Inf 431 – Cours 8

Programmation par objets
Librairies graphiques

jeanjacqueslevy.net

secrétariat de l'enseignement:
Catherine Bensoussan
cb@lix.polytechnique.fr
Aile 00, LIX,
01 69 33 34 67

www.enseignement.polytechnique.fr/informatique/IF

Plan

Sous-classes $\leftrightarrow$ sous-typage (1/2)

- Notation :
  $x : t$ signifie que la variable Java x est de type t.
  $t <: t'$ signifie $x : t \Rightarrow x : t'$ pour tout x. (conversion implicite)
  On dit que t est un sous-type de t'.
- En Java,
  byte $<:$ short $<:$ int $<:$ long float $<:$ double char $<:$ int
  $C <: C'$ si C est une sous-classe de C'.
  Par exemple `PointC` $<:$ `Point`.
- Posons
  $t \to t'$ pour le type des fonctions de t dans t',
  $t \times t'$ pour le type du produit cartésien de t et t'.
- Alors
  $t <: t'$ et $u <: u' \quad \Rightarrow \quad t' \to u <: t \to u'$
  $t <: t'$ et $u <: u' \quad \Rightarrow \quad t \times u <: t' \times u'$.
  ($<:$ contravariant à gauche et covariant à droite de $\to$)

Sous-classes $\leftrightarrow$ sous-typage (2/2)

- Exemple de dérivations de types :
  `rotation: Point` $\times$ double $\to$ `Point`
  `PointC` $<:$ `Point`

  `PointC` $\times$ double $<:$ `Point` $\times$ double
  `Point` $\times$ double $\to$ `Point` $<:$ `PointC` $\times$ double $\to$ `Point`
  `rotation: PointC` $\times$ double $\to$ `Point`

  mais on ne peut pas dériver :
  `rotation: PointC` $\times$ double $\to$ `PointC`

Exercice 1 Donner la loi de sous-typage pour les tableaux.

Théorie plus compliquée en présence de polymorphisme ou de méthodes binaires
$\Rightarrow$ sous-classe ne correspond plus à la notion de sous-typage.
$\Rightarrow$ Ocaml, GJ (Generic Java = Java 1.5), Generic C#

La théorie des types est introduite dans le cours Principes des Langages de Programmation de Majeure 1. Cf. livre de Benjamin Pierce ou de Abadi-Cardelli.

Notation objet et héritage

C est une classe et o un objet de cette classe ($o : C$)

	statique	« dynamique »
variable	$C.x$	$o.x$
fonction	$C.f()$	$o.f()$

En terminologie objet, fonction = méthode.

Exemples quand on déclare `String s,t;` et `Point p;`

	`System.in`	`p.x`
	`Point.rotation(p)`	`s.equals(t)`

- $o.f(\ldots)$ est l'application de la méthode la plus spécialisée s'appliquant à o.
- C'est l'application de f tel que

$o : C$	f méthode de C	C minimum pour $<:$

  Dans l'évaluation de f, alors this vaudra o.

Exercice 2 Expliquer la notation `System.out.println`.

Héritage et surcharge (1/3)

- la surcharge est déterminée à la compilation
- l'héritage est dynamique ;
  la méthode est sélectionnée à l'exécution
- l'héritage permet d'appeler une méthode sans savoir exactement quelle sous-classe va l'instancier (liaison tardive) ;
  objets = monde ouvert

Exercice 3 Quelle est la valeur imprimée par le programme suivant ?

```
class C {
  void f() { g(); }
  void g() { System.out.println(1); }
} }

class D extends C {
  void g() { System.out.println(2); }

  public static void main (String[ ] args) {
    D x = new D();
    x.f();
} } Exécution
```

Héritage et surcharge (2/3)

Exercice 4 Quel est le résultat produit par le programme suivant quand $T, T' \in \{A, B\}$ et $U, U' \in \{A, B\}$ avec $T' <: T$ et $U' <: U$?

```
class A {
  public static void main (String[ ] args) {
    T x = new T'(); U y = new U'();
    System.out.println (x.f(y));
  }

  int f (A y) {return 1; }
}

class B extends A {
  int f (B y) {return 2; }
} Exécution
```

Héritage et surcharge (3/3)

- héritage simple, chaque classe a une seule classe parente : super
- super.f est la méthode f de la classe parente
- super() (avec d'éventuels arguments) est le constructeur (appelé par défaut) de la classe parente

- un champ final ne peut être redéfini.
  (On peut donc optimiser sa compilation).
- pour définir des constantes, on écrit
  `final static int JAUNE = 0, ROUGE = 1;`
- une classe final ne peut être spécialisée.

