

## TECHNOLOGIE JAVA WEB START

### Java Web Start et les applets

*Ce qui a fait le grand succès de Java, ce sont les applets. En effet, au moyen des applets, nous disposons de l'équivalent de tout ce que nous trouvons dans des applications (des menus, des boutons, des images de fond, des zones d'éditions, des combobox, etc.), et ceci au sein d'une page Web.*

*De ce fait, et de la même manière, vous n'avez besoin de rien côté client. Il suffit juste de disposer d'un navigateur. Vous retrouvez sur votre poste client, un programme relativement sophistiqué au milieu d'une page Web. La différence entre une application classique et une applet est juste le conteneur. Effectivement, dans un cas, nous avons une fenêtre cadre (JFrame), et de l'autre, nous avons une page Web qui gère cette applet (par le truchement de JApplet).*

### Que fait Java Web Start dans ce contexte là puisqu'il est presque similaire aux applets ?

*Nous allons donc recenser quelques petites différences entre Java Web Start et les applets :*

1. *Java Web Start permet de délivrer des applications Java ordinaires, qui sont démarrées par un appel à la méthode main() d'une classe. Il n'est pas nécessaire d'hériter de JApplet.*
2. *Une application Java Web Start ne demeure pas dans un navigateur. Elle s'affiche en dehors de lui.*
3. *Une application Java Web Start peut être lancée depuis le navigateur, mais le mécanisme sous-jacent est assez différent de celui d'une applet. En effet, le navigateur lance simplement une application externe lorsqu'il charge un descripteur d'application Java Web Start. Le mécanisme ressemble à celui utilisé pour lancer d'autres applications d'aide comme Adobe Acrobat ou RealAudio. Ainsi, les fournisseurs de navigateurs, qui se montrent plutôt réticents avec les applets, ne pourront pas interférer avec ce mécanisme.*
4. *Une fois qu'une application Java Web Start a été téléchargée, elle peut être démarrée ensuite en dehors du navigateur.*

*Attention, Java Web Start, comme pour l'applet, possède une "bac à sable" qui empêche d'accéder aux ressources locales pour les applications non signées.*

§

### Déploiement des applications avec Java Web Start

*Vu ce que nous venons déjà de découvrir, je vous propose de mieux connaître le rôle de Java Web Start, de recenser ces principales caractéristiques, et de voir également comment déployer une application Java.*

### Principales caractéristiques de Java Web Start

*Java Web Start est une technologie relativement récente destinée à livrer, en Intranet et sur Internet, des applications qui présentent les caractéristiques suivantes :*

5. *Elles sont généralement délivrées par un navigateur. Une fois qu'elles ont été téléchargées, elles peuvent être démarrées sans navigateur.*
6. *Elles ne demeurent pas dans un navigateur. Elles s'affichent en dehors de lui.*
7. *Elles n'utilisent pas l'implémentation Java du navigateur. Le navigateur lance simplement une application externe lorsqu'il charge un descripteur d'application Java Web Start. Le mécanisme ressemble à celui utilisé pour lancer d'autres applications d'aide comme Adobe Acrobat Reader.*
8. *Les applications signées numériquement peuvent recevoir des droits d'accès arbitraires sur la machine locale. Des applications non signées s'exécutent dans un "bac à sable" qui interdit les opérations potentiellement dangereuses.*

## Comment déployer une application Java sur Internet

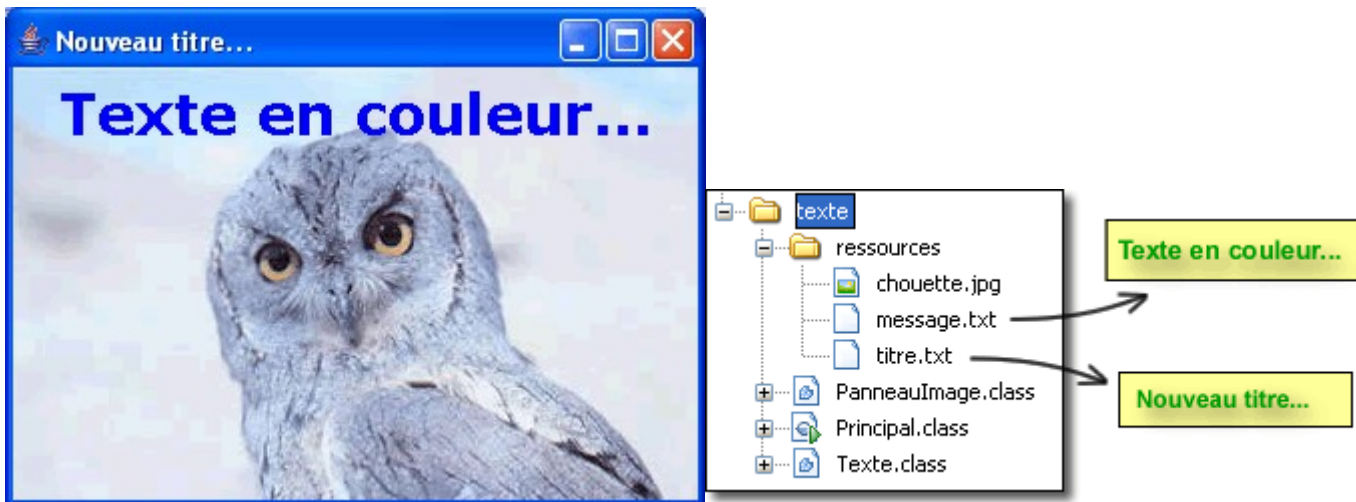
Pour préparer une application en vue d'une livraison par Java Web Start, vous devez :

9. Packager l'application dans un ou plusieurs fichiers JAR.
10. Préparer ensuite un fichier de description au format Java Network Launch Protocol (JNLP).
11. Après cela, placez ces différents fichiers dans un serveur Web.
12. Vous devez vous assurer que votre serveur Web rapporte un type MIME application/x-java-jnlp-file pour les fichiers ayant une extension <.jnlp>

### Application qui va servir au déploiement

Cette application comporte plusieurs classes ainsi qu'une image à faire afficher en papier peint. Par ailleurs le texte qui s'affiche est sensible au mouvement du curseur de la souris, puisque lorsque ce dernier se déplace sur le texte, celui-ci change alors de couleur. Le texte reprend ensuite sa couleur d'origine lorsque le curseur de la souris s'en va en dehors de la zone de texte.

Cette application comporte en plus plusieurs ressources : le titre de la fenêtre, le message d'invite, ainsi que l'image de fond. Voici donc la vue définitive avec l'architecture des différents répertoires, suivi pour terminer du code source correspondant :



#### Principal.java

```
package texte;

import java.awt.event.*;
import java.io.*;
import java.util.Scanner;
import javax.imageio.ImageIO;
import javax.swing.*;
import java.awt.*;

public class Principal extends JFrame {
    public Principal() throws IOException {
        this.setDefaultCloseOperation(this.EXIT_ON_CLOSE);
        this.setSize(350, 250);
        Scanner titre = new Scanner(this.getClass().getResourceAsStream("ressources/titre.txt"));
        this.setTitle(titre.nextLine());
        Image image = ImageIO.read(this.getClass().getResource("ressources/chouette.jpg"));
        PanneauImage panneau = new PanneauImage(image);
        this.getContentPane().add(panneau);
    }
}
```

```

    public static void main(String[] args) throws IOException {
        new Principal().setVisible(true);
    }
}

class PanneauImage extends JPanel {
    private Image image;
    private Texte invite = new Texte("C'est chouette...");
    public PanneauImage(Image image) {
        Scanner message = new
Scanner(this.getClass().getResourceAsStream("ressources/message.txt"));
        invite = new Texte(message.nextLine());
        this.image = image;
        invite.setCouleurSurvol(Color.red);
        invite.setCouleurNormale(Color.blue);
        this.add(invite);
    }
    protected void paintComponent(Graphics g) {
        g.drawImage(image, 0, 0, this);
    }
}

class Texte extends JLabel implements MouseListener {
    private Color couleurSurvol, couleurNormale;
    public Texte(String invite) {
        super(invite);
        this.setFont(new Font("Verdana", Font.BOLD, 28));
        this.addMouseListener(this);
    }
    public void setCouleurSurvol(Color couleur) {
        couleurSurvol = couleur;
    }
    public void setCouleurNormale(Color couleur) {
        this.setForeground(couleurNormale = couleur);
    }
    public void mouseEntered(MouseEvent e) {
        this.setForeground(couleurSurvol);
    }
    public void mouseExited(MouseEvent e) {
        this.setForeground(couleurNormale);
    }
    public void mouseClicked(MouseEvent e) {}
    public void mousePressed(MouseEvent e) {}
    public void mouseReleased(MouseEvent e) {}
}

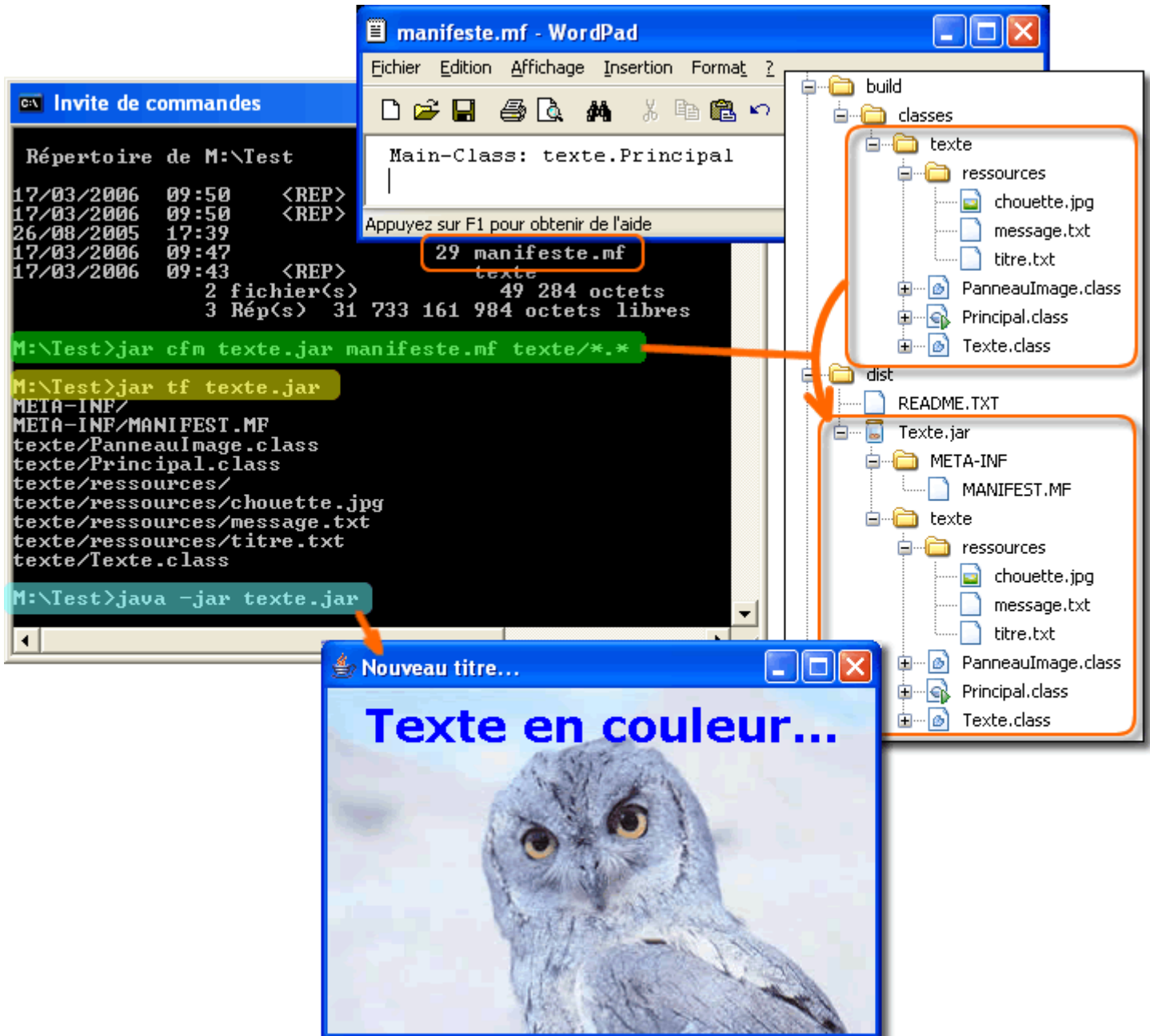
```

