this Vs. *static*

Introduction

Si vous avez essayé d'écrire d'importants programmes Java *multi-classes*, vous avez certainement eu des problème de *communication* entre plusieurs classes. Pour appeler à partir d'une classe B une méthode ou un objet (une variable…) d'une classe A, cette méthode, objet, ou variable doit être déclaré *statique*. La notion de variables, objets ou méthodes statiques a déjà été abordée dans les chapitres précédents. Dans ce chapitre, nous allons expliquer comment utiliser le *pointeur* **this** dans une simulation de contexte statique.

Note : « une simulation de contexte statique » paraîtra ignominieux pour les puristes, mais, vous allez le constater, je trouve que l'expression exprime assez bien le procédé décrit dans ce chapitre.

Note 2 : si vous lisez en parallèle de ce cours des livres spécialisés dans l'apprentissage de Java (comme Le Programmeur, Java 2 de Roger Cadenhead & Laura Lemay, ed. Campus Press), vous remarquerez certainement une remarque à l'attention des programmeurs C/C++ disant que les *pointeurs* sont inexistants en Java. C'est vrai, à l'exception de **this**.

Utilisation de **this**

Avant d'en arriver au cœur du problème (que je n'ai pas encore exposé), le pointeur **this** et le mot clé **static** doivent être présentés.
this signifie en anglais « ce » ou « cet », autrement dit, « cet objet, ce composants là et pas un autre ». Par exemple, imaginez une variable de classe portant le même nom qu'une variable déclarée comme argument d'une méthode. Si on utilise ce nom de variable au sein de la méthode en question, la référence sera faite à la variable de classe. Comment alors faire référence à cette variable déclarée en argument de la méthode ? En utilisant le pointeur **this** !
Examinez l'exemple suivant :

```
class EssaiPointeur {
String nom ;
public void maMéthode(String nom) {
nom = "Bonjour" ; // fait référence à la variable String de classe nom
this.nom = "Au revoir" ; // fait référence à la variable locale nom
}
}
```

Mais **this** est rarement employé ainsi : allez-vous créer 2 variables différentes et pourtant de même nom ? On utilise par contre très souvent **this** dans la création d'une interface graphique, pour faire référence à la fenêtre en cours. Par exemple, si une classe *étend* de JFrame, pour définir le titre de la fenêtre ('Caption' pour les connaisseurs de Delphi), on écrira : **this.setTitle("Ma fenêtre") ;**
Voici un cadre créé avec JBuilder :

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

public class MonCadre extends JFrame {
  JButton jButton1 = new JButton();
  FlowLayout flowLayout1 = new FlowLayout();

  public MonCadre() {
    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  private void jbInit() throws Exception {
```

```
    jButton1.setText("jButton1");
    this.getContentPane().setLayout(flowLayout1);
    this.setTitle("Ma fenêtre !");
    this.getContentPane().add(jButton1, null);
  }
}
```

Utilisation de **static**

L'utilisation de **static** a déjà été traitée dans les chapitres précédents.
Ce mot clé sert à rendre les méthodes et variables d'une classe accessibles à partir d'autres classes. Par exemple, examinez le code suivant :

```
class EssaiStatic {
  String nonStatique;
  static String statique;
}


class LeTest {
  public static void main(String args[]) {
    EssaiStatic.statique = "Bonjour";
    System.out.println(EssaiStatic.statique);
    /*Le code suivant provoquerait une erreur de compilation:
    EssaiStatic.nonStatique = "Bonjour";
    System.out.println(EssaiStatic.nonStatique);
    */
  }
}
```

Voici le message d'erreur que le compilateur indique si on essaie de compiler le code mis en remarque ci-dessus :

```
0: Can't make a static reference to nonstatic variable nonStatique in class EssaiStatic.
EssaiStatic.nonStatique = "Bonjour";
```

Sinon, à l'exécution, le texte « Bonjour » est bien affiché.

Notre problématique

Nous en arrivons enfin à la problématique de ce chapitre : je veux référencer **this** depuis un contexte statique, par exemple pour changer le titre de ma fenêtre principale depuis une autre fenêtre.
Pour faire ceci, on va essayer d'écrire un code du genre :