Classes abstraites – Interfaces (1/2)

- Une classe abstraite contient des champs indéfinis.
- $\Rightarrow$ on ne peut créer que des objets de classes non abstraites.

- Un interface est une classe abstraite dont tous les champs sont indéfinis.
  Ses champs de données sont constants ;
  ses méthodes sont abstraites et publiques.
- Un interface peut spécialiser un autre interface.
- Une classe peut implémenter un ou plusieurs interfaces
  $\Rightarrow$ Héritage multiple pour les interfaces.
- Notion différente des interfaces de Modula, ML, Ada, Mesa. En Java, la classe qui les implémente porte un nom différent.
- Pas de fonctions statiques, ou champs de données modifiables dans un interface.
- Les interfaces sont une bonne manière d'exiger la présence de certains champs dans une classe.

Classes abstraites – Interfaces (2/2)

```
public interface Couleur {
  final int ROUGE = 0, JAUNE = 1;
}

interface PointCirculaire {
  void static rotation (Point p, double theta);
}

interface PointMobile {
  void static translation (Point p, double dx, double dy);
}

class Point implements PointCirculaire, PointMobile {
  ...
}
```

Exercice 5 Quel type utilise-t-on pour référencer les objets d'une classe implémentant un interface ?

Exercice 6 Même question à l'intérieur de la classe implémentant l'interface ?

Initialisation des classes

Le mot-clé static suivi d'un bloc d'instructions permet de faire une initialisation de la classe à son chargement (lors du premier accès à un de ses champs).

```
class Point {
  double x, y;
  static void translation (Point p, double dx, double dy) {
    p.x = p.x + dx; p.y = p.y + dy;
  }

  static Point rotation (Point p, double theta) {
    double x = p.x * Math.cos(theta) - p.y * Math.sin(theta);
    double y = p.x * Math.sin(theta) + p.y * Math.cos(theta);
    return new Point(x, y);
  }

  static {
    System.out.println ("La classe Point vient d'être chargée");
  }
}
```

Racine de la hiérarchie des classes

`Object` est la classe la plus générale :

$$\forall C \quad C <: \texttt{Object}$$

(C est une classe ou un tableau quelconque).

- La hiérarchie des classes est une simple arborescence.
- Les méthodes de `Object` sont
  `clone`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `wait`, `equals`, `toString`.
- Seules `equals`, `toString`, `finalize` peuvent être redéfinies.
- Tous les objets contiennent ces méthodes.

- On convertit les scalaires int, float, char, ... en objets avec un « conteneur » :

```
int x = 3;
Objet xobj = new Integer(x);
int y = xobj.intValue();
```

Réflexivité

- Toutes les classes C dans un environnement courant sont réifiées en objets de la classe `Class`.

```
void printClassName(Object x) {
  System.out.println("La classe de " + x
                     " est " + x.getClass().getName());
}
```

  $\Rightarrow$ les données de l'interpréteur (JVM) peuvent être réifiées.

- On peut agir sur l'environnement courant (réflexion)
- Ce genre de programmation est à banir, car souvent obscure.
- La réflexivité est utilisée pour faire des débogueurs.

Polymorphisme (1/3)

Pas de polymorphisme en Java (avant 1.5). Pour l'approximer, on peut tenter de se servir de la classe `Object`.

```
class Liste {
  Object valeur; Liste suivant;

  static int length (Liste x) {
    if (x == null) return 0;
    else return 1 + length (x.suivant);
  }

  static Liste append (Liste x, Liste y) {
    if (x == null) return y;
    else return new Liste(x.valeur, append(x.suivant, y));
  }
}
```

Marche pour `length`, mais pas pour `append` où on doit faire des conversions explicites.

Polymorphisme (2/3)

ou en style orienté-objet.

```
abstract class Liste {
  abstract int length () ;
  abstract Liste append (Liste y);
}

class Nil extends Liste {
  int length () { return 0; }
  Liste append (Liste y) { return y; }
}

class Cons extends Liste {
  Object valeur; Liste suivant;

  static int length () { return 1 + suivant.length(); }
  static Liste append (Liste y) {
    return new Cons(valeur, suivant.append(y));
  }
}
```

Exercice 7 Comment calculer la longueur d'une liste d'entiers à partir de la fonction précédente.