### Packaging de l'application texte.jar

*D'après la procédure, nous devons donc archiver (packager) l'ensemble de cette application afin d'obtenir un seul fichier qui plus est compressé. Cette démarche paraît indispensable afin d'éviter Pd'avoir un temps de chargement trop important. En effet, il faut bien comprendre que ce temps de chargement n'est pas dû à la taille des fichiers classes, relativement petits, mais à la surcharge considérable découlant d'une connexion avec un serveur Web avec plusieurs requêtes demandées pour chacun des fichiers qui composent notre applications.*

***Pour packager une application, réunissez tous les fichiers nécessaires à votre application dans un fichier JAR et ajoutez-y un manifeste spécifiant la classe principale de votre programme - celle qui doit être invoquée en premier par le lanceur de programme Java.***

Avant la fabrication du fichier archive JAR, préparez le fichier manifeste "manifeste.mf" afin d'indiquer la classe principale de l'application. Créer ensuite votre archive au moyen de la commande jar. Vous avez ci-dessous en image toute la procédure à suivre. Remarquez en dernier lieu le test effectué avec le lancement de l'application au moyen de la commande java, à partir de l'archive qui vient d'être construite.



Pour en savoir plus sur les archives Java.  
§

### Fichier de description texte.jnlp

Nous devons maintenant indiquer comment déployer notre application. Pour cela, il est nécessaire de mettre en oeuvre un fichier qui décrit comment réaliser cette opération. Il s'agit d'un fichier de lancement au format XML et qui respecte le protocole prévu à cet effet, c'est-à-dire le protocole JNLP. Nous devons donc préparer ce fichier qui pour notre exemple se nommera texte.jnlp avec le contenu suivant :

# texte.jnlp

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <jnlp codebase="http://portable:8084/download/">
3   <information>
4     <title>Application Texte</title>
5     <vendor>Emmanuel REMY</vendor>
6     <icon href="icone.jpg" />
7     <offline-allowed />
8     <shortcut>
9       <desktop />
10      <menu submenu="Applications distantes" />
11    </shortcut>
12  </information>
13  <resources>
14    <j2se version="1.5+" />
15    <jar href="texte.jar" />
16  </resources>
17  <application-desc />
18 </jnlp>

```

Depuis la version 6.0 de Java, vous pouvez utiliser le nom de balise `<java>` au lieu de `<j2se>`, ce qui est d'ailleurs assez logique. Ainsi :

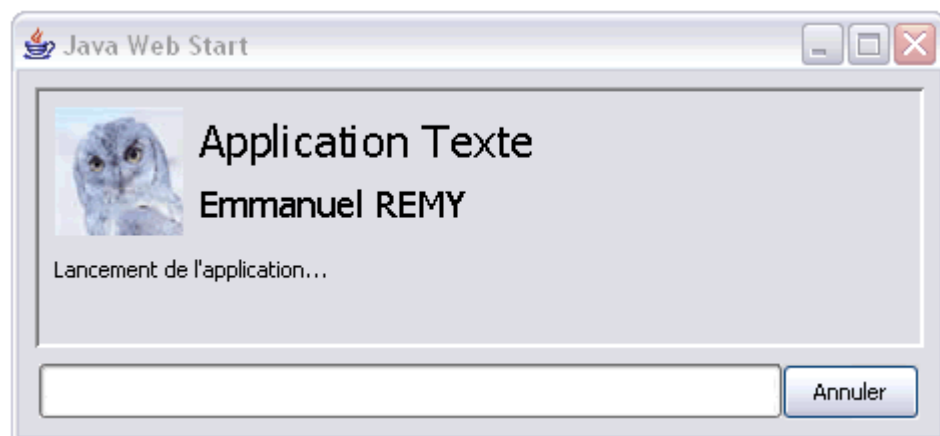
`<j2se version="1.5+" />` devient `<java version="1.5+" />`

## <jnlp>

C'est l'élément racine d'un document ".jnlp". Il doit comporter trois éléments enfants au minimum : `<information>`, `<resources>` et `<application-desc>` (ou `<applet-desc>`) et éventuellement d'autres éléments enfants comme `<security>`. Cet élément racine peut également posséder plusieurs attributs. Un qui est important et qui est donc indispensable stipule quel est l'application Web qui sert de conteneur aux applications à déployer. Il s'agit de l'attribut `codebase`.

## <title>

Cet élément est obligatoire et précise le nom de l'application qui apparaîtra lorsque nous souhaiterons la lancer. Voici ci-dessous ce que vous aller voir suivant que vous lancer l'application la première fois lors du téléchargement, ou bien lorsqu'elle est lancée à partir du poste client (application déjà présente sur le poste local) :



**<vendor>**

Nom de l'auteur. Egalement obligatoire.

**<icon>**

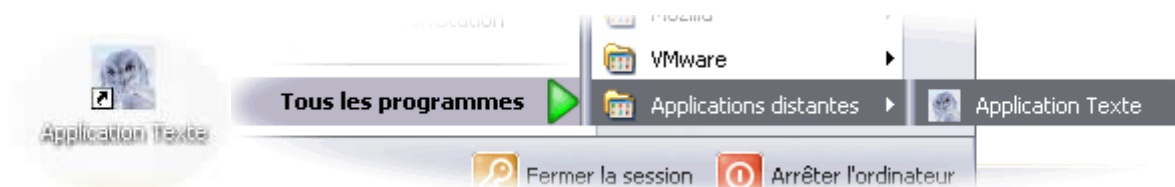
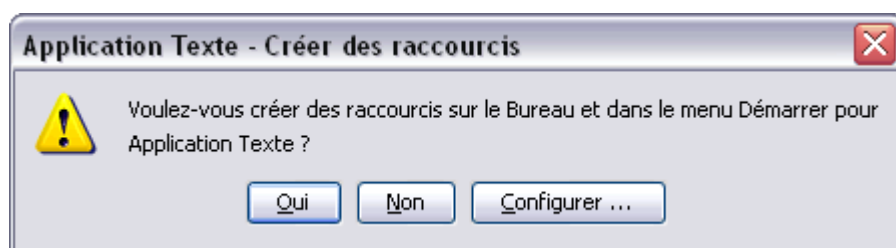
Permet de proposer une icône personnalisée plutôt que l'icône de Java. Le format requis est 64x64 pixels. Cet élément est optionnel.

**<offline-allowed />**

Élément également optionnel qui stipule, lorsqu'il est placé dans ce descripteur, que nous désirons que l'application demeure sur le poste client. Dans ce cas là, nous pouvons l'utiliser sans faire appel au service Web distant. Ceci dit, si le service Web est actif, le système contrôle si une nouvelle version n'existe pas. Si c'est vrai, elle est automatiquement téléchargée et remplace la précédente.

**<shortcut>**

Élément optionnel qui permet de placer automatiquement des raccourcis, éventuellement sur le bureau, et éventuellement dans le menu démarrer. C'est quand même le client qui donne l'autorisation du placement de ces raccourcis :



**<j2se>**

Elément obligatoire. Vous devez spécifier la machine virtuelle installée. Dans ce cas là, nous spécifions que nous prenons la JRE 5.0 au supérieure. Attention, ne marquez pas 5.0, mais bien 1.5, puisque c'est la notation qui est prise par le système d'exploitation.

**<jar>**

Elément obligatoire qui indique qu'elle est l'application packagée correspondante. C'est l'application que nous désirons déployer.

**<application-desc />**

Elément obligatoire qui indique quel est le type de programme. S'agit-il d'une application classique <application-desc/>, ou d'une applet <applet-desc/>.

### Mise en place de l'application Web <http://portable:8084/download/>

*Comme nous l'avons évoqué plus haut, la technologie Java Web Start utilise le serveur Web comme moyen de déploiement. De toute façon, il faut qu'il ait un service de monter. Autant que ce soit donc celui qui est le plus universel. D'autre part, grâce au serveur Web, vous pourrez du coup déployer vos applications directement sur Internet.*

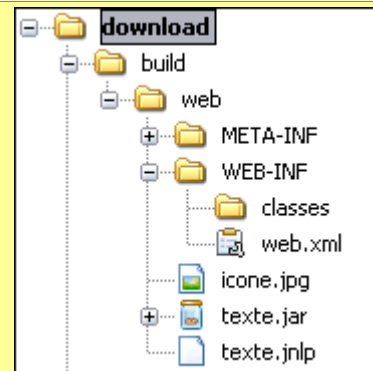
Pour cela, il paraît judicieux de fabriquer une application Web qui soit spécialisée sur les déploiement d'applications à distance. Il suffit donc, dans votre serveur Web, de fabriquer une application Web /download/ à l'intérieur de laquelle vous placerez toutes vos applications à déployer.

*Attention, l'application Web doit respecter une certaine architecture pour qu'elle soit opérationnelle. Ceci dit, c'est très simple, il s'agit d'un ensemble de répertoires prédéfinis et d'un descripteur de déploiement <web.xml> vide (ou presque) situé dans le répertoire (privé) <WEB-INF> dont voici le codage interne ci-dessous.*

### Descripteur de déploiement web.xml + Application Web /download/

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">
</web-app>
```



