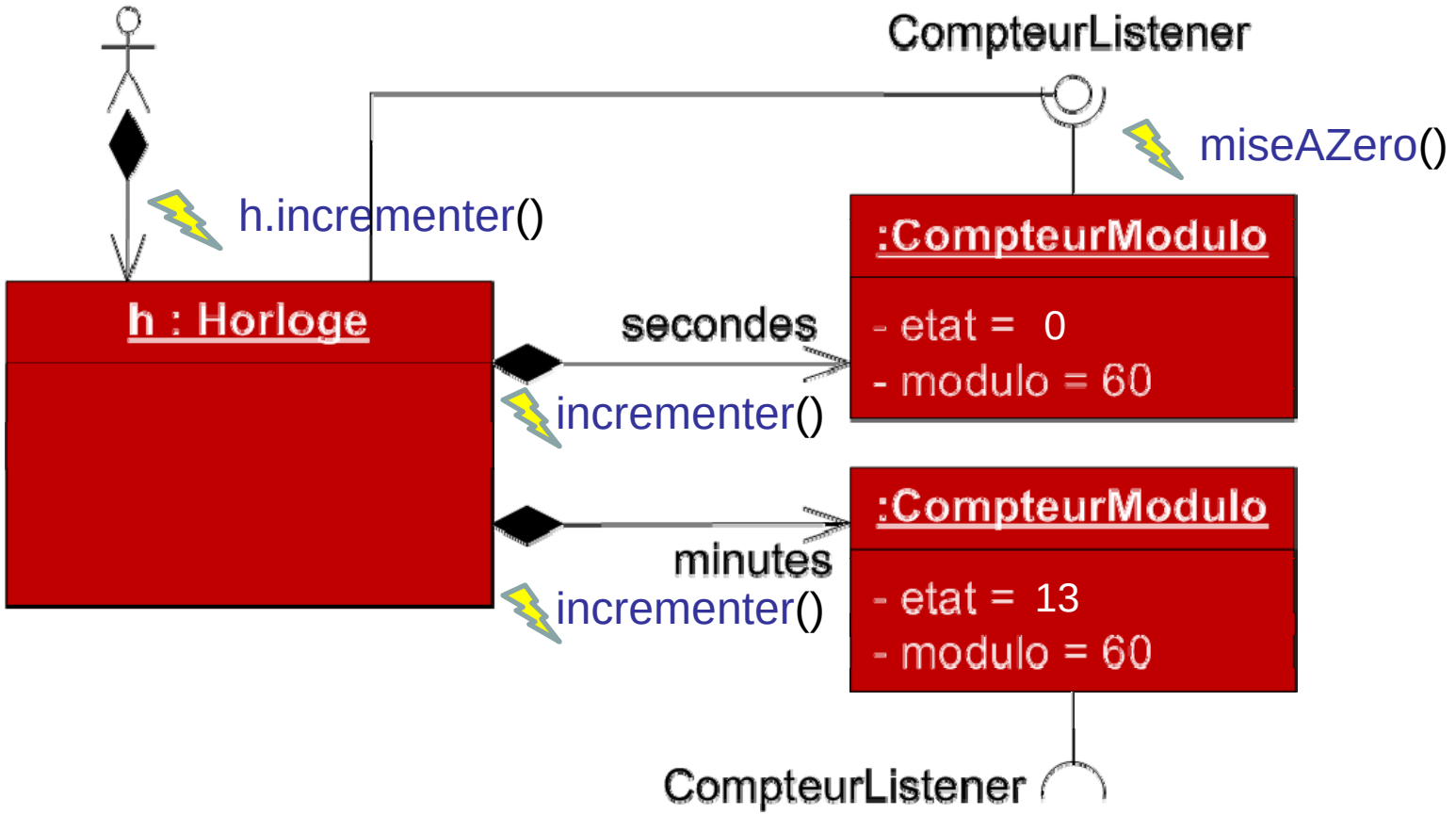
Une horloge h: 12 min 59 secondes

h.incrementer();
h.secondes.incrementer();
h.secondes.ecouteur.miseAZero();

Horloge

```
class Horloge {
  private CompteurModulo secondes, minutes;
  Horloge(int sec, int min) {
    this.secondes = new CompteurModulo(60, sec);
    this.minutes = new CompteurModulo(60, min);
  }

  void incrementer() { // les secondes
    this.secondes.incrementer();
  }
  public void miseAZero() {
    this.incrementerMinutes();
  }
  void incrementerMinutes() {
    this.minutes.incrementer();
  }
}
```





Une horloge h: 12 min 59 secondes

h.incrementer();

h.secondes.incrementer();

h.miseAZero(); h.minutes.incrementer();

