

Premiers pas avec la librairie Java 3D

par [Alain Mari](#)

Date de publication : 14/09/2005

Dernière mise à jour :

Dans ce chapitre nous allons faire les premiers pas indispensables avec Java 3D, c'est à dire l'installation de la librairie Java 3D puis son intégration dans des IDE (Integrated Development Environment) comme JBuilder. Enfin, nous agrémenterons le tout d'un premier exemple très simple.

Introduction

- 1 - Installation de Java 2 et Java 3D
 - 1.1 - Installation de Java
 - 1.2 - Installation de Java 3D
 - 1.3 - Mise à jour du PATH de Windows
- 2 - Un premier exemple avec Java3D
 - 2.1 - Visite guidée du code source ColorCube3D.java de notre premier exemple
 - 2.2 - Exécution du programme
- 3 - Utilisation de Java 3D avec JBuilder
- 4 - En préparation (Eclipse, chap 2, ...)
- 5 - Téléchargements

Introduction

Il existe plusieurs bibliothèques permettant d'écrire des applications 3D en Java. On peut notamment citer en sus de [Java 3D](#), les bibliothèques [JOGL](#) (Java for OpenGL) et [LWJGL](#) (LightWeight Java Game Library). Java 3D est sans doute la bibliothèque la plus simple à appréhender pour celui qui veut aborder la 3D en Java, en contrepartie c'est sûrement celle qui est la plus orientée objet, il est donc nécessaire d'avoir de bonnes connaissances en langage orienté objet avant d'aborder cette partie. Java 3D a été développée par Sun qui, dans un premier temps, l'a amenée jusqu'à la version 1.3.1, puis le code source a été ouvert et dès lors un groupe de passionnés continue à en assurer le développement. Actuellement, nous en sommes à la version 1.3.2 et une partie du code source est soumis à la licence [BSD](#) (Berkeley Software Distribution), ce qui fait que qu'il est opensource et librement redistribuable. Le nouveau projet consacré à Java 3D à partir de la version 1.3.2 est consultable à l'adresse [suivante](#).

La bibliothèque JOGL (développée conjointement par Sun et SGI) est considérée comme l'interface de référence entre OpenGL et Java. Elle est beaucoup plus proche d'OpenGL que ne l'est Java 3D tout en étant plus difficile à aborder mais, en même temps, elle a une approche beaucoup moins orientée objet.

Enfin, LWJGL est plutôt destinée aux développeurs de jeux 3D en Java basés sur la technologie OpenGL.

Dans cette partie consacrée à Java 3D, nous commencerons par les notions dites "de base" (objets, transformations géométriques, matériaux, textures, éclairage ...) puis nous aborderons des notions plus avancées comme l'interaction avec l'utilisateur, les animations, le morphing, la création d'images et de vidéos, le mipmapping, l'antialiasing, l'importation d'objets 3D provenant d'autres applications etc... Enfin nous terminerons notre étude de Java 3D par l'écriture d'un petit jeu vidéo 3D prenant en compte la plupart des notions abordées tout au long de cette partie.

Cette étude de Java 3D se présente à la fois comme un petit cours et comme une liste de recettes. J'ai essayé d'agrémenter chaque aspect d'exemples concrets et commentés. Bien sûr, rien n'est parfait, et si vous voyez une exactitude au niveau de la programmation, ne vous gênez pas pour le signaler, le forum et le bouton contact sont là pour ça !

Comme nul ne peut être exhaustif, certains aspects de Java 3D ne seront pas abordés ici comme les textures 3D et l'audio.

1 - Installation de Java 2 et Java 3D


1.1 - Installation de Java

Java 3D est une librairie qui est une extension de Java 2. Cette extension fournit tout un ensemble de méthodes permettant de créer "facilement" des applications ou applets graphiques 3D interactives. Elle va servir de lien avec les librairies système, dites de bas niveau, que sont DirectX (sous Windows uniquement) et OpenGL (présent sur la plupart des systèmes d'exploitation dont Windows et Linux). Ces librairies dites de "lien" sont appelées API binding en anglais. La raison pour laquelle j'ai décidé de créer une section spéciale Java 3D est que cette librairie est utilisable à la fois sous DirectX et OpenGL.

