

I-Langage Java : Caractéristiques

- 1995 - Sun Microsystems - James Gosling - nom = café
 - Sun distribue gratuitement les outils pour développer en java:
`http://java.sun.com/products/jdk/1.2`
`.../docs/index.html`
 - langage objet (objet =zone mémoire organisée selon Classe (~type))=> simplification maintenance
 - langage avec code intermédiaire (javacode) créé par un compilateur (qui effectue la grosse partie de traduction entre le langage humain (ou presque!) et le langage machine): intérêt le javacode est commun à toutes sortes de plate-formes, l'interpréteur spécifique a chaque système ne fait que la petite partie de la traduction au moment de l'exécution.
 - Mise en oeuvre :
 - Compilation avec le programme compilateur: `.../bin/javac`
`javac ClasseX` (pour compiler le fichier `ClasseX.java` => `ClasseX.class` (1ère lettre majuscule pour les classes). `ClasseX.class` est du javacode
 - Exécution interprétée sous plusieurs formes:
 - "en **Applet**" = lancé par une page html depuis un browser ou appletviewer `xxx.html`
 - ou -"en **application**" lancé par `java ClasseX`, mais demande une fonction "main" en plus qui fait ce que fait le browser.
 - Résumé du développement d'un programme:
 - Edition avec `jfe` ou `editPlus` ou ...bloc notes
 - Compilation avec `...jdk12/bin/javac ClasseX.java`
 - Interprétation (exécution): `jdk12/bin/java ClasseX` (exécution à partir du « main » écrit dans `ClasseX.java`)
- Et puis on recommence puisqu'il y a certainement(?) des fautes a corriger !!
- Rien de bien neuf du point de vue de la syntaxe des types primitifs(int, float etc ...), c'est pratiquement celle du C.
- Et de nombreux "tutorials" sur le Web
- Java est un langage => apprentissage par l'exemple et à base de copier-coller!!(l'invention du siècle, et soyons paresseux : utilisons ce qui est déjà fait).
- Les langages objets ont été créés pour simplifier la maintenance (60% du coût du développement informatique), c'est réussi, si bien que tout programme n'est que la modification d'un ancien programme !

II-Premier programme:

Le "Panel"(=Panneau rectangulaire) et programme Applet est lancé à partir d'un fichier html

Donc deux fichiers à écrire avec un traitement de texte: un fichier html et code source java

```
<html>...
<body>
<applet code=Coucou.class width=300 height=300>
<! Exemple: transmission de maCouleur= »gris » >
<param name="maCouleur" value="gris">
java n'est pas activé dans votre navigateur!.
</applet>
</body>
</html>
```

Le code source en java:

Nom du fichier: Coucou.java // obligatoire : nom de la classe avec Majuscule, extension java

```
import java.applet.*;
import java.awt.*;
public class Coucou extends Applet // le nom Coucou est public : accessible
//de l'extérieur du fichier
```

```
//extends applet : classe Coucou est une extension (variante de la classe prédéfinie Applet
{ Color color = Color.blue; // ***Rajout 1
  Font bigFont= null; // ***Rajout 2
  // zone commune de l'applet = connue de toutes les fonctions de la classe
  // Noter les lettres minuscules/majuscules (objet et méthode/ classes)
```

Le minimum : la fonction paint() appelé par le système, donc de format imposé. Noter que définir cette fonction consiste à dire au système « Si tu (systeme) veux faire apparaître quelque chose dans le rectangle lorsque tu le jugeras utile (par exemple parce que la fenêtre qui cachait le panneau rectangle a été fermée), il faut dessiner ceci ! »

On se rapproche ainsi d'une programmation « déclarative » (un programme = suite de règles = s'il faut peindre, faire comme ceci etc.) différente de la programmation « impérative » (un programme = succession d'ordres = faire ceci puis ceci, puis cela etc.)