Polymorphisme (3/3)

- GJ (*Generic Java*) défini par [Bracha, Odersky, Stoutamire, Wadler, 1997].
- En Java 1.5, on peut avoir des variables de types

```
public class LinkedList<E> {
  void add (int i, E x) { ... }
  E get (int i) { ... }
  E getFirst () { ... }
  ListIterator<E> listIterator(int i) { ... }
}

public interface ListIterator<E> extends Iterator<E>
  boolean hasNext();
  E next();
}
```

- génériques de Java $\neq$ génériques de C++ (*templates*)
  pas de duplication du code
- polymorphisme total $\Rightarrow$ ML, Caml, Ocaml

Procédural ou Objets ?

(bis)

- choix du contrôle : par les fonctions ou par les données ?
- le style de programmation diffère entre petits programmes et gros programmes ($> 10^4$ lignes).
- dépend de la stratégie de modification.
- en dernière analyse : affaire de goût.
- objets $\neq$ modularité.
- programmation par objets utilisée dans :
  - les boîtes à outils graphiques (*look* commun à toutes les fenêtres)
  - réseau (données et méthodes se déplacent simultanément).
- programmation incrémentale très populaire dans l'industrie.

MacLib (1/2)

Initialement compatible avec *QuickDraw* du MacInstosh.

```
class Lissajoux {
  static final double X0 = 100, Y0 = 100, K = 80, A = 3, B = 2, D = 0.9;

  static int X(double t) { return (int) (X0 + K*Math.cos(A*t)); }
  static int Y(double t) { return (int) (Y0 + K*Math.sin(B*t+D)); }

  public static void main(String[ ] args) {
    final int N = 120;
    double t = 0;
    MacLib.initQuickDraw(); // initialisation du graphique
    MacLib.moveTo(X(0), Y(0));
    for (int i = 0; i < N; ++i) {
      t = t + 2 * Math.PI/N;
      MacLib.lineTo(X(t), Y(t));
} } } Exécution
```

Merci à Philippe Chassignet (LIX).

Paquetages et Classes graphiques

On utilisera (au choix) 3 bibliothèques graphiques :

- MacLib, plus simple, locale à l'X. Pour trouver la documentation consulter le lien suivant

  http://www.enseignement.polytechnique.fr/profs/informatique/Philippe.Chassignet/MACLIB/Java/maclib_java.html
- AWT (*Abstract Window Toolkit*) de Sun Microsystems, dans la version 1.1. Orienté-objet. Compatible avec les classes des appliquettes *applet*, exécutables sous un navigateur.
- Swing de Sun Microsystems, pour les versions plus récentes de JDK. Avec une boîte à outils plus garnie.

MacLib (2/2)

Ou avec la notation objets :

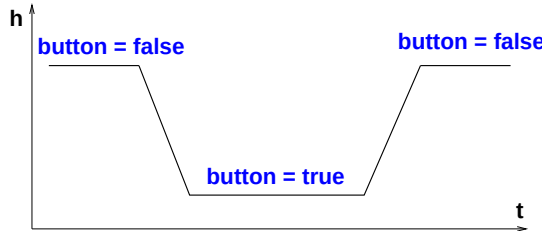
```
class Lissajoux {
  static final double X0 = 100, Y0 = 100, K = 80, A = 3, B = 2, D = 0.9;

  static int X(double t) { return (int) (X0 + K*Math.cos(A*t)); }
  static int Y(double t) { return (int) (Y0 + K*Math.sin(B*t+D)); }

  public static void main(String[ ] args) {
    final int N = 120;
    double t = 0;
    GrafPort g = new GrafPort("mon dessin");
    g.moveTo(X(0), Y(0));
    for (int i = 0; i < N; ++i) {
      t = t + 2 * Math.PI/N;
      g.lineTo(X(t), Y(t));
} } }
```

Permet d'avoir plusieurs `GrafPort`.

Entrée graphique (1/3)



Front descendant

(ne pas oublier d'attendre le front montant)

```
for(;;) {
    while (!g.button())
        ;
    Point p = g.getMouse ();
    while (g.button())
        ;
    action (p.h, p.v);
}
```

Attente active (*busy wait*).