DirectX et OpenGL sont des librairies dites de "bas niveau" car elles sont directement liées au matériel, en particulier les cartes graphiques. Aujourd'hui, quasiment toutes les cartes graphiques du marché - et pas seulement le haut de gamme - possèdent des fonctions internes directement cablées pour DirectX et OpenGL. En général, lorsqu'on installe une carte graphique, il faut toujours mettre à jour les pilotes avec la dernière version disponible car OpenGL est fourni avec. Pour DirectX, il fait désormais partie intégrante de Windows, et les dernières versions et mises à jour peuvent toujours être téléchargées [ici](#) sur le site web de Microsoft.

OpenGL et DirectX sont tellement proches du matériel et du système d'exploitation que cela rend leur approche directe extrêmement complexe. Heureusement Java 3D se présente sous un angle beaucoup plus simple et convivial en se chargeant de faire le "pont" entre le développeur d'un côté et le système d'exploitation de l'autre. Les plus curieux qui voudront aller plus loin en 3D en visitant des sites web anglophones sur le sujet verront que les librairies comme Java 3D sont souvent désignées par le terme de wrapper (librairie enveloppe).

La librairie Java 3D ne peut pas fonctionner seule, il est nécessaire d'avoir installé au préalable Java 2. Java 2 comporte une machine virtuelle ainsi que toutes les librairies nécessaires à la création d'applications ou d'applets Java. La version 1.4.2 de Java 2 peut être téléchargée gratuitement [ici](#) sur le site web de Sun. Plus récemment, Sun a mis en ligne la version 1.5 (maintenant appelée JDK version 5.0) dite "Tiger" qui peut être téléchargée [ici](#). Pour pouvoir créer vos propres programmes Java, il est nécessaire de télécharger la version J2SE SDK (Java 2 Standard Edition Software Development Kit) qui contient l'intégralité des bibliothèques de programmation, tandis que la version J2SE JRE (Java 2 Standard Edition Java Runtime Environment) ne permet que l'exécution de programmes Java.

 *Tous les exemples Java 3D de cette partie ont été réalisés et testés sous Windows, cependant, ils fonctionnent également très bien sous Linux par exemple (merci Java !!!). L'utilisateur est libre de télécharger la version de Java adaptée pour le système d'exploitation de son choix. Cependant, en ce qui concerne la version Windows, je vous conseille de télécharger la version Windows Offline car il ne sera pas nécessaire de télécharger à nouveau le fichier en cas de réinstallation de Java 2. Dans la suite de ce paragraphe, nous détaillerons l'installation de Java 2 et Java 3D pour Windows uniquement.*

Ensuite, il suffit de double cliquer sur le fichier téléchargé et de se laisser guider par le processus d'installation. Toutefois, veillez à ce que le Plug-In Java soit bien enregistré pour le navigateur par défaut, cela permet de faire tourner des applets 3D au sein même du navigateur internet.



Une fois la procédure terminée, vous serez peut être amené à redémarrer le PC.

! Apparemment, la version 1.5.0 ne s'intègre pas toujours bien avec les navigateurs web lorsqu'on veut lancer une applet. Si une boîte de dialogue de message d'erreur apparaît lorsqu'on veut lancer une applet, il faut supprimer le fichier : `C:\Documents and Settings\[nom d'utilisateur]\Application Data\Sun\Java\Deployment\deployment.properties` puis fermer toutes les instances des navigateurs ouvertes et relancer l'applet. Celle-ci devrait alors s'exécuter sans problème.

1.2 - Installation de Java 3D

La dernière version pour Windows de Java 3D publiée par Sun est la version 1.3.1, elle peut se télécharger [ici](#). Comme nous l'avons vu dans l'introduction, le code source de Java 3D a désormais basculé dans le monde du libre, et une version 1.3.2 (Windows et Linux) peut être téléchargée [ici](#).

