Double buffering dans une applet

1 Le problème

Lors de l'appel à la méthode `paint` d'une applet, tous les appels aux fonctions de dessin se font directement à l'écran. Lorsque les objets dessinés deviennent nombreux, un phénomène de saccade apparait.

2 La solution

2.1 Définition du double buffering

L'idée est simple : au lieu de dessiner directement à l'écran, on dessine dans une image (non affichée), et lorsqu'on a fini de dessiner, cette image est affichée à l'écran. L'image est en général appelée *backbuffer*. Le nom de double buffering vient de l'utilisation de deux objets de dessins.

2.2 Mise en place

Seule la classe d'applet est à modifier, en quatre étapes :
- création des objets pour le backbuffer
- initialisation de ces objets
- modification de la méthode `paint` pour prendre en compte le backbuffer
- ajout de la méthode `update`

2.2.1 Création des objets pour le backbuffer

Il faut ajouter deux attributs à la classe applet : un Graphics et une `Image`. Par exemple :

```
import java.awt.Image;
import java.awt.Graphics;
// ...

class MonApplet
{
    private Image backBuffer;
    private Graphics backBufferGraphics;
}
```

2.2.2 Initialisation des objets pour le backbuffer

Les nouveaux attributs doivent être initialisés dans la méthode `init` :

```
class MonApplet
{
    public void init()
    {
        backBuffer = createImage(getWidth(), getHeight());
        backBufferGraphics = backBuffer.getGraphics();
    }
}
```

2.2.3 Modification de la méthode `paint`

Au lieu de dessiner sur l'objet `Graphics g` passé en paramètre, on dessinera sur l'objet `Graphics` attaché au backbuffer. On finira la méthode `paint` par l'appel à :

```
g.drawImage(backbuffer, 0, 0, this);
```

2.2.4 Méthode `update`

Il faut ensuite ajouter à l'applet une méthode publique appelée `update` :

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