*Vous devez ensuite placer l'ensemble de vos applications packagées à la racine de votre application Web ainsi que les fichiers de lancement (avec éventuellement les icônes). Ici le répertoire racine est appelé <web>.*

*Ici, je ne vais pas m'étendre beaucoup plus sur la notion d'application Web. Ce sujet est largement traité dans l'ensemble de mes cours, notamment dans la partie J2EE.*

### Déploiement de vos applications vers le poste client



*Il ne vous reste plus qu'à lancer votre serveur Web. Tout est prêt pour le déploiement de vos applications vers les postes clients. Une fois que votre serveur Web est actif, pointer le navigateur du poste client vers le fichier JNLP correspondant à l'application que vous désirez récupérer. Dans notre exemple, tapez l'URL suivante :*

`http://portable:8084/download/texte.jnlp`

Une autre alternative consiste à lancer l'application Web /download/ en entier et de choisir l'application désirée au moyen de la souris. Pour cela tapez l'URL suivante :

`http://portable:8084/download/`

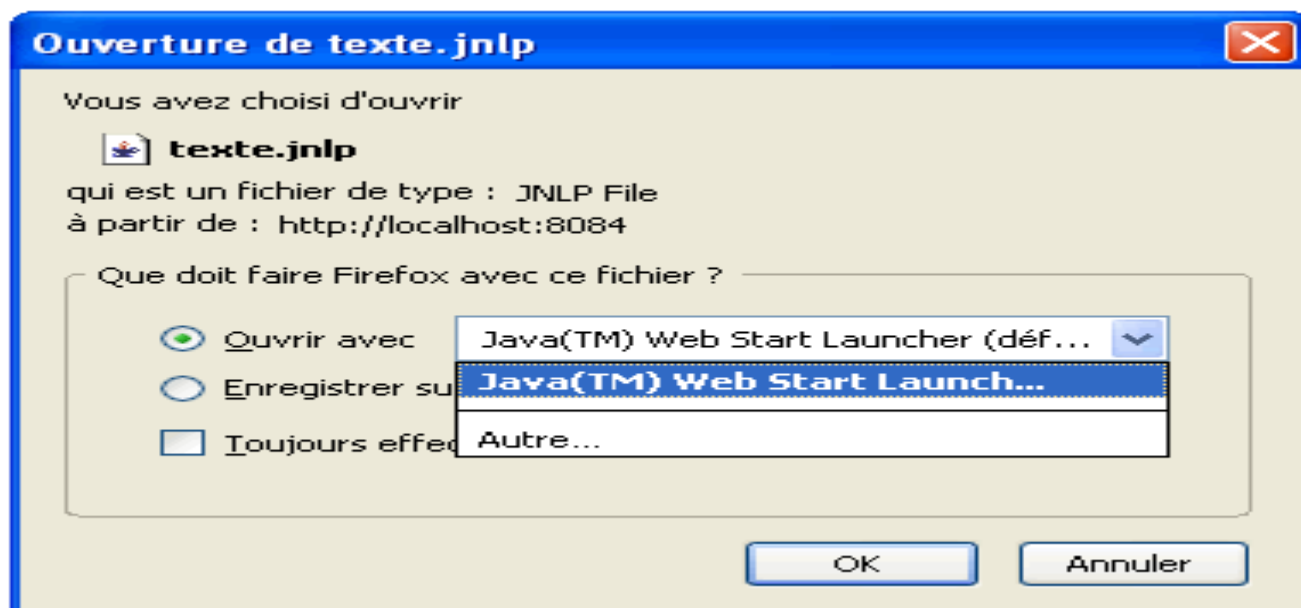
Ce qui donne l'affichage suivant :



Et maintenant, vous pouvez cliquer sur `texte.jnlp`.

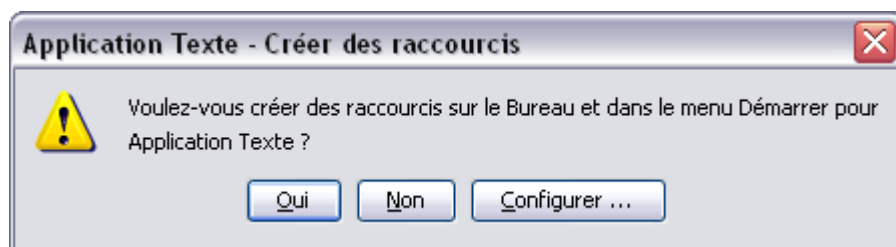
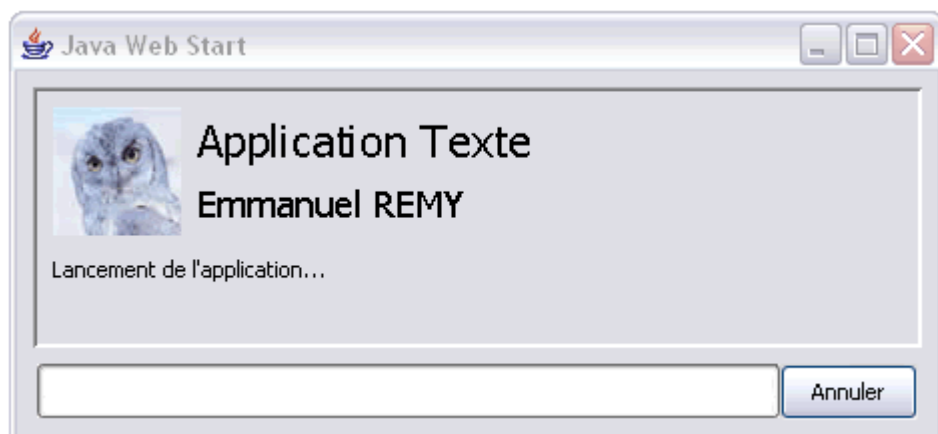
Quelque soit la solution, la première fois, vous avez une boîte de dialogue standard dans le milieu du Web qui apparaît et qui vous demande ce que vous désirez faire avec ce type de fichier. Souhaitez-vous ouvrir ou enregistrer ce fichier sur votre poste local. Par ailleurs, le système reconnaît l'application qui est capable d'interpréter ce type de fichier : ici Java Web Start. Choisissez l'ouverture plutôt que l'enregistrement.





*Pour éviter que cette boîte de dialogue s'affiche à chaque fois que vous désirez lancer une nouvelle application, je vous invite à cocher la case "Toujours effectuer cette action pour ce type de fichier".*

Vous avez ensuite la fenêtre de lancement de l'application qui s'affiche suivie de la boîte de dialogue éventuelle permettant d'installer les raccourcis.