A ceux qui tiennent à télécharger la version 1.3.1 de Sun, je conseille fortement de prendre la version OpenGL plutôt que DirectX car certaines fonctionnalités de Java 3D ne sont activées qu'en OpenGL. Cependant, à partir de la version 4.09 des drivers Catalyst pour les cartes ATI, la version OpenGL de Java 3D 1.3.1 ne semble plus fonctionner, il est donc nécessaire dans ce cas de se rabattre sur la version DirectX. La version 1.3.2 résout les problèmes avec les cartes ATI. Ensuite, pour installer la version 1.3.1, il suffit de double cliquer sur l'exécutable précédemment téléchargé et de suivre les instructions affichées à l'écran.



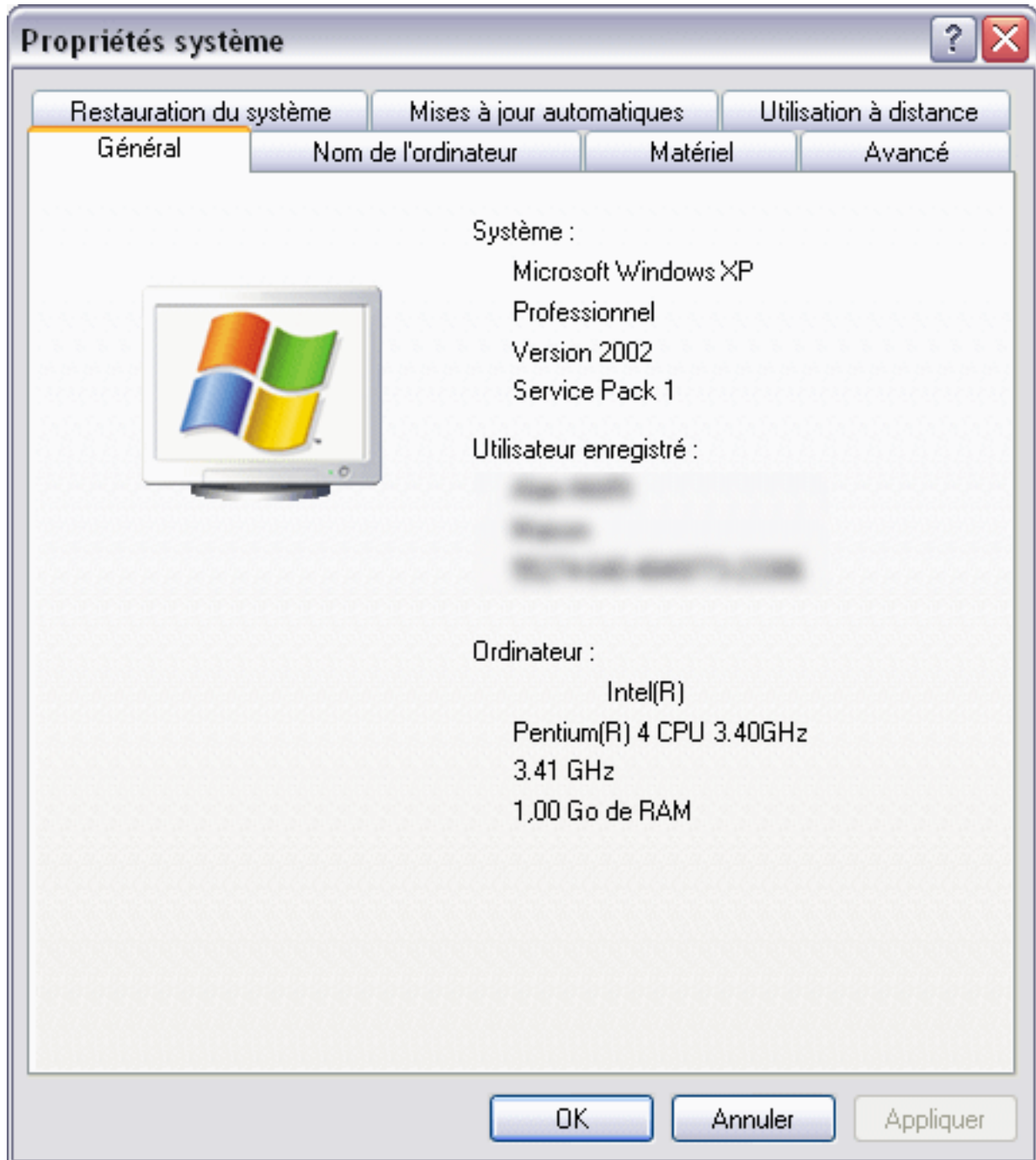
Pour un bon fonctionnement des applications 3D, il est indispensable d'installer Java 3D dans le même répertoire que celui de Java 2. En principe, l'installeur automatique choisit par défaut le répertoire où Java 2 a été installé, mais une vérification s'impose quand même !

