

# Plan de la présentation

- Les packages [java.awt](#) et [javax.swing](#)
- Les composants et composites graphiques
- Les panels
- Les zones de dessin
- Les applets

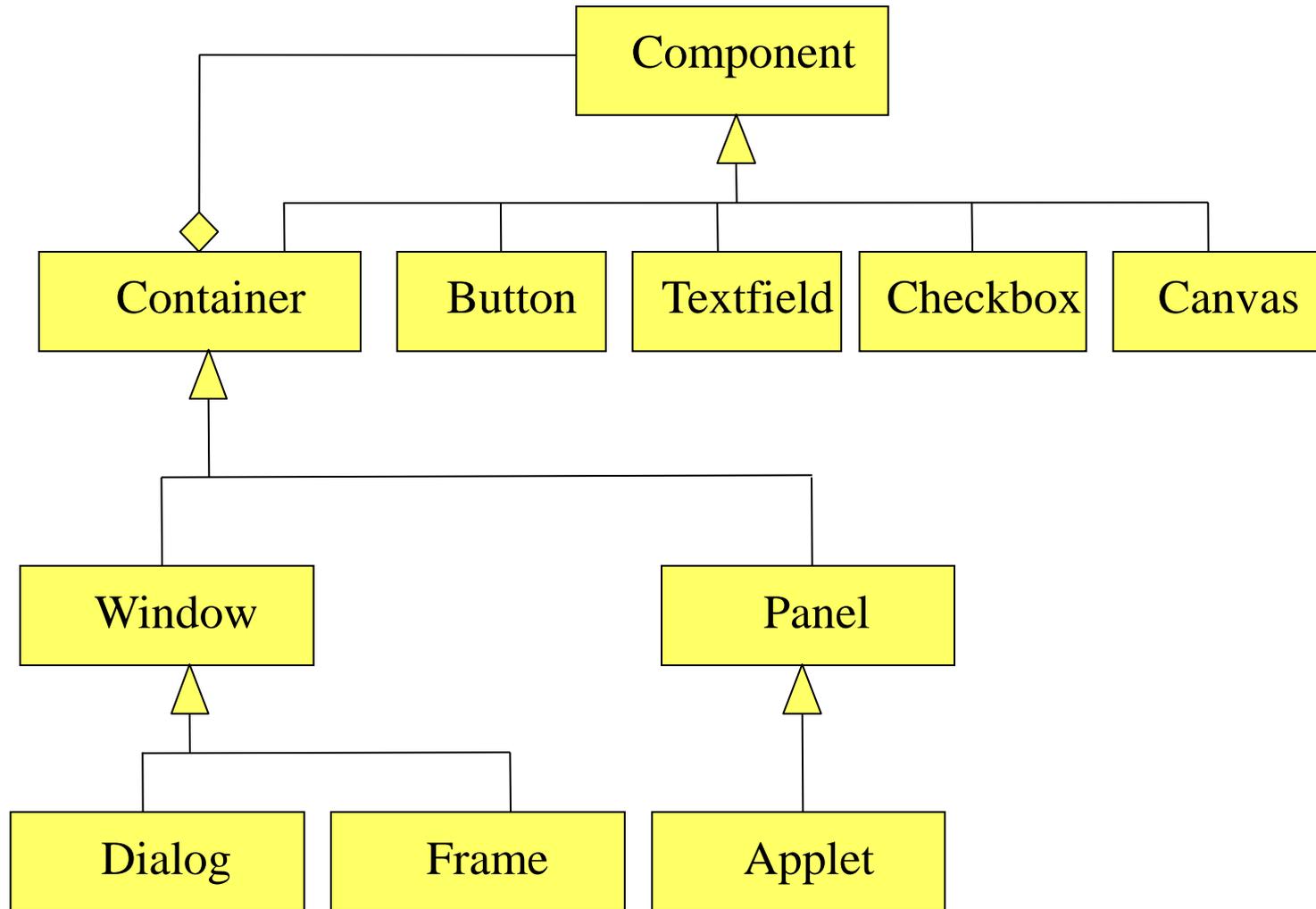
Aide Sun pour les "[api](#)" de Java

# Le framework AWT (1)

- Java propose un premier package de composants graphiques nommé awt.
- Il est composé de
  - De classes graphiques implémentées comme une arborescence (pattern Composite).
  - De classes pour placer les composants graphiques dans les composites graphiques.
  - De classes et d'interfaces de gestion d'événements implémentées (pattern Commande).

