

# Les applets Java

## Table des matières

Les applets Java .....	1
Création d'une applet.....	2
Le code source java .....	2
Le fichier HTML.....	3
Couleurs et fontes .....	3
Un héritage important.....	3
Les couleurs .....	4
La classe Color .....	4
Utilisation dans l'applet.....	4
Les fontes .....	4
La classe Font.....	5
Utilisation dans l'applet.....	5
Mesure de la taille d'un texte.....	5
La classe FontMetrics .....	5
Mise en application .....	6
Dessiner avec Java .....	6
La classe Graphics.....	6
Couleurs et fontes .....	6
Dessin et remplissage de formes.....	6
Un exemple .....	7
Ciel et prairie .....	7
Le soleil.....	8
Mur de la maison et porte.....	8
Toit de la maison .....	8
Exercices .....	8

# Création d'une applet

Nous allons créer une applet qui affiche "Bonjour le monde !". Cela nécessite l'écriture d'au moins deux fichiers : le fichier source java à compiler et le fichier html qui permettra d'exécuter l'applet.

[Le code source java](#)

Nous devons commencer par annoncer l'utilisation de deux packages : java.applet (pour utiliser la classe Applet) et java.awt (pour utiliser les fonctions graphiques). Cela se fait avec les deux lignes :

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

La classe que nous créons est une classe dérivée de la classe Applet. Nous le déclarons en écrivant :

```
public class bonjour extends Applet {
}
```

Notre classe va utiliser un champ nommé msg de type String qui contiendra le message à afficher. Elle surchargera aussi deux méthodes héritées de la classe Applet qui ont un rôle spécifique :

- la méthode **init** qui est appelé une seule fois au démarrage de l'applet et qui permet d'effectuer des initialisations
- la méthode **paint** qui est appelé à chaque fois qu'il est nécessaire d'afficher l'applet

On obtient le code suivant :

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

public class bonjour extends Applet {
    String msg;

    public void init() {
        msg="Bonjour de java !";
    }

    public void paint(Graphics g) {
        g.drawString(msg, 20, 20);
    }
}
```

L'affichage du texte, comme d'ailleurs tout affichage, se fait par l'intermédiaire d'une variable de type **Graphics** qui est ici fournie en paramètre dans la méthode paint. La classe Graphics, qui se trouve dans le package java.awt, fournit de nombreuses méthodes pour écrire du texte ou dessiner des figures géométriques. Nous allons l'explorer dans la suite de ce TD.

## Le fichier HTML

Il suffit de créer un fichier HTML minimal contenant la balise **<APPLET>**. Cette balise devra préciser au moins trois attributs :

- **CODE** avec le nom de la classe contenant l'applet
- **WIDTH** avec la largeur de la zone réservée à l'applet
- **HEIGHT** avec la hauteur de la zone réservée à l'applet

Cela donne, par exemple, le fichier suivant :

```
<HTML>
<HEAD>
<TITLE>Applet bonjour</TITLE>
</HEAD>
<BODY BGCOLOR=white>
<H1>Bonjour</H1>
<HR>
<DIV ALIGN=center>
<APPLET CODE="bonjour.class" WIDTH="300" HEIGHT="100">
</APPLET>
</DIV>
<HR>
</BODY>
</HTML>
```

Enregistrer ce fichier en le nommant **bonjour.html**. On peut alors l'utiliser directement avec un navigateur. Il est aussi possible de passer par l'appletviewer qui se contentera d'afficher l'applet. Il faut alors entrer la commande : **appletviewer bonjour.html** dans une console DOS.

## Couleurs et fontes

Nous allons reprendre l'applet bonjour et la compléter pour définir une couleur de fond, une couleur du texte et la fonte utilisée pour le texte.

## Un héritage important

En consultant la documentation du JDK, on voit que la classe Applet hérite des champs et méthodes de nombreuses classes.