Et votre application s'affiche enfin :

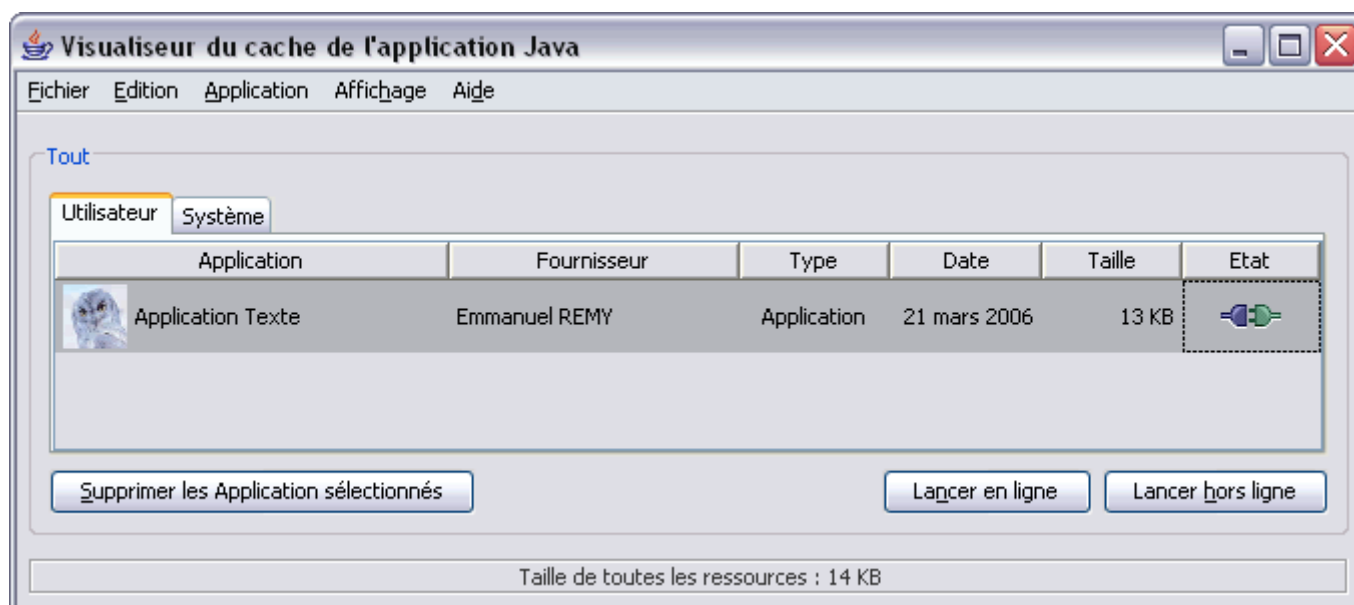


*Remarquez la présence du bandeau : Java Application Window.*

### Relancer l'application sur le poste client

*Vu le choix que nous avons fait, l'application se trouve dans le cache de l'ordinateur client. Si vous avez des raccourcis sur votre poste, il suffit de les solliciter afin de lancer de nouveau l'application concernée. Si le serveur Web est encore actif, Java Web Start va contrôler s'il n'existe pas une version plus récente sur le serveur, auquel cas elle sera déployée à la place de l'ancienne version. Dans le cas contraire, l'application présente est tout simplement lancée.*

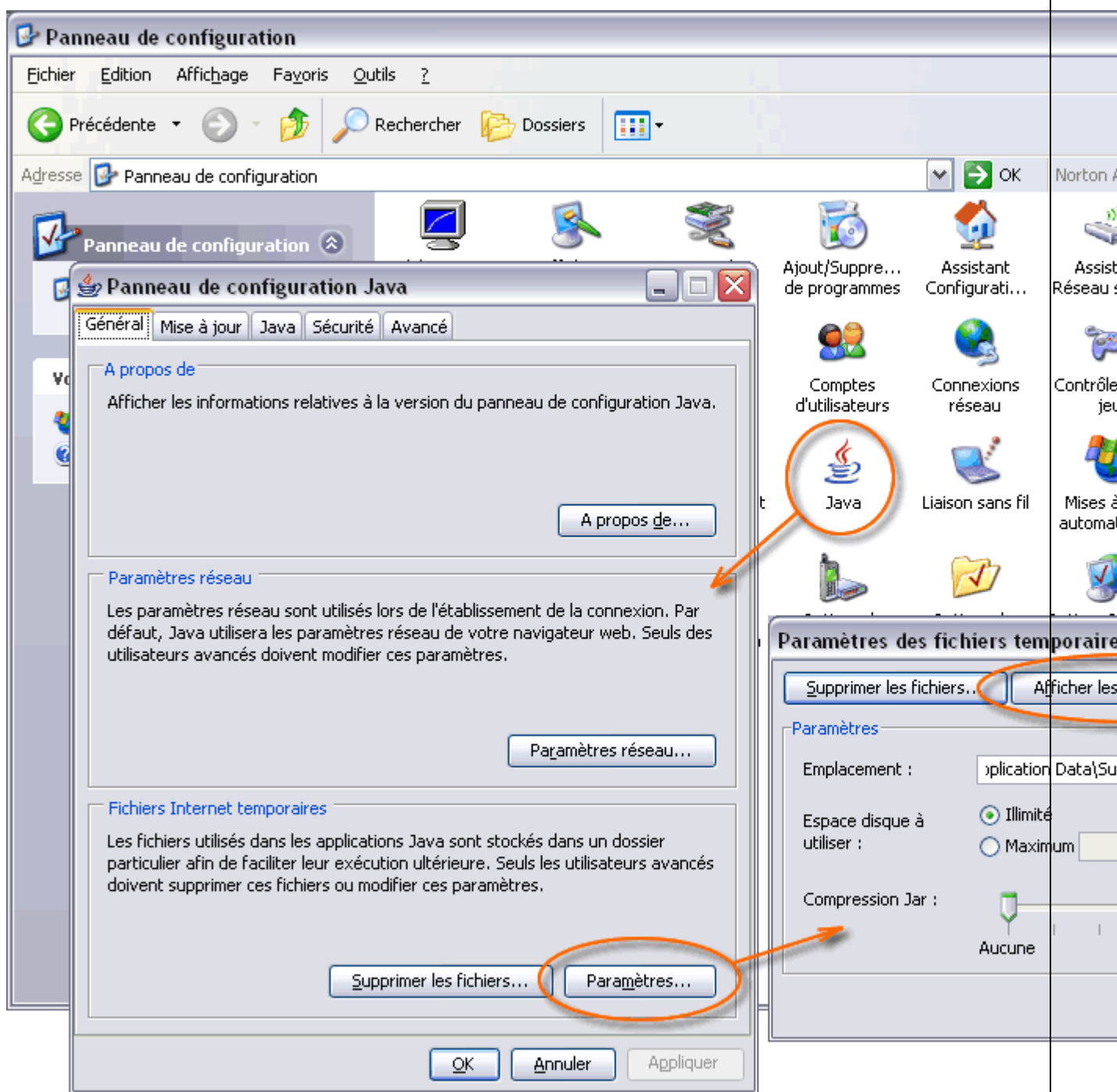
Si vous ne disposez pas de raccourcis, vous pouvez quand même accéder à nouveau au fichier JNLP, l'application est récupérée depuis la mémoire tampon du cache. Vous pouvez consulter le contenu à l'aide du panneau de contrôle du plug-in Java :



*De puis le JDK 5.0, ce panneau sert à la fois pour les applets et les applications Java Web Start. Ce panneau de contrôle appelé Visualisateur de cache permet de recenser les applications déjà présentes sur le poste client. Vous pouvez ainsi supprimer les applications que vous n'utilisez plus ou lancer celles qui vous désirez, en local ou depuis le serveur distant.*

*Lorsque vous supprimer une application, il s'agit en fait d'une désinstallation. Si vous avez des icônes sur le bureau ou des raccourcis dans le menu Démarrer, ils sont automatiquement enlevés.*

**Pour retrouver ce panneau de contrôle sous Windows, vous devez passer par le Panneau de configuration :**



**Lancer l'application distante depuis le fichier JAR ou depuis le fichier de lancement JNLP**

*Revenons à notre navigateur au moment où notre application Web /download/ apparaît :*



*Nous remarquons la présence à la fois de l'application packagée `texte.jar` ainsi que du lanceur d'application `texte.jnlp`. A ce niveau, rien ne nous empêche de cliquer sur le lien `texte.jar`. Si vous effectuez cette action et si votre système d'exploitation est réglé pour rattacher ce type de fichier avec le programme adapté, c'est-à-dire ici la machine virtuelle Java, vous aurez effectivement l'application qui se lance automatiquement comme si elle se trouvait sur le poste local.*

**Alors pourquoi s'embêter avec ce fichier de lancement ? Est-ce que Java Web Start est avantageux ? Quels sont ses atouts ?**

1. *Si vous passer directement par l'archive, votre application a accès à toute les ressources de la machine cliente, alors que Java Web Start propose un contrôle minimum. Il est généralement souhaitable que l'application soit contrôlée afin d'éviter tout disfonctionnement ultérieur.*
2. *Avec l'archive, une fois que l'application a fini son travail, elle n'existe plus du tout sur la machine locale. Sauf, bien entendu si vous avez demandé, au moment du téléchargement, à l'enregistrer sur le disque dur local.*
3. *Avec Java Web Start, toutes les applications déployées sur la machine cliente se situent au même endroit. Il est ainsi facile de recenser ce que nous avons sur notre poste local et de supprimer les applications qui nous intéressent plus. Nous pouvons même vider complètement le cache.*
4. *Ainsi, avec Java Web Start, il est possible de lancer l'application sans être connecté avec le serveur distant.*
5. *Lorsque le serveur distant est opérationnel, Java Web Start vérifie s'il n'existe pas une version plus récente de l'application que nous désirons démarrer.*
6. *Au moment du déploiement, Java Web Start installe éventuellement des raccourcis sur le bureau (ou/et) dans le menu Démarrer qui permettront une utilisation ultérieur beaucoup plus aisée.*
7. *Lors de la suppression d'une application, les éventuels raccourcis sont automatiquement désinstallés.*

*Dans ce fichier de lancement JNLP, nous avons très peu de lignes à décrire. C'est très rapide à mettre en oeuvre. Par ailleurs, nous pouvons ainsi choisir le nom que portera l'application, l'icône représentative, etc. Et vu les avantages que nous venons de voir, il est largement avantageux de travailler avec cette technologie.*

## Restructuration de l'application Web /download/

*Finally, to avoid confusion between the two types of file, it would be desirable to see the Web application so that only the choice of the launch file is proposed. When we connect to the Web application /download/, the ideal would be to present a welcome page that displays the set of downloadable applications. Thus, when we click on one of the proposed links, we will automatically activate the corresponding JNLP file.*