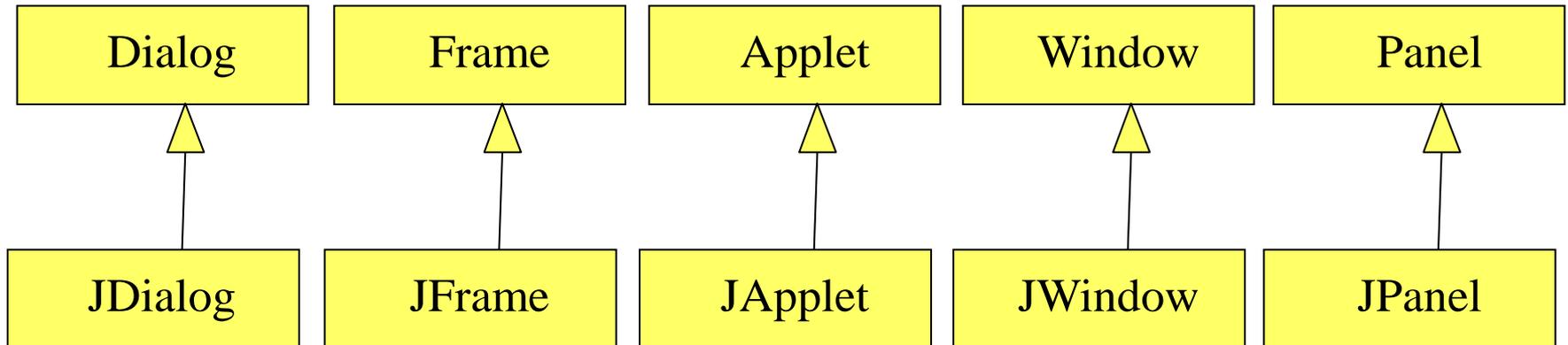


# L'arbre des composants graphiques awt



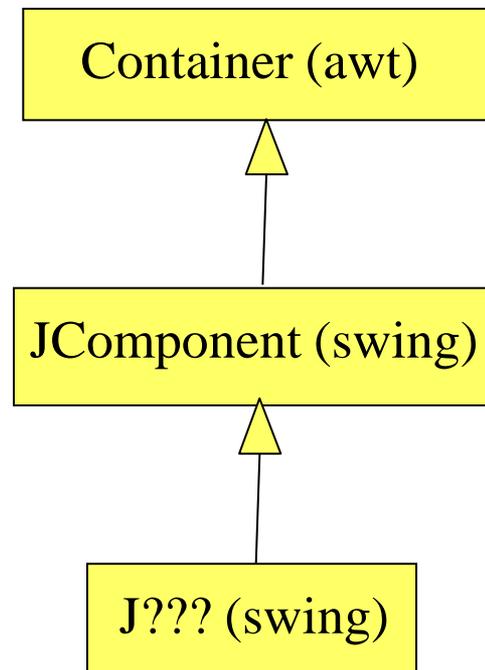
# L'arbre des composants graphiques `javax.swing`

- Les classes graphiques de type conteneurs.



# L'arbre des composants graphiques javax.swing (2)

Tous les autres composants awt ont leur équivalent swing défini comme suit :



# Les classes `Frame` et `JFrame`

- Une application graphique en Java contient une et une seule **fenêtre de fond**.
- Cette fenêtre est un objet de la classe **`Frame`** ou **`JFrame`**.
- Lors de l'exécution de l'application, cette fenêtre est la référence de l'application pour le système d'exploitation.