// La méthode standard (format imposé) qui dit ce que le système doit faire apparaître dans le panneau

```
public void paint(Graphics g) // g est une zone mémoire avec les caractéristiques connues du système
{
  // du « contexte graphique » type de // police, couleur du fond etc..
  if (bigFont != null) g.setFont(bigFont); // Rajout 2
  g.drawString("Hey hey hey", 20, 20); // dessin du texte (String) en x,y.
  g.setColor(this.color); // Rajout 1
  g.drawString("Helloow World", 20, 40);
}
```

// Méthode init() automatiquement appelée par le *browser* quand l'applet est lancée

// si elle n'est pas définie (=écrite), c'est la fonction init() de Applet qui est appelée

```
public void init() ***Rajout 1
{
  bigFont = new Font("Arial", Font.BOLD, 16); // Rajout 2
  Color une_couleur = new Color(255, 0, 0); // On crée un objet de type Color (classe
prédéfinie). Un objet = zone mémoire avec les caractéristiques : composantes RVB ici, de l'objet
  this.color = une_couleur; // La couleur à utiliser de l'objet courant (=this) sera une_couleur
  String colortxt = this.getParameter("maCouleur"); // Rajout 3
  if (colortxt.equals("gris")) this.color = Color.gray; // Rajout 3
  } // fin fonction (=méthode) init()
} // end class Coucou
```

Transformation en application autonome

(Pas tout à fait vrai, pas de link avec les bibliothèques, le fichier créé n'est pas un xxxx.exe)

Que rajouter pour transformer en application?

- une fonction "main" toujours définie de cette façon (prototype (format) imposé parce qu'elle est appelée par le système), nécessairement à l'intérieur de la définition d'une classe, ici dans la classe Coucou

```
public static void main(String arg[]) { // arg[] = param. Sur ligne de commande
Frame f = new Frame("titre apparaissant sur la fenêtre à l'exécution");
modAppl appl = new Coucou(); // création de la zone mémoire des caractéristiques du programme
f.setSize(460, 520);
appl.init();
try { if (arg[0].equals("gris")) color = Color.gray; // essai lecture ligne de com.
} catch (Exception ex) {} // interception erreur si rien à lire : pas de arg[]
f.add("Center", appl);
f.show();
appl.start();
}
```

Le "debugging" élémentaire : on peut insérer où l'on veut:

```
System.out.print("\nla variable chaine :"+txt + ", l'âge du capitaine :",a );  
// ecrit sur le stream out
```

Nota sur la terminologie :

Ecrire un programme c'est **définir** (= dire tout) une ou des classes.

Définir n'est pas équivalent à **déclarer** (= dire que ca existe, ex. déclarer un enfant à sa naissance !).

Ex. : En C ou java : `int x=3` ; Déclaration ou définition ? `int x` : déclaration et `x=3` : définition.

Objet (Object): zone mémoire regroupant des caractéristiques (comme un enregistrement dans une base de données ou une fiche).

Classe (Class): façon d'organiser ces caractéristiques + méthodes (= fonctions) en principe appliquées à ces caractéristiques.

Construire un objet de classe classeX (= de type classeX) : réserver une zone mémoire pour icelui.

Appeler une fonction (ou une méthode), dans un programme : faire en sorte qu'elle soit exécutée au moment où on l'appelle à l'exécution du programme (un programme étant une séquence d'instructions données par le programmeur (vous) à la machine (votre esclave) qui n'obéira que lorsque l'utilisateur (pas nécessairement vous) lancera le programme.

III- DU VISUEL ! (Dessin et composants de base)

Contrairement au C, java inclut à la base des possibilités visuelles importantes incontournables dans un programme moderne!

```
import java.awt.*;  
import java.applet.*;
```

```
On peut rajouter dans paint() ceci :  
// On dessine un rectangle (xg,yh, width,height); // draw : dessin du contour  
g.drawRect(100,100,100,100);  
// On dessine un rectangle plein : fill : remplissage  
g.fillRect(110,110,80,80);
```

```
// Changement de couleur du "crayon", variable du "contexte graphique" g  
Color ohMaJolieCouleur =new Color(rrr,ggg,bbb) ; // des goûts et des couleurs ....  
g.setColor(ohMaJolieCouleur);
```

```
// a circle (int x, int y, int width, int height,int startAngle, int  
arcAngle);
```

```
g.fillArc(120,120,60,60,0,360);  
g.drawOval((xg,yh, width,height); // dessin d'un cercle en dessinant l'ovale inscrit dans un rectangle  
g.setColor(Color.yellow); // couleur standard : yellow est une variable statique de la classe Color
```