```
Class java.applet.Applet
|
+----java.lang.Object
|
+----java.awt.Component
|
+----java.awt.Container
|
+----java.awt.Panel
|
+----java.applet.Applet
```

Intéressons nous à la classe `Component` qui est l'ancêtre commun à toute classe représentant un élément à afficher. Elle fournit un certain nombre de méthodes importantes dont la classe `Applet` hérite et que l'on peut donc utiliser.

On trouve, parmi d'autres :

1. **Affichage** : méthodes `paint` et `repaint`
2. **Couleur du fond** : méthodes `getBackground` et `setBackground`
3. **Couleur de dessin par défaut** : méthodes `getForeground` et `setForeground`
4. **Dimensions** : méthodes `getSize` et `setSize`
5. **Fonte pour le texte** : méthodes `getFont` et `setFont`
6. **Mesure de la taille du texte** : méthode `getFontMetrics`

Il est nécessaire de se familiariser avec ces méthodes que nous retrouverons souvent.

## Les couleurs

### La classe `Color`

C'est la classe **`Color`** (du package `java.awt`) qui permet de gérer les couleurs. Celles-ci sont définies, dans un constructeur, par leurs trois composantes RVB (Rouge, Vert, Bleu). Ainsi pour utiliser la couleur rouge, on pourra passer par une variable `c_rouge` en écrivant :

```
Color c_rouge=new Color(255,0,0);
```

L'applet [RVB](#) vous permet de voir évoluer les couleurs en fonction des composantes RVB.

La classe `Color` fournit quelques constantes représentant les couleurs de base : `Color.black`, `Color.blue`, `Color.cyan`, `Color.darkGray`, `Color.gray`, `Color.green`, `Color.lightGray`, `Color.magenta`, `Color.orange`, `Color.pink`, `Color.red`, `Color.white` et `Color.yellow`.

### Utilisation dans l'applet

Reprenons l'applet `bonjour` et choisissons une couleur de fond et une couleur de dessin. Cela peut se faire dans la méthode `init`.

```
public void init() {  
    msg="Bonjour de java !";  
    String parm=getParameter("nom");  
    if (parm!=null) msg=parm+" Java te dit bonjour !";  
    //on définit les couleurs utilisées  
    setBackground(Color.black);  
    setForeground(Color.yellow);  
}
```

On obtient le résultat suivant :



## Les fontes

## La classe Font

C'est la classe Font du package java.awt qui permet de gérer les fontes. Elle possède un constructeur qui utilise 3 paramètres : le nom de la fonte, son style (normal, italique, gras) et sa taille.

Le nom est une chaîne de caractères.

Le style est un nombre entier qui peut prendre les valeurs prédéfinies Font.PLAIN (style normal), Font.ITALIC (style italique) et Font.BOLD (style gras).

La taille est un nombre entier.

Les noms de fontes utilisables peuvent être obtenus dans un tableau de chaînes de caractères par l'instruction suivante :

```
String[] fontes=Toolkit.getDefaultToolkit().getFontList();
```

En général on peut utiliser : Dialog, SansSerif, Serif, Monospaced, Helvetica, TimesRoman, Courier, DialogInput, ZapfDingbats.

## Utilisation dans l'applet

Reprenons l'applet bonjour et choisissons une fonte pour l'affichage du texte.