# Création d'une instance graphique

```
import xxx.*; // on importe le package graphique
Class AppliGraphique1 {
    public static void main(String [] args) {
        ZZZ f = new ZZZ("Le titre");
        // On place des composants dans la fenêtre de fond
        ...
        f.setSize(700,600); // on taille la fenêtre
        // On inscrit l'application auprès du gestionnaire graphique
        // du système d'exploitation. La fenêtre est affichée
        f.setVisible(true);
    }
}
xxx vaut java.awt ou javax.swing
ZZZ vaut Frame ou JFrame
```



## Création d'une instance graphique (2)

```
import xxx.*; // on importe le package graphique
Class AppliGraphique1 {
    public static void main(String [] args) {
        ZZZ f = new ZZZ("Le titre");
        // On place des composants dans la fenêtre de fond
        ...
        f.pack(); // on fabrique la fenêtre en fonction des composants
        // On inscrit l'application auprès du gestionnaire graphique
        // du système d'exploitation. La fenêtre est affichée
        f.setVisible(true);
    }
}
xxx vaut java.awt ou javax.swing
ZZZ vaut Frame ou JFrame
```



# Le thread graphique

- L'exécution de la classe donne lieu à la création d'une tâche (ou thread) gérée par l'OS.
- Le premier appel de la méthode **setVisible** par un objet Frame donne lieu à la création d'une autre tâche (dite tâche graphique) gérée par l'OS.
- Le processus lancé avec la machine virtuelle ne se termine que si les deux tâches sont terminées.



## AWT : Le thread graphique (2)

- La tâche graphique d'une application consiste pour l'OS
  - à afficher tous les éléments graphiques de l'application
  - à intercepter tous les événements utilisateurs qui se produisent dans la fenêtre de fond ou dans l'un de ces composants graphiques.



# AWT : Le thread graphique (2)

- Les événements sont :
  - Des événements Souris.
  - Des événements clavier.
  
- Le traitement des événements seront vus dans la présentation sur l'événementiel.



# AWT : Les gestionnaires de placement (1)

– Pour placer un composant graphique dans un composite graphique en Java :

- Sans gestionnaire

1. On précise que le composite n'a pas de gestionnaire  
`composite.setLayout(null);`

2. On taille le composant et on prévoit sa place à l'avance en coordonnées relatives exprimées en pixels.

`composant.setBounds(dx,dy);`

3. On ajoute le composant dans le composite  
`composite.add(composant);`



# AWT : Les gestionnaires de placement (2)

## Avec gestionnaire de placements

Il existe dans awt, des classes qui permettent d'utiliser des agencements de composants prévus à l'avance.

Il y a plusieurs classes qui dérivent de la classe `LayoutManager`. Parmi les plus connues

La classe `BorderLayout`

La classe `GridLayout`

La classe `FlowLayout`



# Le gestionnaire BorderLayout

```
Frame f = new Frame("Affichage de t...");  
String [] texte =  
{ "texte1", "texte2", "texte3", "texte4", "texte5" };  
  
f.setLayout(new BorderLayout());
```



```
f.add(new Label(texte[0],Label.CENTER),BorderLayout.NORTH);  
f.add(new Label(texte[1],Label.CENTER),BorderLayout.SOUTH);  
f.add(new Label(texte[2],Label.CENTER), BorderLayout.EAST);  
f.add(new Label(texte[3],Label.CENTER), BorderLayout.WEST);  
f.add(new Label(texte[4],Label.CENTER),BorderLayout.CENTER);
```



## Le gestionnaire GridLayout

```
Frame f = new Frame("Affichage de t...");  
String [] texte =  
{ "texte1", "texte2", "texte3", "texte4", "texte5" };
```

```
f.setLayout(new GridLayout(0,2));  
for(int i = 0; i < texte.length; i++)  
    f.add(new Label(texte[i],Label.CENTER));
```



# Le gestionnaire FlowLayout

```
Frame f = new Frame("Affichage de t...");  
String [] texte =  
{ "texte1", "texte2", "texte3", "texte4",  
  "texte5" };  
  
f.setLayout(new FlowLayout());  
for(int i = 0; i < texte.length; i++)  
    f.add(new  
        Label(texte[i],Label.CENTER));  
f.setVisible(true);
```