```
// Dans ma classe principale
class MaClasse {
public static void changerTitre(String titre) {
    this.setTitle(titre) ;
}
}

// Dans ma classe secondaire
MaClasse.changerTitre("Mon Nouveau titre !") ;
```

Et ce code ne fonctionne pas ! Le compilateur indique que **this** ne peut être référence depuis un contexte statique. Vous pouvez toujours essayer des combinaisons de méthodes imbriquées faisant appel les unes aux autres pour arriver à référencer **this**, vous n'y arriverez jamais.
Alors, comment changer le titre de la fenêtre principale depuis une autre classe ?
On peut le faire en 2 étapes : créer une variable statique à laquelle on assigne un texte, définir le texte de cette variable comme titre de notre fenêtre depuis une méthode, non statique, appelée par (par exemple) un gestionnaire d'événement. Si vous ne voyer pas vraiment comment cela peut fonctionner, ne vous inquiétez pas, et laissez tomber cette idée. Pour information, ce procédé est utilisé pour certaines fonctions de JTE2 (disponible

dans la section Programmes du site JGFL, http://perso.wanadoo.fr/guillaume/), grâce au bouton *Rafraîchir* de la barre d'icônes.

Mais ce procédé n'est vraiment pas sympathique et il peut être remplacé facilement :

- Créer une classe A
- Créer une instance de cette classe A dans une classe B
- On a ainsi accès à toutes les méthodes statiques ou non de la classe A dans la classe B, même si elles utilisent le pointeur **this**.

Voici un exemple complet :

- Classe Lancer.class = lancer le programme, possède la méthode main()
- Classe Cadre1.class = la première fenêtre
- Classe Cadre2.class = la seconde fenêtre

```
package essaiStatic;

import javax.swing.UIManager;

public class Lancer {
  boolean packFrame = false;

  //Construire l'application
  public Lancer() {
    Cadre1 frame = new Cadre1();

    //Valider les cadres ayant des tailles prédéfinies
    //Compacter les cadres ayant des infos de taille préférées - ex. depuis leur
disposition
    if (packFrame) {
      frame.pack();
    }
    else {
      frame.validate();
    }
    frame.setVisible(true);
  }

  //Méthode principale
  public static void main(String[] args) {
    new Lancer();
  }
}
```

```
 package essaiStatic;

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

public class Cadre1 extends JFrame {
  FlowLayout flowLayout1 = new FlowLayout();
  JButton jButton1 = new JButton();

  public Cadre1() {

    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  private void jbInit() throws Exception {
    this.setSize(400, 60);
    jButton1.setText("Lancer");
    jButton1.addActionListener(new Cadre1_jButton1_actionAdapter(this));
```

```
    this.getContentPane().setLayout(flowLayout1);
    this.getContentPane().add(jButton1, null);
    this.setVisible(true);
    this.show();

  }

  void jButton1_actionPerformed(ActionEvent e) {
       Cadre2 cadre = new Cadre2(this);
  }
}

class Cadre1_jButton1_actionAdapter implements java.awt.event.ActionListener {
  Cadre1 adaptee;

  Cadre1_jButton1_actionAdapter(Cadre1 adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jButton1_actionPerformed(e);
  }
}
```

```
package essaiStatic;

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

public class Cadre2 extends JFrame {
  JPanel jPanel1 = new JPanel();
  BorderLayout borderLayout1 = new BorderLayout();
  JLabel jLabel1 = new JLabel();
  JButton jButton1 = new JButton();

  /* ============ Créer une instance de Cadre1 ==================== */
  Cadre1 parent;

  /* === Le constructeur doit avoir comme argument une instance de Cadre1 === */
  public Cadre2(Cadre1 monCadre) {

    /* ======== Variable de classe a la même valeur que la locale ======== */
    parent = monCadre;