*Ceci dit, rien n'empêche de se connecter directement sur le fichier archive au moyen de l'URL suivante <http://portable:8084/download/texte.jar>. Toutefois, le client n'est pas obligé de le savoir.*

Si vous désirez qu'une page d'accueil soit automatiquement lancée lors de la connexion à votre application Web, il est souvent souhaitable de respecter le nom par défaut qui est généralement index.html. Toutefois, rien ne vous empêche d'utiliser un autre nom de page. C'est ce que je vais faire ici, je vais l'appeler applications.html. Par contre, il faut que le serveur Web connaisse le nom de cette page d'accueil. Pour cela il faut changer le contenu du descripteur de déploiement <web.xml> et se servir de la balise <welcome-list-file> comme cela vous est montré ci-dessous :

### web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <welcome-file-list>
    <welcome-file>applications.html</welcome-file>
  </welcome-file-list>
</web-app>
```

Voici maintenant le contenu de notre page d'accueil applications.html qui vous le remarquez est relativement modeste :

### Applications.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
  <head>
    <title>Applications distantes</title>
  </head>
  <body>
    <h2>Liste des applications &agrave; d&eacute;ployer</h2>
    <hr />
    
    <a href="texte.jnlp">Application Texte</a>
  </body>
</html>
```

### Gestion de la sécurité - Signer une archive jar

*L'exemple que nous venons de traiter fonctionne parfaitement. Il faut dire que cette application fonctionne en autonomie sans utiliser les ressources du poste local. Cette application reste dans sa zone bien délimité qui est, ce que nous appelons, le "bac à sable".*

### Le bac à sable

*Dès lors qu'un code est chargé depuis un site distant, puis exécuté en local, la sécurité devient un enjeu essentiel. Le fait de cliquer sur un seul lien peut lancer une application Java Web Start, téléchargé sur Internet, qui présente donc un risque intrinsèque et s'exécute dans un "bac à sable" avec un accès minimal.*

*Effectivement, lorsqu'une application se trouve dans "le bac à sable", elle dispose de droits limités. Elle n'utilise que les capacité du processeur et de sa zone mémoire. Elle n'a pas le droit d'accéder aux ressources de la machine locale, comme les fichiers sur le disque dur. Elle n'a pas non plus la possibilité d'accéder à une autre machine au travers du réseau local.*

*Intérêt du bac à sable : L'affichage d'une page Web lance automatiquement tous les applets de la page. Si vous cliquez sur un lien ou que vous visitez une page Web, vous pourriez installer un code arbitraire sur l'ordinateur de l'utilisateur. Les criminels n'auraient aucunes difficulté à voler des informations confidentielles, à accéder à des données financières ou à prendre le contrôle des machines pour envoyer des spams.*

### Gestion de la sécurité sur le poste local

*Pour que sa technologie ne soit pas employée à des fins coupables, Java propose intrinsèquement un modèle de sécurité élaboré. Un gestionnaire de sécurité vérifie en permanence l'accès à toutes les ressources du système. Par défaut, il n'autorise que les opérations inoffensives.*

*Pour autoriser des opérations supplémentaires, le code doit être numériquement signé et l'utilisateur doit approuver le certificat de signature.*  
§

### Peut-on travailler uniquement avec le bac à sable ?

*Mais que peut faire un code distant (récupéré sur Internet) sur toutes les plates-formes locales ?*

1. *Il est toujours bon de pouvoir afficher des images,*
2. *de diffuser du son,*
3. *de récupérer les frappes de touche et les clics de l'utilisateur*
4. *et de renvoyer la saisie utilisateur à l'hôte à partir duquel le code a été chargé.*

*Vous avez là suffisamment de fonctionnalités pour afficher des faits et des figures ou recevoir une saisie utilisateur visant à passer commande. L'environnement d'exécution restreint est souvent appelé "bac à sable". Le code qui s'y exécute ne peut donc pas modifier le système de l'utilisateur ni l'espionner.*

#### **Enumération des restrictions du bac à sable**

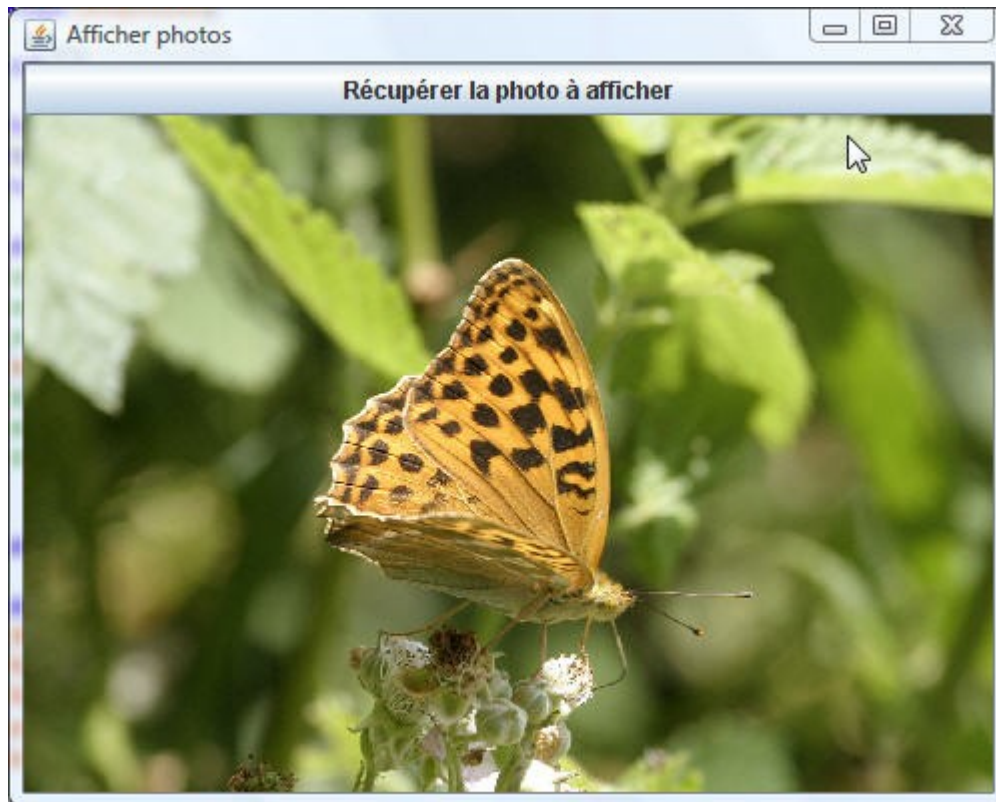
*Les programmes du bac à sable présentent notamment les restrictions suivantes :*

5. *Ils ne peuvent jamais exécuter de programme exécutable local.*
6. *Ils ni lire ni écrire sur l'ordinateur local. Toutefois, avec des privilèges de sécurité minimale, l'API JNLP fournit le système de fichiers.*
7. *Ils ne peuvent retrouver aucune information sur l'ordinateur local, à l'exception de la version de Java utilisée et de quelques détails inoffensifs sur le système d'exploitation. Notamment, le code du bac à sable ne retrouve pas le nom de l'utilisateur, son adresse e-mail, etc.*
8. *Les programmes chargés à distance ne peuvent pas communiquer avec un hôte autre que le serveur à partir duquel ils ont été téléchargés ; ce serveur est appelé hôte d'origine. La règle protège les utilisateurs du code qui pourraient essayer d'espionner les ressources Intranet (depuis Java SE 6, une application Java Web Start peut réaliser d'autres connexions réseau, mais l'utilisateur du programme doit y consentir).*
9. *Toutes les fenêtres contextuelles véhiculent un message d'avertissement. Ce message est une fonction de sécurité pour s'assurer que les utilisateurs ne confondent pas la fenêtre avec une application locale. La crainte est qu'un utilisateur peut suspicieux visite une page Web, soit amené par erreur à exécuter un code distant, puis tape un mot de passe ou un numéro de carte de crédit, qui peut ensuite être renvoyé au serveur Web. Le message inscrit est soit "Java Applet Window" ou "Java Application Window".*

#### **Application qui utilise les ressources du poste local à déployer au travers de Java Web Start**

*Nous avons quelque fois besoin de développer des applications qui nécessitent de travailler avec les fichiers du poste local. Voici, par exemple, une application qui est une visionneuse des photos présentes sur la machine locale :*





#### jwsphotos.AfficherPhotos

```
package jwsphotos;

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import javax.swing.*;

public class AfficherPhotos extends JFrame implements ActionListener {
    private JButton soumettre = new JButton("Récupérer la photo à afficher");
    private PanneauImage panneau = new PanneauImage();

    public AfficherPhotos() {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize(500, 400);
        setTitle("Afficher photos");
        soumettre.addActionListener(this);
        add(soumettre, BorderLayout.NORTH);
        add(panneau);
        setVisible(true);
    }

    public static void main(String[] args) {
        new AfficherPhotos();
    }

    public void actionPerformed(ActionEvent e) {
        JFileChooser fichier = new JFileChooser();
        fichier.setDialogTitle("Sélectionner la photo");
        fichier.showOpenDialog(this);
        panneau.setImage(fichier.getSelectedFile());
        panneau.repaint();
    }
}
```

```

    }
}

class PanneauImage extends JPanel {
    private BufferedImage image;
    private double ratio;

    public void setImage(File fichier) {
        try {
            image = ImageIO.read(fichier);
            ratio = (double)image.getWidth()/image.getHeight();
        }
        catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    protected void paintComponent(Graphics surface) {
        if (image!=null) surface.drawImage(image, 0, 0, this.getWidth(), (int)(this.getWidth()/ratio), null);
    }
}

```

### Déploiement au travers de Java Web Start

*Si nous mettons en oeuvre tout le système de déploiement en utilisant les compétences de Java Web Start, nous obtenons bien le téléchargement automatique de notre application, toutefois cette dernière ne fonctionne pas correctement. En effet, la boîte de dialogue d'ouverture de fichier n'apparaît pas. Ainsi, il est impossible de récupérer l'image à visionner. Effectivement, pour l'instant, nous n'avons rien indiqué de spécial, il est normal que l'application reste dans "le bac à sable", avec donc des droits limités.*

### Augmenter les droits de l'utilisateur à l'aide de la balise <security>

*Il est possible d'augmenter les droits d'utilisation sur votre application. Pour cela, il faut utiliser la balise <security> dans votre fichier de description jnlp :*

<security>

Sécurité à apporter pour l'application. Si vous ne faites aucune spécification, l'application s'exécute dans le « bac à sable », donc comme une applet avec juste assez de droit pour son fonctionnement. Pas d'accès au disque dur, de communication réseau, etc.