Front montant (plus sûr)

```
for (;;) {
    while (!g.button())
        ;
    while (g.button())
        ;
    Point p = g.getMouse ();
    action (p.h, p.v);
}
```

Entrée graphique (2/3)

- in.read() est blocant
  ⇒ problème pour lire simultanément le clavier et la souris.
- problème de la séquentialité. Il faut gérer l'asynchronie.
- solution 1 : avoir un *read* non blocant
- solution 2 : faire des processus et gérer le parallélisme asynchrone.
- solution 3 : avoir une structure d'événements gérés par le système. Par exemple le pilote (*driver*) d'événements de X-window.
- solutions plus sophistiquées
  - *callbacks* dans la programmation par objets
  - langages spécialisés pour gérer modulairement les interactions : Squeak [Cardelli-Pike] ; Esterel [Berry- ···] cas particulier des systèmes réactifs.
- pour double-click, il faut estampiller les événements par le temps.

Entrée graphique (3/3)

- En style procédural :
  Un programme d'interaction a typiquement la structure suivante

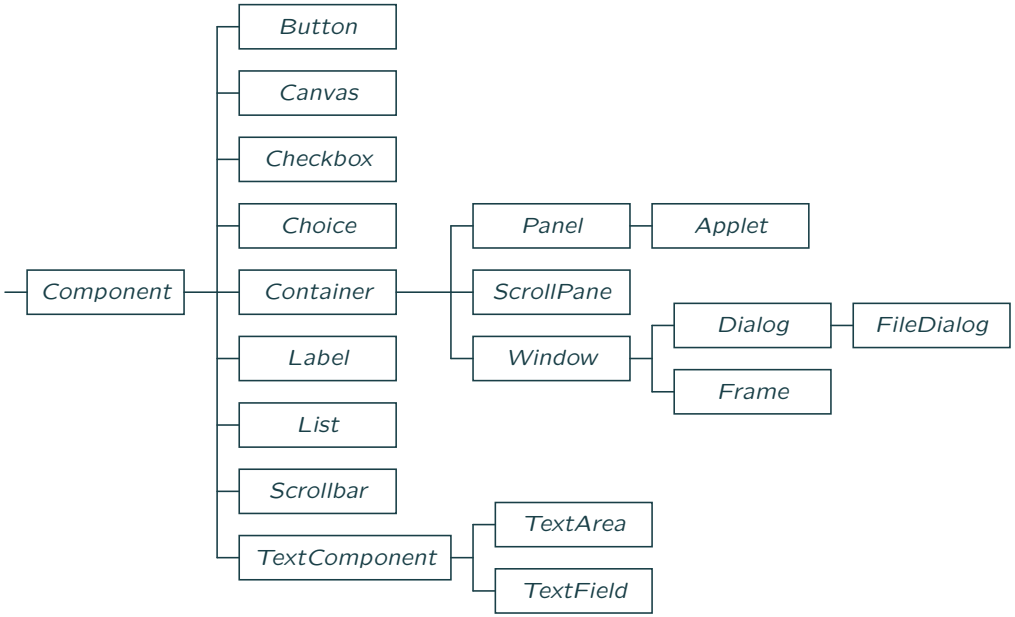
```
for (;;) {
  Event e = nextEvent();
  switch (e.type) {
  case keyPressed: ... break;
  case keyReleased: ... break;
  case mousePressed: ... break;
  case mouseReleased: ... break;
  ...
  }
}
```

  ⇒ peu modulaire, car la boucle principale d'interaction *multiplexe* toutes les interactions.
- En programmation par objets : La structure des classes permet d'y remédier en hiérarchisant les récepteurs possibles pour chaque événement (cf. AWT).
- Remarque : en style procédural, avec la présence de primitives pour les automates non déterministes, on y arriverait tout aussi bien (exemple des langages réactifs).