    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  private void jbInit() throws Exception {
    this.setSize(400, 70);
    jPanel1.setLayout(borderLayout1);
    jLabel1.setText("Cliquez ici pour changer le titre de la première fenêtre:");
    jButton1.setText("CLIC !");
    jButton1.addActionListener(new Cadre2_jButton1_actionAdapter(this));
    this.setResizable(false);
    this.getContentPane().add(jPanel1, BorderLayout.CENTER);
    jPanel1.add(jLabel1, BorderLayout.CENTER);
    jPanel1.add(jButton1, BorderLayout.SOUTH);
    this.setVisible(true);
    this.show();
  }

  void jButton1_actionPerformed(ActionEvent e) {
       parent.setTitle("Vous voyez bien que ça fonctionne !!!");
  }
```

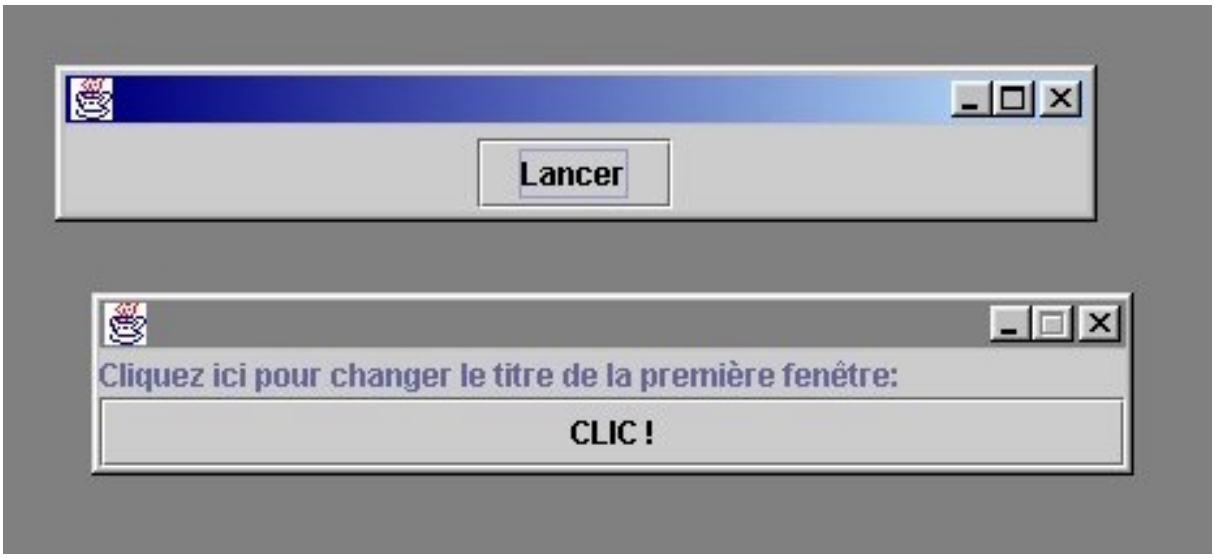
```
}

class Cadre2_jButton1_actionAdapter implements java.awt.event.ActionListener {
  Cadre2 adaptee;

  Cadre2_jButton1_actionAdapter(Cadre2 adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jButton1_actionPerformed(e);
  }
}
```

Prenez votre temps pour examiner cette application. Regardez 2 choses plus particulièrement :

- L'appel dans **Cadre1** au constructeur de **Cadre2** qui admet comme argument un objet de type **Cadre1** (exactement comme si il s'agissait d'un objet de type String)
- La variable de classe *parent* de type **Cadre1** déclarée dans **Cadre2**
- La variable locale *monCadre* dans **Cadre2** (qui est de type **Cadre1)** et qui est l'image de **this** dans l'appel du constructeur de **Cadre2** (appel fait dans **Cadre1**).
- L'utilisation de l'instruction **parent.setTitle()** dans **Cadre2** comme s'il s'agissait de **this.setTitle()** dans **Cadre1**.

Vous pouvez télécharger le source du programme dans le rubrique Exemples du site JGFL.
Voici le programme à l'œuvre, en 3 étapes :


Lancer


Lancer
Cliquez ici pour changer le titre de la première fenêtre:
CLIC !


Vous voyez bien que ça fonctionne !!!
Lancer
Cliquez ici pour changer le titre de la première fenêtre:
CLIC !