L'installation de la version 1.3.2 est un peu plus compliquée car il n'y a pas d'installeur automatisé. Il faut décompresser l'archive `java3d-1_3_2-windows-i586.zip` (Windows) dans un répertoire quelconque, puis décompresser l'archive `j3d-132-win.zip` dans le répertoire `C:\...\j2sdk1.4.2_06\jre` (`C:\...\j2sdk1.4.2_06` étant le répertoire d'installation de Java 2). En ce qui concerne Linux, la démarche est identique en tout point.

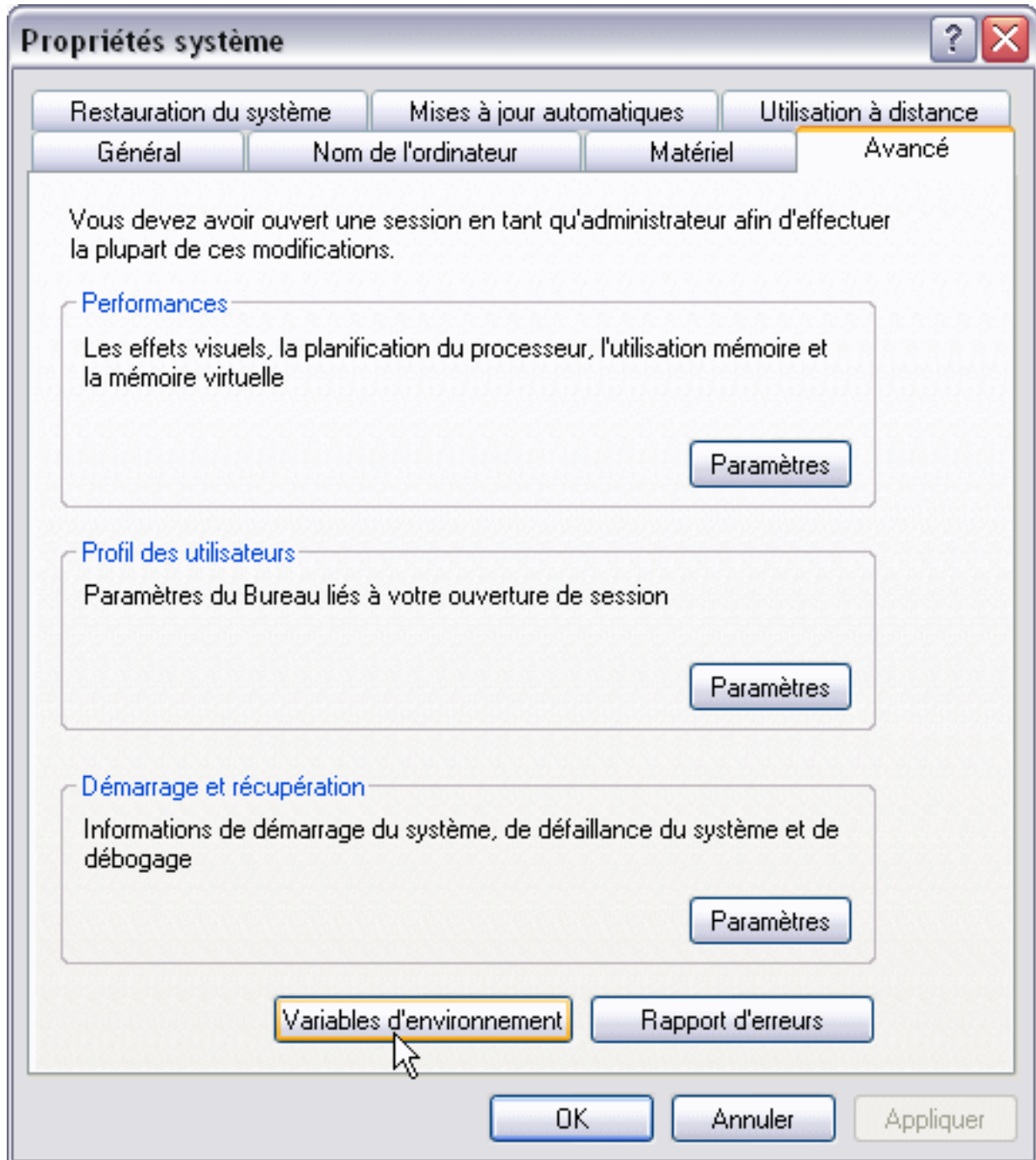
Une fois Java 2 et Java 3D correctement installés, il est nécessaire de mettre à jour la variable PATH de Windows (ou de Linux).