<all-permissions />

Définit l'application comme ayant tous les droits d'une application normale java. Attention, dans ce cas là, il est nécessaire que TOUTES les archives jar soient signées, et que l'utilisateur ait donné son accord au moment du lancement.

<j2ee-application-client-permissions />

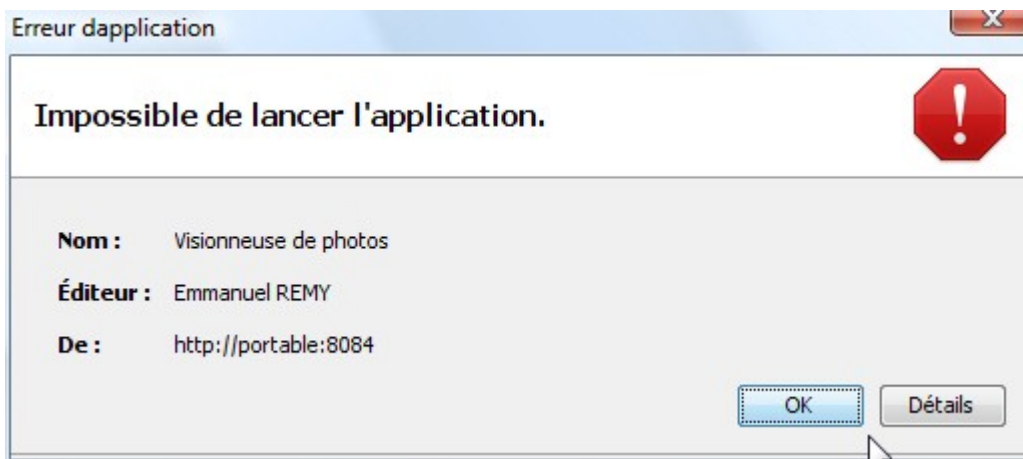
Indique que l'application d'entreprise doit avoir les droits d'une application classique Java, avec les mêmes critères que précédemment.

En tenant compte de ces nouvelles possibilités, voici le fichier de description jnlp que nous pouvons mettre en oeuvre :

photos.jnlp

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp codebase="http://portable:8084/download/" >
  <information>
    <title>Visionneuse de photos</title>
    <vendor>Emmanuel REMY</vendor>
    <offline-allowed />
    <shortcut>
      <desktop />
      <menu submenu="Applications distantes" />
    </shortcut>
  </information>
  <resources>
    <j2se version="1.6+" />
    <jar href="JWSPHOTOS.jar" />
  </resources>
  <security>
    <all-permissions />
  </security>
  <application-desc />
</jnlp>
```

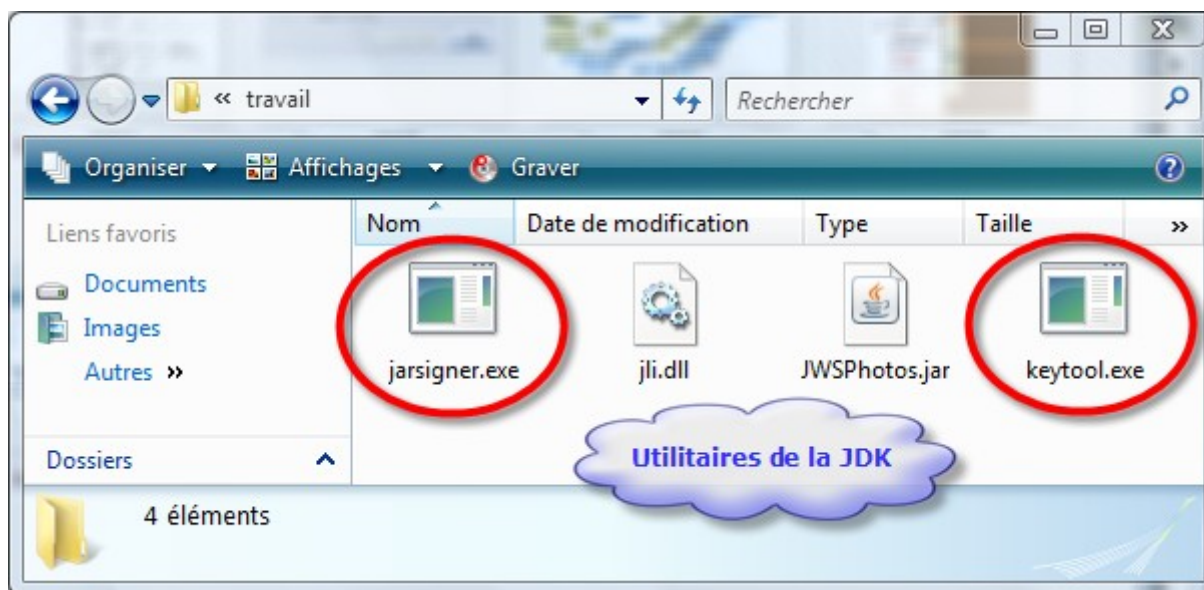
Si nous tentons de nouveau de déployer notre application au travers de Java Web Start, cette fois-ci le déploiement ne s'effectue pas correctement, et vous verrez apparaître le message suivant :



### Signer une archive jar

*Lorsque vous utiliser la balise <all-permissions /> à l'intérieur de la balise <security>, vous devez impérativement signer votre archive, ici JWSPHOTOS.jar. La signature d'une archive est très importante, elle permet de savoir qui a fait l'archive de manière fiable. C'est une façon de prouver qu'il s'agit bien de votre travail. C'est une sorte de signature électronique. Elle va permettre à notre application d'avoir le droit d'accéder aux fichiers systèmes, à la communication sur le réseau, etc. à l'identique d'un programme java standard.*

*Le principe de la signature marche sur un cryptage clé privée/clé public. La signature va ajouter quelques lignes dans le fichier manifest ainsi que d'autres fichiers permettant de confirmer qu'il s'agit de vous et que l'archive n'a pas été modifiée après signature. La JDK délivre deux utilitaires qui permettent de signer votre archive à déployer, keytool et jarsigner :*



### L'utilitaire keytool

*Le programme keytool gère des magasins de clés (key stores), bases de données de certificats et de clés privés. Chaque entrée dans le magasin de clés possède un alias. keytool permet ainsi d'obtenir une clé privée associée à des clés publiques. Celles-ci vont être stockées dans un fichier qui va servir plus tard pour la signature. Chaque clé est identifiée par un alias, généralement le nom du signataire. Ce fichier peut contenir la signature de plusieurs personnes.*

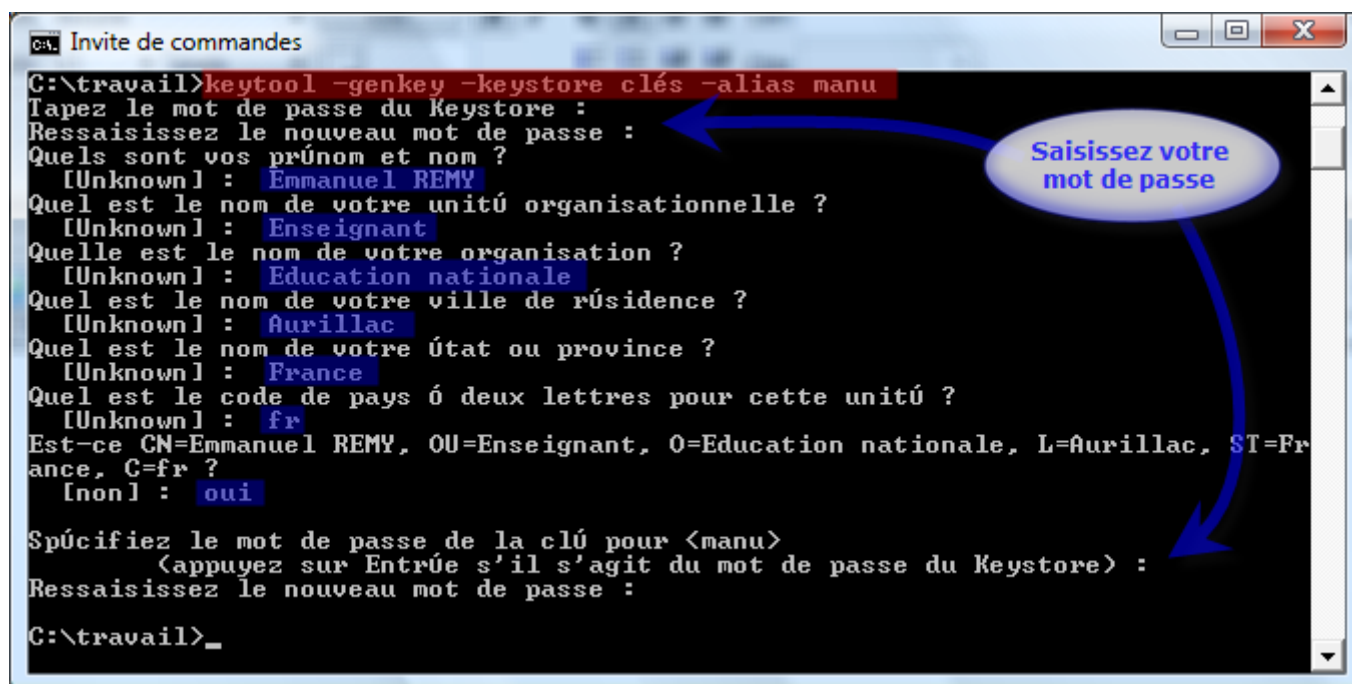
### Générer une clé

*Voici, d'une façon générale, les paramètres à spécifier pour l'utilisation de cet outil :*