```
public void init() {  
    msg="Bonjour de java !";  
    String parm=getParameter("nom");  
    if (parm!=null) msg=parm+" Java te dit bonjour !";  
    //on définit les couleurs utilisées  
    setBackground(Color.black);  
    setForeground(Color.yellow);  
    //on choisit une fonte  
    setFont(new Font("TimesRoman", Font.BOLD+Font.ITALIC, 20));  
}
```

Remarque : les constantes de style peuvent être additionnées pour combiner deux styles.

On obtient le résultat suivant :



## Mesure de la taille d'un texte

### La classe FontMetrics

Pour positionner correctement le texte à afficher, il est nécessaire de connaître ses dimensions, c'est à dire sa largeur et hauteur en pixels. Evidemment ces dimensions vont dépendre de la longueur du texte, mais aussi de la fonte choisie et de la taille de la fonte. La classe

**FontMetrics** permet d'obtenir ces résultats. Elle propose les méthodes suivantes :

- **getAscent()** : détermine le dépassement au dessus de la ligne de base.
- **getDescent()** : détermine le dépassement en dessous de la ligne de base
- **getHeight()** : détermine la hauteur totale
- **stringWidth(String)** : détermine la largeur d'une chaîne

Le constructeur de la classe `FontMetrics` étant déclaré `protected`, il n'est pas possible de créer directement une variable de ce type. Il faudra passer par une méthode prévue à cet effet, par exemple la méthode statique `getFontMetrics`.

#### Mise en application

Reprenons une dernière fois l'applet `bonjour` pour qu'elle affiche son message en le centrant horizontalement. Il nous faudra calculer l'abscisse de la position d'affichage en appliquant la formule :

$(\text{largeur\_totale} - \text{largeur\_texte}) / 2$ .

Nous effectuerons ce calcul dans la méthode `paint`.

```
public void paint(Graphics g) {
    FontMetrics fm=getFontMetrics(getFont());
    //getSize().width donne la largeur totale
    int x=(getSize().width-fm.stringWidth(msg))/2;
    g.drawString(msg, x, 20);
}
```

## Dessiner avec Java

Tout affichage (texte ou dessin) se fait en utilisant la classe `Graphics` qui fournit de nombreuses méthodes.

### La classe `Graphics`

Le constructeur de la classe `Graphics` est déclaré `protected`. On ne peut donc pas créer directement des variables de type `Graphics`; on les obtient avec des méthodes spécifiques comme **`getGraphics`** dans la classe `Component` ou comme paramètre de la méthode **`paint`**.

#### Couleurs et fontes

Lorsqu'on utilise le paramètre de type `Graphics` fourni par la méthode `paint`, le composant lui attribue une couleur de dessin (définie par `setForeground`) et une fonte par défaut (définie par `setFont`). Ces valeurs peuvent être modifiées par des méthodes de la classe `Graphics`.

- **`setColor(Color)`** permet de fixer une nouvelle couleur de dessin
- **`setFont(Font)`** permet de choisir une nouvelle fonte.

D'autre part, on peut obtenir une variable de type `FontMetrics` correspondant à la fonte en cours en utilisant la méthode **`getFontMetrics()`**.

L'affichage de texte se fait avec la méthode **`drawString(String, int, int)`** déjà rencontrée.

#### Dessin et remplissage de formes

La classe `Graphics` permet de dessiner et de remplir des figures ayant des formes élémentaires. La méthode de dessin est préfixée par "draw", la méthode de remplissage est préfixée par "fill".

- **Segments et suites de segments**  
*drawLine(int, int, int, int)* permet de tracer un segment; les 4 paramètres sont les coordonnées des extrémités.  
 Pour une suite de segments, on pourra utiliser *drawPolyline(int[], int[], int)* les tableaux d'entiers passés en paramètre contiennent les coordonnées des points à relier; le 3ème paramètre est le nombre de points; si le premier et le dernier point sont confondus, on retrouve *drawPolygon*.
- **Rectangles**  
*drawRect(int, int, int, int)* et *fillRect(int, int, int, int)*  
 pour dessiner et remplir des rectangles aux côtés verticaux et horizontaux; les deux premiers paramètres représentent les coordonnées du coin supérieur gauche, les deux autres représentent la largeur et la hauteur.  
 On peut obtenir des rectangles à coins arrondis avec les méthodes *drawRoundRect(int, int, int, int, int, int)* et *fillRoundRect(int, int, int, int, int, int)*; dans ce cas les deux derniers paramètres définissent la façon de faire les arrondis.  
 On peut aussi dessiner des rectangles avec effet 3D (on n'utilise pas la même couleur pour les 4 côtés) en utilisant *draw3DRect(int, int, int, int, boolean)* et *fill3DRect(int, int, int, int, boolean)*; le 5ème paramètre de type boolean permet de choisir entre deux effets : en relief ou en creux.
- **Polygones**  
*drawPolygon(int[], int[], int)* et *fillPolygon(int[], int[], int)*  
 pour dessiner et remplir des polygones; les deux tableaux d'entiers contiennent les coordonnées des sommets, le 3ème paramètre contient le nombre de sommets.  
 On peut aussi utiliser la classe *Polygon* et les méthodes *drawPolygon(Polygon)* et *fillPolygon(Polygon)*
- **Ellipses et cercles**  
*drawOval(int, int, int, int)* et *fillOval(int, int, int, int)*  
 pour dessiner et remplir des ellipses inscrites dans le rectangle dont les caractéristiques sont passées en paramètre; pour obtenir un cercle, il suffit de prendre la largeur égale à la hauteur, donc les deux derniers paramètres égaux.
- **Arcs**  
*drawArc(int, int, int, int, int, int)* et *fillArc(int, int, int, int, int, int)*  
 pour dessiner des arcs et remplir des secteurs circulaires; les 4 premiers paramètres définissent l'ellipse utilisée, les deux derniers les angles de début et de fin de l'arc.

Ce petit tour d'horizon n'épuise pas toutes les possibilités offertes par la classe *Graphics*.

## Un exemple

Nous allons dessiner un petit paysage (prairie, ciel, maison, soleil) dans un rectangle de 300 pixels sur 200.

Il suffira d'écrire la méthode *paint* de l'applet. Tout est une affaire de coordonnées.

Ciel et prairie

On remplit deux rectangles en bleu et en vert.