1.3 - Mise à jour du PATH de Windows

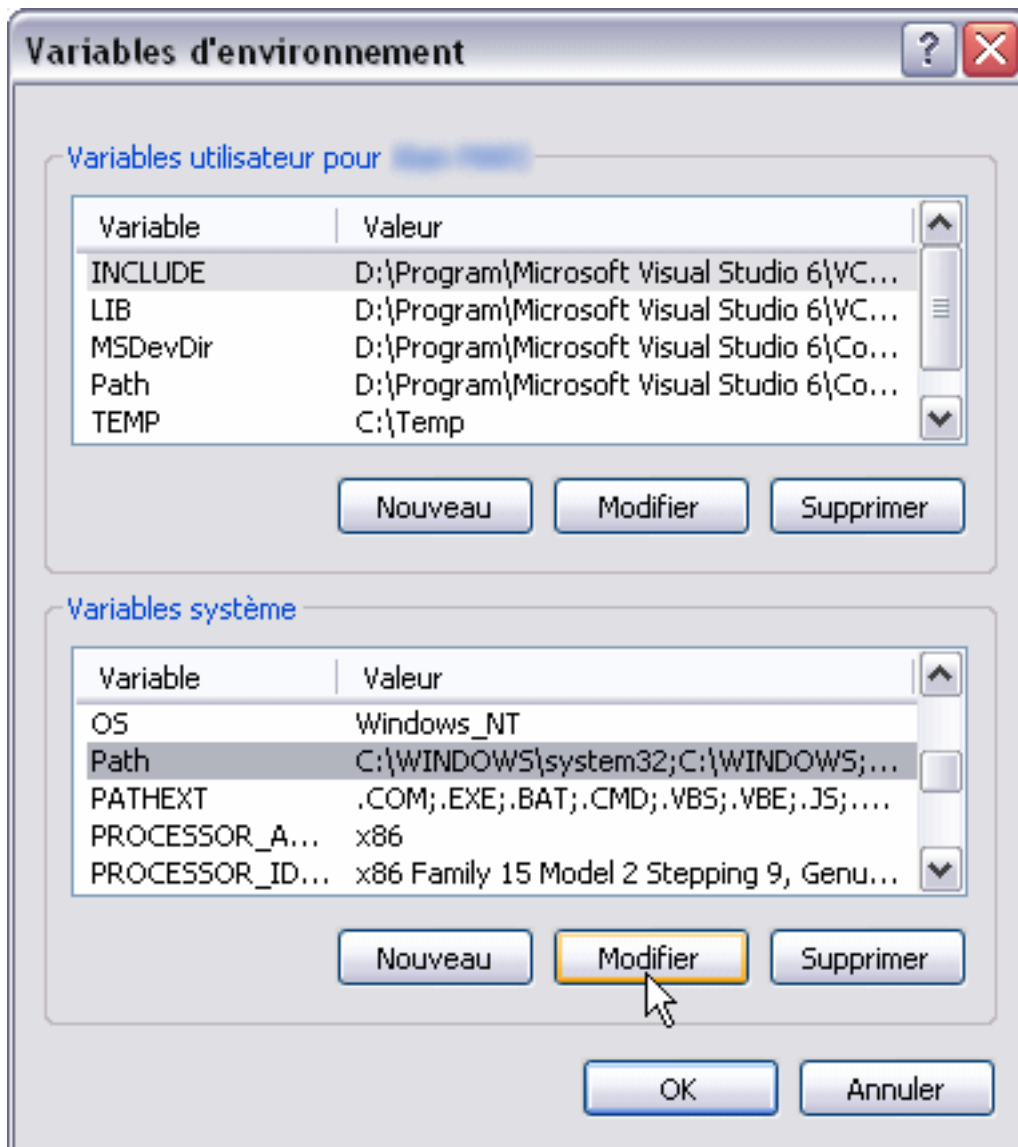
Cliquez avec le bouton droit sur Poste de travail, puis sélectionnez le menu Propriétés. La fenêtre Propriétés système apparaît.