`static` : variable non associée à un objet particulier mais à une classe (définition pratique, en fait var. ou méthode implantée non pas dans la zone mémoire correspondant à un objet, mais en permanence même s'il n'y a pas d'objet de la classe)

```
// Draw a line (int x1, int y1, int x2, int y2)  
g.drawLine(140,140,160,160);  
// remise de la couleur de dessin à noir  
g.setColor(Color.black);  
}
```

```
}
```

```
// Dessine un polygone, apres avoir récupéré la largeur et la hauteur du contenant (le panneau-
programme = Applet)
int n=7; int ww= this.getSize().width; int hh= this.getSize().height;
int xx = new int[7]; int yy = new int[7];
for (int i=0; i<n; i++) {
xx[i]= ww/2 + ww/3*Math.cos(2*Math.PI*i/n); // attention : méthode statiques
yy[i]= hh/2 + hh/3*Math.sin(2*Math.PI*i/n);
}
g.fillPolygon(xx,yy,n);
}
// polyline() idem, mais non fermé => courbe mathématique
```

Boutons etc. ("basic components") sans action programmée

```
// Création d'un objet de la classe Button avec texte associé
// objet=zone-mémoire avec caractéristique
okButton = new Button("A button");
// Zone de texte à éditer
nameField = new TextField("A TextField",100);
// Groupe de boutons radio (un seul est sélectionné)
radioGroup = new CheckboxGroup();
// Création de chaque bouton : son texte, son groupe, son état
radio1 = new Checkbox("Radio1", radioGroup,false);
radio2 = new Checkbox("Radio2", radioGroup,true);
// Label (étiquette) et case à cocher
option = new Checkbox("Option",false);
// Choix multiple (~ menu)
Choice ch=new Choice();
ch.addItem("rouge");
ch.addItem("jaune");
Pour écrire un traitement de texte :
TextArea monPtitWord= new TextArea(33,44);
```

Lorsqu'un paint() de l'applet est déclenché (par le système), les composants créés doivent aussi être redessinés, il faut donc les associer à leur contenant : "container" le panneau applet (). Après leur création, en général dans init().

```
add(nameField); // on peut aussi écrire this.add(nameField);
this.add(okButton); //association à l'objet courant (this) ici l'objet de la
// classe Applet parce ligne écrite dans une fonction de cette classe
add(radio1);
add(radio2);
add(option);
add(ch);
```

Le dessin du contenant et du contenu peut se faire sur plusieurs niveaux si le contenu est lui-même contenant (arborescence visuelle).

Ne pas confondre cette arborescence visuelle avec l'arborescence des classes !!

IV- Gestion des événements-actions (Si à l'exécution un bouton ou la touche <Enter> est pressé)

```
/* On implémente une action "réflexe" (call-back) lancée par le système quand
l'utilisateur cliquera.
*/
import java.awt.*;
import java.applet.*;
//package à importer s'il y a de la gestion d'événements
import java.awt.event.*;

//Une fonction à nom imposé est implantée (implemented) pour définir l'action
public class ActionExample extends Applet
implements ActionListener // on prévient le compilateur
```

```

{Button okButton; // zone commune de l'applet = connu de toutes les fonctions
Button wrongButton;
TextField nameField;
CheckboxGroup radioGroup;
Checkbox radiol;
Checkbox radio2;
Checkbox radio3;

public void init() // on y crée, associe au contenant puis associe à un
// objet sensible aux actions (listener=qui écoute)
{ okButton = new Button("Action!");
wrongButton = new Button("Don't click!");
nameField = new TextField("Type here Something",35);
radioGroup = new CheckboxGroup();
radiol = new Checkbox("Red", radioGroup,false);
radio2 = new Checkbox("Blue", radioGroup,true);
radio3 = new Checkbox("Green", radioGroup,false);
add(okButton); // On relie les composants à l'applet pour qu'ils soient redessinés
add(wrongButton);
add(nameField);
add(radiol);
add(radio2);
add(radio3);

// On associe un écouteur : ce sera l'objet courant (this)
okButton.addActionListener(this);
wrongButton.addActionListener(this);
}