```
//ciel
g.setColor(Color.blue);
g.fillRect(0,0,300,110);
```

```
//prairie
g.setColor(Color.green);
g.fillRect(0,110,300,90);
```

### Le soleil

On se contentera d'un cercle jaune.

```
//soleil
g.setColor(Color.yellow);
g.fillOval(220,20,30,30);
```

### Mur de la maison et porte

Un rectangle blanc, un contour en noir et quelques traits pour la porte.

```
//mur de la maison
g.setColor(Color.white);
g.fillRect(80,100,50,70);
g.setColor(Color.black);
g.drawRect(80,100,50,70);
//porte
g.drawRect(90,140,20,30);
g.drawLine(110,155,105,155);
```

### Toit de la maison

On se contentera d'un triangle rouge.

```
//toit en triangle
int[] x={80,130,105};
int[] y={100,100,50};
g.setColor(Color.red);
g.fillPolygon(x,y,3);
g.setColor(Color.black);
g.drawPolygon(x,y,3);
```

## Exercices

### 1. Texte en relief

Affichage d'un texte en relief : il suffit d'afficher trois fois le même texte en changeant de couleur et en introduisant un petit décalage.

### 2. Texte encadré

Créer une applet qui affiche un texte au centre de la zone qui lui est réservée et qui l'encadre.

### 3. Texte encadré avec effet 3D

Créer une applet qui affiche un texte au centre de la zone qui lui est réservée et qui l'encadre en utilisant l'effet 3D de la méthode draw3DRect. (utiliser deux rectangles pour l'encadrement)

### 4. Texte paramétré

Créer une applet qui affiche un texte donné en paramètre "texte" avec une fonte donnée en paramètre "fonte" et une taille donnée en paramètre "taille". Les 3 paramètres seront définis dans le fichier html associé à l'applet.



5. **20 carrés**

Créer une applet qui affiche 20 carrés dont la position, la taille et la couleur sont tirées au hasard.

6. **Cible**

Créer une applet qui dessine une cible au centre de la zone qui lui est réservée. (faire une série de cercles pleins)

7. **Histogramme**

Les résultats d'un sondage sont :

- OUI 37%
- NON 52%
- Sans opinion 11%

Représenter ces résultats par un histogramme, puis par un camembert.