Librairie AWT

Un interface utilisateur graphique (*GUI*) est formé de :

- composants (*component*) : boutons, canevas, *checkbox*, choix, *container*, étiquette, liste de textes, barre de déroulement, texte, etc.
- conteneurs (*container*) : listes de composants, affichés de devant vers l'arrière. Plusieurs types de conteneurs : les panneaux, les appliquettes, les zones de déroulement, les fenêtres ou les cadres (*frames*), ...
- détecteurs (*listeners*) d'événements attachés à des composants par la méthode `addXXXListener`.
  On spécialise les méthodes dans ces détecteurs pour engendrer des actions spécifiques.
- Modèle/Présentation/Contrôle (*MVC*) sont séparés (comme en Smalltalk).



awt

Contexte graphique

```
class Frame3 extends Frame {
  Frame3 (String s) {
    super(s);
    add ("North", new Label ("Bonjour les élèves!", Label.CENTER));
    setSize (300, 400);
    setVisible(true);
  }

  public static void main (String[ ] args) {
    Frame3 f = new Frame3 ("Frame2");
  }

  public void paint (Graphics g) {
    int w = getSize().width;
    int h = getSize().height;
    int x = w/2; int y = h/2;
    g.setColor(Color.red);
    g.fillRect(x-w/4, y-w/4, w/4, w/2);
    g.setColor(Color.yellow);
    g.fillRect(x, y-w/4, w/4, w/2);
  }
}
```

Exécution

Les méthodes `paint` et `repaint` affichent les composants.

Fenêtre graphique

```
import java.awt.*;
class Frame1 {
  public static void main (String[ ] args) {
    Frame f = new Frame ("Bonjour");
    f.add ("Center", new Label ("Bonjour les élèves!", Label.CENTER));
    f.setSize (300, 100);
    f.setVisible(true);
} }
```

Exécution

ou en programmation par objets :

```
class Frame2 extends Frame {
  Frame2 (String s) {
    super(s);
    add ("Center", new Label ("Bonjour les élèves!", Label.CENTER));
    setSize (300, 400);
    setVisible(true);
  }
  public static void main (String[ ] args) {
    Frame2 f = new Frame2 ("Bonjour");
} }
```

Exécution

Un bouton avec interaction

```
import java.awt.*;
import java.awt.event.*;

public class Frame4 extends Frame {
  Button q = new Button ("Quit");

  public Frame4 (String title) {
    super (title);
    setLayout (new FlowLayout());
    add(q);
    setSize (300, 100);
    q.addActionListener (new Quit());
    setVisible(true);
  }

  public static void main (String[ ] args) {
    Frame4 f = new Frame4 (args[0]);
} }

class Quit implements ActionListener {
  public void actionPerformed (ActionEvent e) {
    System.exit(0);
} }
```

Exécution

Deux boutons et un texte

```
public class Frame5 extends Frame {
  Button q = new Button ("Quit");
  Button a = new Button ("A");
  TextField t = new TextField (20);

  public Frame5 (String title) {
    super (title); setLayout (new FlowLayout());
    add(q); add(a); add(t);
    setSize (300, 100);
    q.addActionListener (new Quit());
    a.addActionListener (new ActionA(t, "Bouton a!"));
    setVisible(true);
  }
  ...
} }

class ActionA implements ActionListener {
  TextField t; String s;
  ActionA (TextField t0, String s0) { t = t0; s = s0; }

  public void actionPerformed (ActionEvent e) {
    t.setText(s);
} } Exécution
```

Evénements souris

- événements souris : bouton enfoncé, bouton relaché, souris déplacée, souris déplacée bouton enfoncé (*drag*), souris entrant sur un composant, souris sortant d'un composant.
- un détecteur d'événements-souris `MouseListener` est un interface contenant 5 méthodes `mousePressed`, `mouseReleased`, `mouseClicked`, `mouseEntered`, `mouseExited`.
- un `MouseAdapter` est une implémentation de l'interface précédent avec les 5 méthodes vides. Ainsi on ne définit que les méthodes à spécifier.
- les méthodes `getX()`, `getY()`, `getPoint()` retournent les coordonnées de l'événement.

Exercice 8 Programmer un petit interface utilisateur où on imprime les coordonnées du point courant sur un clic souris.

Exercice 9 Programmer un petit interface utilisateur qui dessine un vecteur entre deux points rentrés à la souris.

Une interaction à la souris

```
public class Frame6 extends Frame {
  public Frame6 (String title) {
    super (title);
    setLayout (new FlowLayout());
    add(q); add(a); add(t);
    validate();
    pack();
    addMouseListener (new ActionB(t, "Bouton relache'."));
    q.addActionListener (new Quit());
    a.addActionListener (new ActionA(t, "Bouton a!"));
    setVisible(true);
  }
  ...
}

class ActionB extends MouseAdapter {
  TextField t; String s;
  ActionB (TextField t0, String s0) {t = t0; s = s0; }

  public void mouseReleased (MouseEvent e) {
    t.setText(s);
  } Exécution
```