```
keytool -genkey -keystore magasinDesClés -alias votreNom
```

*Lorsque vous créez ou ouvrez un magasin de clés, vous êtes invité à fournir un mot de passe pour le magasin de clés. Il convient de choisir un vrai mot de passe et de protéger ce fichier, car il contient les clés de signature privées.*

Lorsque vous générez une clé, vous êtes invité à fournir les informations suivantes :

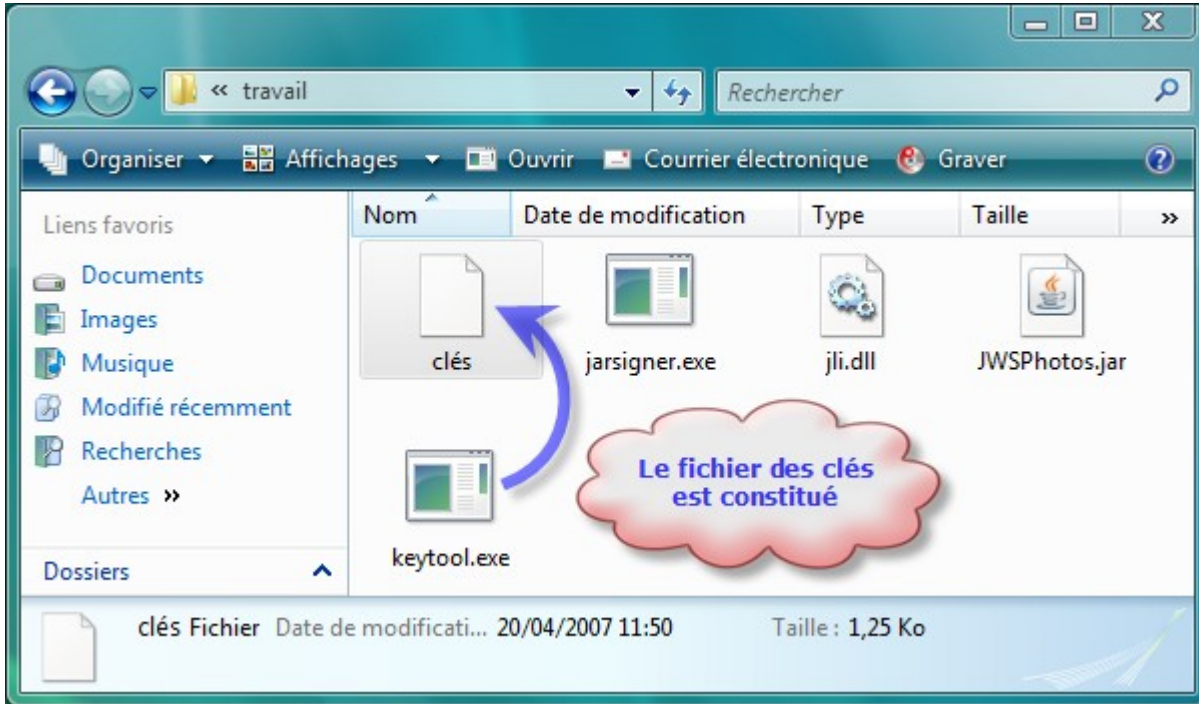


*Vous remarquez qu'à la fin de la saisie, vous devez fournir un mot de passe spécifique pour la clé que vous venez de créer, ou appuyer sur Entrée pour utiliser le mot de passe du magasin de clés comme mot de passe de clé.*

L'outil keytool utilise les noms distingués X.500 pour identifier les propriétaires de clé et les émetteurs de certificat, avec les composants suivants :

1. Common Name (CN) : prénom et nom
2. Organizational Unit (OU) : service
3. Organization (O) : société
4. Location (L) : ville
5. State (ST) : état
6. Country (C) : code pays

Grâce à cet outil keytools, le fichier de clés est en place :



#### Mise en oeuvre d'un certificat

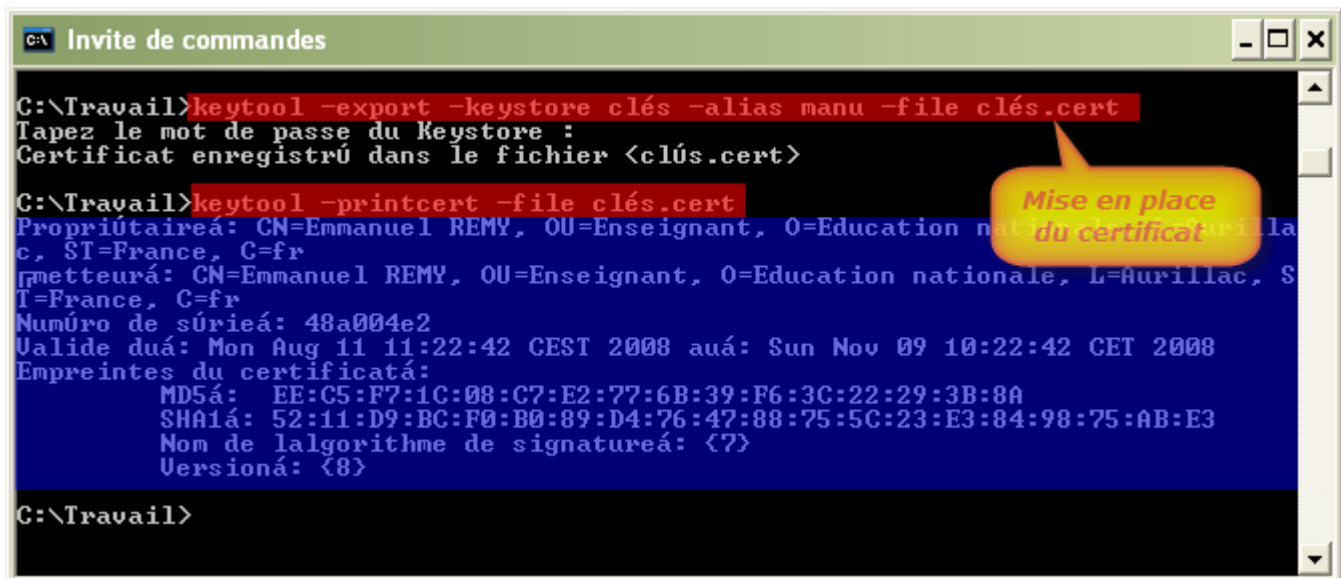
Supposons que manu veuille donner sa clé publique à Paul. Il doit alors exporter un fichier de certificat au moyen de l'option export de l'outil keytool :

```
keytool -export -keystore clés -alias manu -file manu.cert
```

Ensuite manu peut envoyer le certificat à Paul. Lorsque Paul reçoit le certificat, il peut l'imprimer au moyen de l'option printcert de l'outil keytool :

```
keytool -printcert -file manu.cert
```

Voici ci-dessous les commandes successives à réaliser à partir du magasin de clés clés :





**Ce certificat est autosigné. Par conséquent, Paul ne peut pas utiliser un autre certificat garanti pour vérifier que ce certificat est valide. Il peut, à la place, appeler manu et lui demander de lire l'empreinte du certificat au téléphone.**

### Importation d'un certificat

Lorsque Paul a toute confiance en son certificat, il peut l'importer dans son magasin de clés au moyen de l'option import de l'outil keytool :