# Les panels

- Un objet de cette classe est un conteneur graphique
- Il sert à définir un composite particulier de l'IHM avec son propre gestionnaire de placement
- Il est lui-même un composant du composite définissant l'IHM



# Exemple d'utilisation d'un panel

```
import java.awt.*;
import javax.swing.*;

public class EssaiPanel {
public static void main(String [] args) {
JFrame jf_1 = new JFrame("Affichage de
t...");
String [] texte = { "texte1", "texte2",
"texte3", "texte4"};

jf_1.add(new JLabel("Ouest",JLabel.CENTER),
BorderLayout.WEST);
jf_1.add(new JLabel("Nord",JLabel.CENTER),
BorderLayout.NORTH);
jf_1.add(new JLabel("Sud",JLabel.CENTER),
BorderLayout.SOUTH);

jf_1.add(new
JLabel("Est",JLabel.CENTER),
BorderLayout.EAST);

JPanel p_1 = new JPanel();
jf_1.add(p_1,BorderLayout.CENTER);
p_1.setLayout(new GridLayout(2,2));
for(int i = 0; i < texte.length; i++)
p_1.add(new
JLabel(texte[i],JLabel.CENTER));

jf_1.pack();
jf_1.setVisible(true);
}
}
```



## Exemple d'utilisation d'un panel (2)



## Les zones de dessin : JPanel et Graphics

- Un objet JPanel (Canvas dans awt) est un composant graphique dans lequel on peut dessiner
- le principe de programmation est
  - Créer sa propre classe qui hérite de Canvas
  - Surcharger la méthode `public void paint(Graphics g)` de classe Canvas



## Les zones de dessin (2)

- Pour dessiner, on sert de l'objet `g` passé en paramètre à la méthode `paint`.
- Cet objet `g` est créé et attaché par la machine virtuelle au canevas lors d'un rafraichissement de la zone de dessin définie par le canevas



## Les zones de dessin (3)

- Le rafraichissement se fait quand
  1. Le programmeur force le rafraîchissement de la zone, en appelant la méthode **repaint** avec l'objet Canvas.
  2. La machine virtuelle demande le rafraîchissement pour redessiner la zone qui avait été momentanément cachée à l'écran
- **Attention : vous ne pouvez pas appeler directement la méthode **paint**.**



# Les zones de dessin (5)

```
class MonCercle extends JPanel {  
    private int x ,y;  
    private int larg, haut;  
  
    MonCercle(int p_x, int p_y, int p_larg, int p_haut) {  
        x = p_x;  
        y = p_y;  
        larg = p_larg;  
        haut = p_haut;  
    }  
  
    public void paint(Graphics g) {  
        g.setColor(new Color(255,0,0));  
        g.fillOval(x,y,larg,haut);  
    }  
}
```