Université Nice Sophia Antipolis

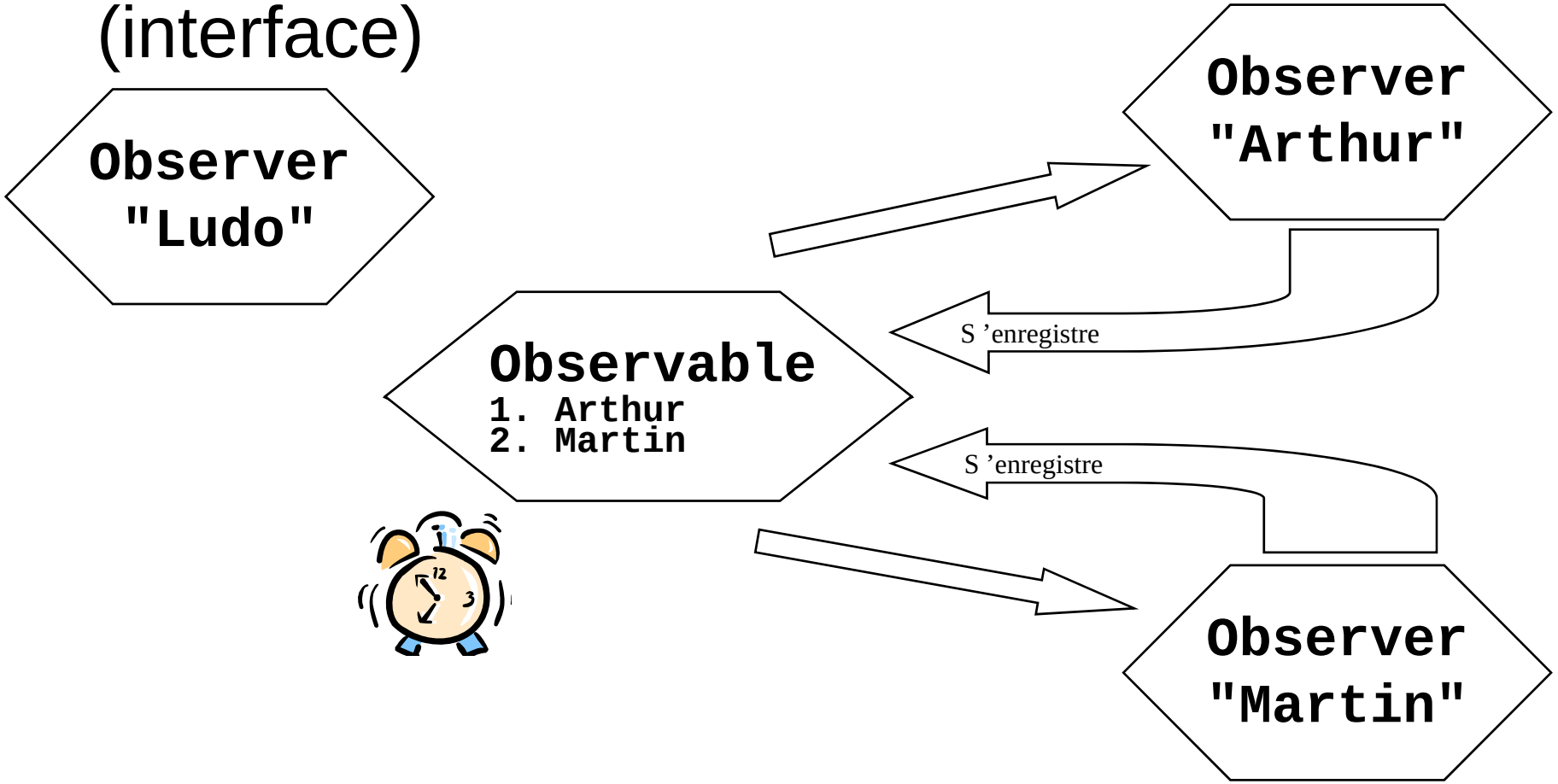
Ecouteur et Décorateur

- Un écouteur peut en cacher d'autres

```
class Ecouteurs implements CompteurListener {
  ArrayList<CompteurListener> ecouteurs =
                    new ArrayList<CompteurListener>();
  void add(CompteurListener e) {
    ecouteurs.add(e);
  }
  public void miseAZero() {
    for(CompteurListener cl : ecouteurs)
      cl.miseAZero();
  }
}
```

Observer / Observable

- Tout le monde peut être un **`Observer`** (interface)