```

// Résultat des actions, ici on change des couleurs, dans la fonction paint() car g : contexte graphique y est transmis par le système

```

public void paint(Graphics g)
{
// Si le radiol box est sélectionné radiol.getState() retourne
// true et l'on changera (exemple) la couleur du "crayon"
if (radiol.getState()) g.setColor(Color.red);
// Sinon
else if (radio2.getState()) g.setColor(Color.blue);
// ou encore
else g.setColor(Color.green);

// On peut récupérer le texte du champ texte (TextField) et le faire apparaître dans
g.drawString(nameField.getText(),20,100);
}

```

**// Quand on clique le bouton, le système lance cette fonction
// C'est dans cette fonction qu'on spécifie les actions.**

```

public void actionPerformed(ActionEvent evt)
{
// Quel bouton est à l'origine?
if (evt.getSource() == okButton)
// On peut aussi tester le texte du bouton: evt.getActionCommand().equals(".")
//Si c'est le okButton
//On demande au système de relancer le paint() (=rafraîchir)
repaint();

// Si wrongButton
else if (evt.getSource() == wrongButton)

```

```

{
// On change le texte sur le bouton (pourquoi pas !)
wrongButton.setLabel("Pas ici!");
// On change le texte dans le TextField
nameField.setText("That was the wrong button!");
// Et on demande le rafraîchissement.
repaint();
}
}
}

```

Le bouton est pressé (à l'exécution)

CheckList de l'implantation de fonctions dites "reflexes" ou " call backs"

- Ajouter un package spécialisé : `import java.awt.event.*;`
- Choisir une classe qui sera sensible à l'action (où l'on définira la fonction ci-dessous), la déclarer comme implémentant ce rôle de récepteur: `implements ActionListener`.
- Y définir la fonction (qui sera appelée par le système, donc prototype-format imposé): `public void actionPerformed(ActionEvent evt) { }`
- Enfin, indiquer au système (en général dans le `init()` juste après la création du bouton) qui sera sensible au bouton:
`theBouton.addActionListener(objetDeLaClasseSensibilisée);`

V- Gestion des événements souris (programme qui permet le dessin d'un hexagone à la souris)

Le dessin sera réalisé dans un rectangle constituant la « toile »(du peintre, canvas en anglais), composant tel un bouton, ajouté dans le panneau-applet. On y spécifie les réactions implémentant l'écoute des mouvements de souris dans deux fonctions appelées par le système: `mouseDragged()` et `mouseMoved()`

```

import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class MyAppl extends Applet {
    MyCanvas toile;
    public void init(){Button btn;
        this.add(btn=new Button("clear")); //bouton pour repartir à zéro
        btn.addActionListener(toile); //toile sera l'objet sensible au click-bouton
        toile=new MyCanvas(); this.add(toile);
        toile.addMouseListener(toile); //toile sensible aussi à la souris
        toile.setSize(this.getSize()); //même taille pour la toile et l'applet
    }
}

class MyCanvas extends Canvas implements ActionListener, MouseMotionListener {
    int xx[]; int yy[]; int nbSommets=6; int count=-1; boolean down= false;
    public void mouseDragged(MouseEvent ev) {
        down=true;
        if (count<0) { xx=new int[nbSommets+1];
            yy=new int[nbSommets+1];
            count++;}
        xx[count]=ev.getX(); // position souris mémorisée dans le sommet courant
        yy[count]=ev.getY();
        repaint();
    }
    public void mouseMoved(MouseEvent ev){
        if (count >= nbSommets) { return;} // hexagone complet
        if (down) { down=false; count++;} // on incrémente le num du sommet courant
        //la première fois qu'on repasse a moved (déplacemnt souris levée)
    }
    public void actionPerformed(ActionEvent ev) { count=-1; repaint();}
    public void paint(Graphics g)