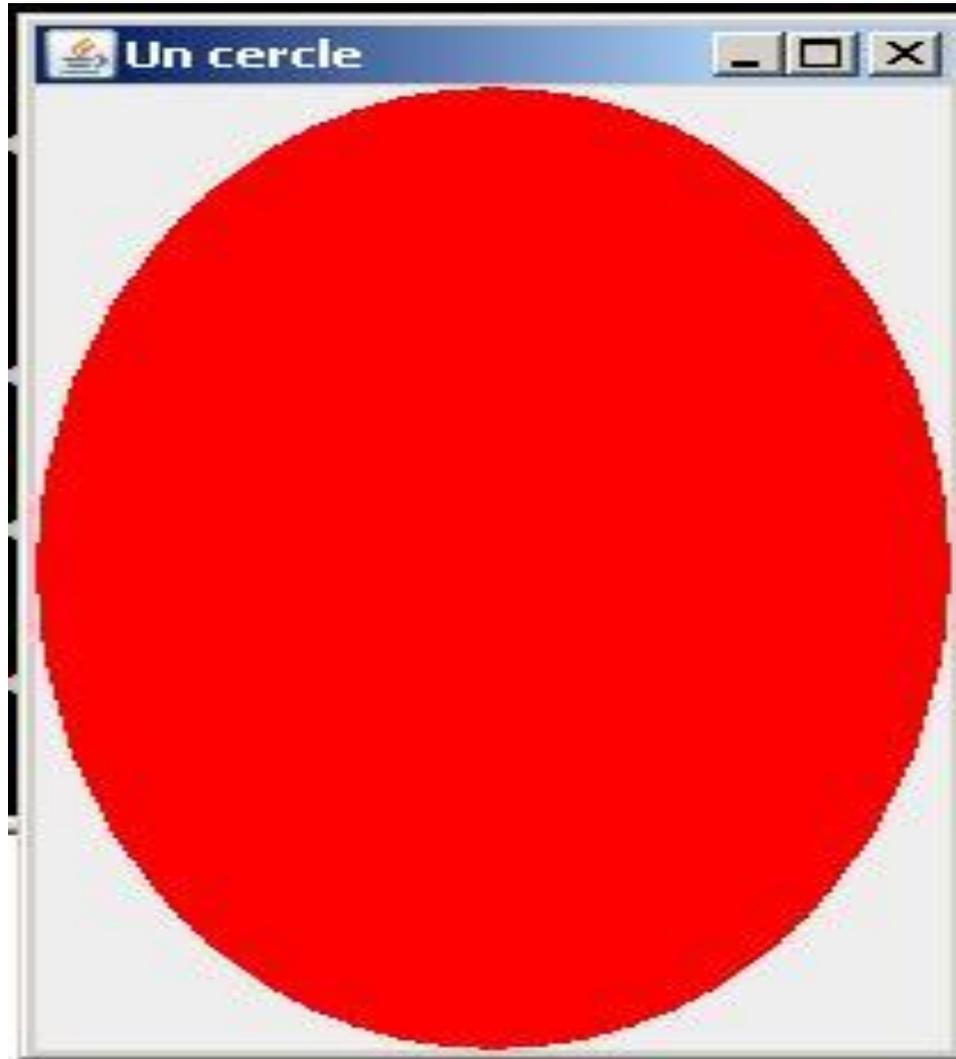
# Les zones de dessin (6)

```
/**
 * classe EssaiCanvasGraphics pour illustrer l'utilisation d'un objet
 * Canvas
 */
public class EssaiCanvasGraphics {
    public static void main(String [] args) {
        JFrame jf_1 = new JFrame("Un cercle");

        MonCercle mc = new MonCercle(0,0,200,300);
        mc.setSize(200,300);
        jf_1.add(mc);
        jf_1.pack();
        // la machine virtuelle rafraichit la zone mc
        jf_1.setVisible(true);
    }
}
```



# Les zones de dessin (7)



# Les applets

- Une applet (petite application ou appliquette) est une application Java qui tourne dans un environnement de type « browser web »
- Une applet n'a pas besoin de contenir de « main » car elle n'est pas invoquée directement par la machine virtuelle
- Le code de l'applet est exécuté par le browser web

(voir l'aide de sun pour les classes [Applet](#) et [JApplet](#))



# Les applets

`java.lang.Object`

`java.awt.Component`

`java.awt.Container`

`java.awt.Panel`

`java.applet.Applet`



# Les applets

- Pour construire une applet
  - Dans un éditeur de texte, on définit une sous classe de `java.applet.Applet`.
  - On compile le source pour obtenir le `.class`
  - Dans un éditeur de texte, on définit un fichier html qui contient l'appel de l'applet
  - On lance dans un browser le fichier html.



# Les applets

```
<html>  
  <head>  
    <title> A Simple Program </title>  
  </head>  
  <body>  
    Here is the output of my program :  
    <applet code="HelloWorld.class"  
      width="150" height="25">  
    </applet>  
  </body>  
</html>
```



# Les applets

- Le cycle de vie d'une applet
  - Juste après avoir instancier l'objet, le browser appelle sur l'objet la méthode **init**.
  - Lorsque la page contenant l'appel de l'applet est remplacée par une autre, le browser appelle avec l'applet la méthode **stop**.



# Les applets

- Le cycle de vie d'une applet
  - Réciproquement quand la page contenant l'applet réapparaît le browser appelle avec l'objet la méthode **start**.
  - Juste avant sa fermeture, le browser appelle avec l'applet la méthode **destroy**.