```
C:\Travail>keytool -import -keystore Paul.store -alias manu -file clés.cert
Tapez le mot de passe du Keystore :
Ressaisissez le nouveau mot de passe :
Propriétaire: CN=Emmanuel REMY, OU=Enseignant, O=Education nationale, ST=France, C=fr
Émetteur: CN=Emmanuel REMY, OU=Enseignant, O=Education nationale, ST=France, C=fr
Numéro de série: 48a004e2
Valide du: Mon Aug 11 11:22:42 CEST 2008 au: Sun Nov 09 10:22:42 CEST 2008
Empreintes du certificat:
MD5: EE:C5:F7:1C:08:C7:E2:77:6B:39:F6:3C:22:29:3B:8A
SHA1: 52:11:D9:BC:F0:B0:89:D4:76:47:88:75:5C:23:E3:84:98:75:AB:E3
Nom de l'algorithme de signature: <?>
Version: <8>
Faire confiance à ce certificat ? [non] : oui
Certificat ajouté au Keystore
C:\Travail>
```

**N'importez jamais dans un magasin de clés un certificat dont vous n'êtes pas absolument sûr. Une fois que le certificat est ajouté au magasin de clés, n'importe quel programme utilisant le magasin de clés suppose que le certificat peut être utilisé pour vérifier les signatures.**

### Documents signés

A présent manu peut commencer à envoyer des documents signés à Paul. L'outil jarsigner signe et vérifie les fichiers JAR.

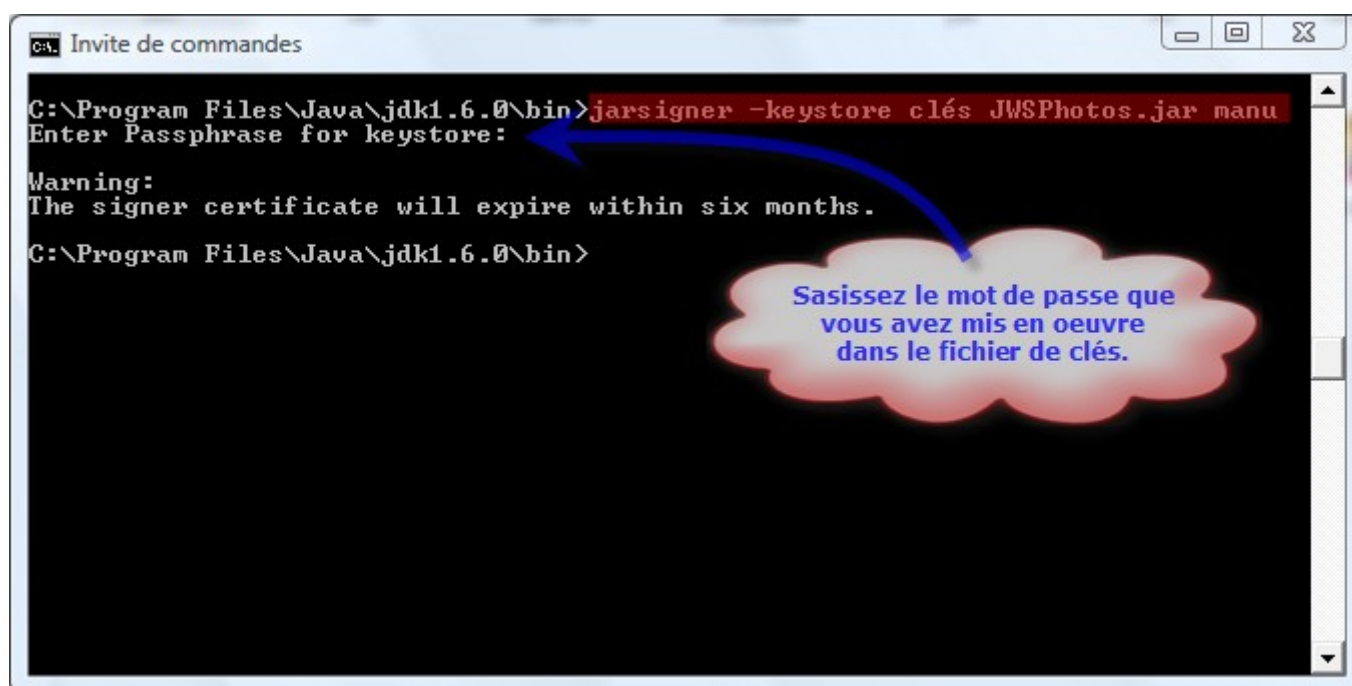
### L'utilitaire jarsigner

A l'aide de ce fichier de clés, nous pouvons maintenant signer notre archive JWSPHOTOS.jar, et ceci grâce à l'utilitaire jarsigner. Voici d'ailleurs la commande générale à soumettre :

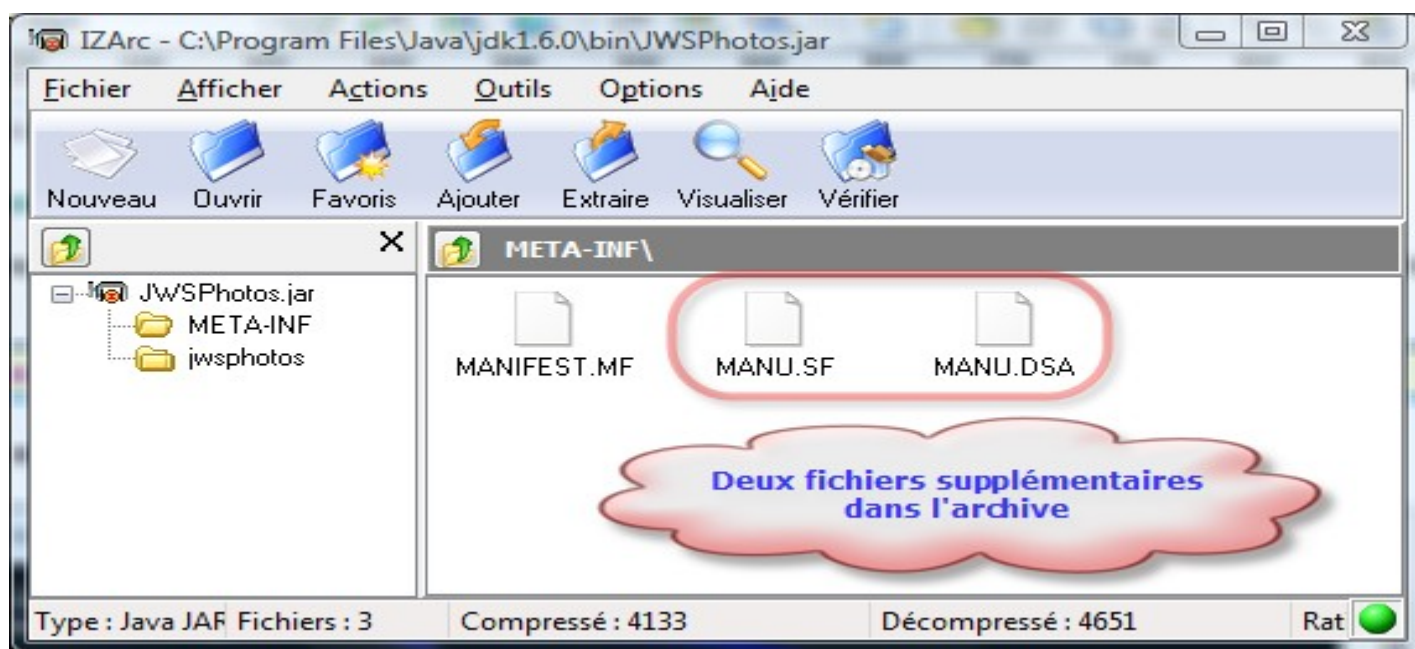
```
jarsigner -keystore magasinDesClés archive votreNom
```

Dans cet outil, vous avez juste à spécifier le même mot de passe que vous utilisez lors de l'élaboration des clés :





Vous remarquez que votre certificat possède une durée de vie limité à six mois. Si vous ouvrez l'archive signée, vous voyez alors apparaître deux fichiers supplémentaires :



Placez maintenant votre archive signée dans votre application Web de déploiement Java Web Start. Cette fois-ci, le déploiement s'effectue correctement. L'utilisateur doit, par contre, confirmer son utilisation sur son ordinateur :



*Après avoir validé, voici donc le programme de votre application qui se lance, cette fois-ci, et contrairement aux applications qui tournent dans "le bac à sable", sans la présence du bandeau Java Application Windows :*

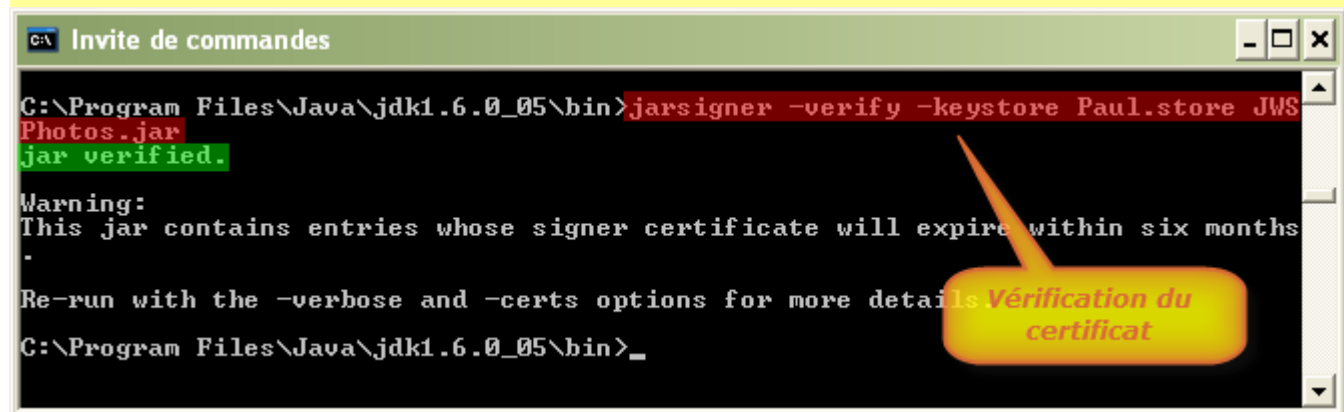


*Au delà de Java Web Start, il est possible d'utiliser un programme sur une machine locale dont le fichier archive est signé. Imaginons que ce soit Paul qui doit utiliser ce programme. Nous avons déjà mis en place un certificat qui lui autorise de l'utiliser. Lorsque Paul reçoit le fichier d'archive, il utilise l'option verify de l'utilitaire jarsigner :*

```
jarsigner -verify -keystore Paul.store JWSPHOTOS.jar
```

*Paul n'a pas besoin de spécifier l'alias de clé. Le programme jarsigner trouve le nom X.500 du propriétaire de clés dans la signature numérique et recherche les certificats correspondants dans le magasin de clés.*

*Si le fichier JAR n'est pas corrompu et que la signature corresponde, le programme jarsigner imprime le texte suivant : jar verified :*



```
C:\Program Files\Java\jdk1.6.0_05\bin>jarsigner -verify -keystore Paul.store JWSPhotos.jar
jar verified.

Warning:
This jar contains entries whose signer certificate will expire within six months
.

Re-run with the -verbose and -certs options for more details
C:\Program Files\Java\jdk1.6.0_05\bin>_
```

Vérification du certificat