```

```

    { for (int i=1;i<=count;i++) //on dessine les côtés du polygone
      g.drawLine(xx[i-1],yy[i-1],xx[i%nbSommets],yy[i%nbSommets]);
    }
}

```

VI-Mise en place graphique (layout)

- Par défaut: pour une applet (< un Panel<Container<Component<Object)
this.setLayout(new FlowLayout());//placement par le système dans l'ordre des additions
- Pas de placement géré par le système:
this.setLayout(null);// pas de placement automatique, il faut préciser les positions des composants
// On doit alors spécifier la position x,y, la largeur et la hauteur de chaque composant.
okButton.setBounds(20,20,100,30);
nameField.setBounds(20,70,100,40);
radio1.setBounds(20,120,100,30);
radio2.setBounds(140,120,100,30);
option.setBounds(20,170,100,30);
- Placement géographique : automatique avec indications (suggestions) du programmeur
this.setLayout(new BorderLayout());
add(okButton1,"Center");
add(okButton2,"North");
add(okButton3,"West");
add(okButton4,"East");
add(okButton5,"South");
- Placement selon une grille avec nbrows lignes et nbcols colonnes :
this.setLayout(new GridLayout(nbrows,nbcols));
- Mise en place arborescente sophistiquée:
Placement par arborescence avec des Panels qui contiennent des panels etc...qui contiennent des composants.

VII-lire et afficher des chaînes, des valeurs etc ...

Numérique => chaîne : theVal.toString() ou bien : ""+theVal !!

Chaîne => numérique :

```

Double.valueOf(theString).doubleValue();
Integer.valueOf(theString).intValue();
Float.valueOf(theString).floatValue();
Long.valueOf(theString).longValue();
Boolean.valueOf(theString).booleanValue();

```

Parsing : Lire une suite de valeurs dans une chaîne txt séparées par un séparateur tel que espace, tab, , ;

```

for (StringTokenizer t = new StringTokenizer(txt, "\\t;"), int k=0 ; t.hasMoreTokens() ;k++)
{ String str = t.nextToken(); val[k]=Float.valueOf(str).floatValue();}

```

VIII- Objets utiles

On peut remplacer les tableaux par des "Vector" : Si on ne connaît pas à l'avance le nombre d'éléments.

```

Vector collection = new Vector();
puis

```

```
collection.addElement(new Point(x,y); // si on collectionne des points
Point pt= (Point) collection.elementAt(i); // pour récupérer le ième élément (i commence à zéro)
```

IX- Améliorations visuelles

Améliorations visuelles (en ne laissant pas faire le système !)

Le système, lorsqu'il veut repeindre appelle une fonction `update(Graphics g)`, qui elle même appelle la fonction `paint(Graphics g)` que l'on a définie soi-même. Mais, le `update()` standard; avant d'appeler `paint()` peint le composant et ses descendants avec la couleur de fond (`background Color`) => clignotement désagréable

Remède : redéfinir soi-même `update()`:

```
public void update(Graphics g) { paint(g); }
```

Pour éviter que les boutons se mélangent au dessin, définir une zone spécifique au dessin : la toile du peintre : `Canvas`

Problème: (=>lourdeur) On est souvent obligé de re-définir un `Canvas` spécialisé sous la forme d'une nouvelle classe `class CanvasPerso extends Canvas { }`, y définir une fonction `paint()` etc ...

On peut associer un `ScrollPane` à ce `Canvas` pour installer un système de vue partielle sur le canvas (avec barres de défilement ...), dans ce cas,

Le `ScrollPane` est créé dans le "container applet" et le canvas est créé dans le container `ScrollPane`
truc créé dans machin signifie: `Truc truc = new Truc() ; theMachin.add(truc);`