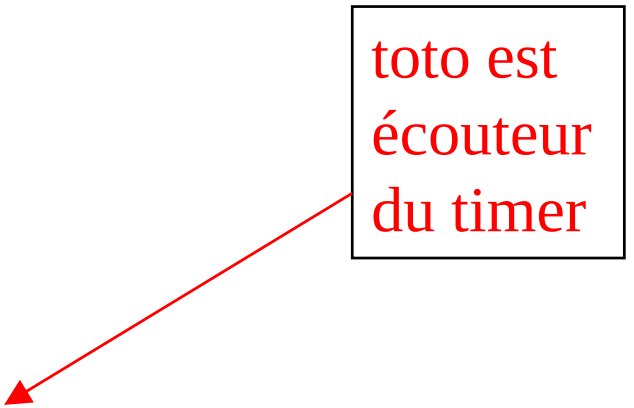


Université Nice Sophia Antipolis

Observer / Observable

Exemple : un timer

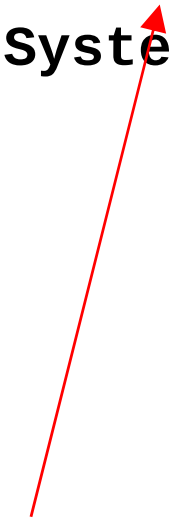
```
class Test {
  Timer timer;

  Test() {
    timer = new Timer();
    Toto toto = new Toto();
    timer.addObserver(toto);
  }

  static public void main(String[] args) {
    Test t = new Test();
    t.timer.run();
  }
}
```



Université Nice Sophia Antipolis

java.util.Observer
Foo est un observateur

```
class Foo implements java.util.Observer {
  public void update(Observable o, Object arg) {
            System.out.println("Temps :  "
                  + ((Timer)o).getSeconds()
                  + " secs");
  }
}
```



Tous les observateurs doivent posséder une méthode update(…)

Université Nice Sophia Antipolis

java.util.Observable
Exemple

```
class Timer extends java.util.Observable {
    private long zzz = 1000;
    private long zero;

    Timer(long zzz) {
      this.zzz = zzz;
    }

    public void run () throws InterruptedException {
      zero = System.currentTimeMillis();
            while (true) {
                  setChanged();
                  notifyObservers(new Long(System.currentTimeMillis()
                                    - zero));
                  TimeUnit.MILLISECONDS.sleep(zzz);
            }
    }
}
```

Les observables doivent spécialiser `Observable`

On notifie les observateurs

… appel de update(…)

Créer ses propres classes d'écouteurs

- Comment ça marche ?
- Créer ses propres classes d'écouteurs, d'événements…
  1. Ecrire l'interface que devront implémenter les écouteurs (remplacer **`Observer/ActionListener`**)
  2. Ecrire la classe qui correspond au nouveau type d'événements (remplacer **`ActionEvent`**)
  3. Ajouter des méthodes dans l'objet observable pour :
     - Pouvoir lui ajouter ou enlever des écouteurs (**`addActionListener`**),
     - Prévenir les écouteurs en leur envoyant un objet « événement » (**`notifyObservers`**)

Université Nice Sophia Antipolis

Exemple avec un minuteur

- On va définir un *minuteur* envoie des événements de type **`TimerEvent`** à ses écouteurs (de type **`TimerListener`**) lorsqu'un certain temps s'est écoulé
- Ce minuteur est un Thread

L'interface TimerListener

```
package timer;
import java.util.*;

public interface TimerListener extends EventListener {
   public void timerTriggered(TimerEvent e);
}
```

On peut choisir le nom qu'on veut ici !

Université Nice Sophia Antipolis

La classe TimerEvent

```
package timer;
import java.util.EventObject ;

public class TimerEvent extends EventObject {
   public TimerEvent(Object source) {
     super(source);
     … // ici on peut stocker les
     … // params supplémentaires éventuels
   }
}
```

On peut ajouter d'autres paramètres, mais le premier est obligatoire

Université Nice Sophia Antipolis

La classe Timer

```
package timer;
import java.awt.event.*;
import java.util.*;

class Timer extends Thread {
   public static final int DEFAULT_DELAY = 5000;
   private int delayMs;
   private ArrayList<TimerListener> timerListeners;

   public Timer() {
         this(DEFAULT_DELAY);
   }

   public Timer(int delayMs) {
         this.delayMs = delayMs;
   }
```

Université Nice Sophia Antipolis

La classe Timer (suite)

```
public synchronized void removeTimerListener(TimerListener l) {
  if (timerListeners != null) {
    timerListeners.remove(l);
  }
}

public synchronized void addTimerListener(TimerListener l) {
  if (timerListeners == null)
    timerListeners = new ArrayList<TimerListener>();
  timerListeners.add(l);
}
```

La classe Timer (suite)

```
protected void fireTimerTriggered(TimerEvent e) {
  if (timerListeners == null) return;
  for(TimerListener tl : timerListeners)
    tl.timerTriggered(e);
}

public void run () {
  while (!canceled)
    try {
      TimeUnit.MILLISECONDS.sleep(delayMs);
      fireTimerTriggered(new TimerEvent(this));
    } catch (Exception e) { }
 }
}
```

Notification des écouteurs

Exemple d'écouteur

```
import timer.*;
public class TestTimer {
   Timer timer1 = new Timer(1000);
   public TestTimer() {
     timer1.addTimerListener(new TimerListener() {
       public void timerTriggered(TimerEvent e) {
         System.out.println("Une sec s'est écoulée…");
       }
     });
     timer1.start();
   }
}
```

Classe interne anonyme dans cet exemple…