Choisissez l'onglet Avancé puis cliquez sur le bouton Variables d'environnement.



Dans la liste déroulante intitulée Variables système, sélectionner Path puis cliquez sur le bouton Modifier.



Dans la zone de texte Valeur de la variable, rajouter au début le chemin complet vers l'exécutable java.exe que l'on vient juste d'installer en le faisant suivre d'un point-virgule.

Voici un exemple de ligne à rajouter dans le PATH :

```
C:\java\jdk1.4.2_06\jre\bin;
```

Cliquez sur le bouton OK des boîtes de dialogue restées ouvertes pour valider la modification (il n'est pas nécessaire de redémarrer le système pour que la modification soit prise en compte).

2 - Un premier exemple avec Java3D

A ce stade, tous les composants requis pour la création d'applications 3D avec Java ont été installés. Nous allons procéder pas à pas en détaillant bien chaque étape pour la création de notre premier programme 3D. Cet exemple se présente à la fois comme une application indépendante et comme un applet que l'on peut exécuter au sein d'un navigateur internet. Il s'agit de construire une petite animation représentant un cube en couleur tournant autour d'un axe vertical. Le code source complet de cette exemple se trouve dans le fichier [ColorCube3D.java](#).

2.1 - Visite guidée du code source ColorCube3D.java de notre premier exemple

Importation des packages Java 2 de base permettant de construire les bases de notre application.

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

Importation des packages Java 3D permettant d'utiliser les fonctions 3D

```
import com.sun.j3d.utils.applet.MainFrame;  
import com.sun.j3d.utils.geometry.ColorCube;  
import com.sun.j3d.utils.universe.*;  
import javax.media.j3d.*;
```

Ensuite, nous passons à l'écriture du constructeur de notre classe ColorCube3D.

Création d'un Canvas3D

La classe Canvas3D dérive de la classe Canvas de l'Abstract Windowing Toolkit (AWT) de Java 2. Elle sert en fait à représenter des objets 3D dans une fenêtre à deux dimensions qui est celle affichée à l'écran. Le constructeur de la classe Canvas3D a besoin d'un objet qui fournit les caractéristiques de la destination graphique dans laquelle nous allons visualiser notre application, c'est à dire l'écran de notre ordinateur dans le cas présent. Cet objet est fourni par la classe utilitaire SimpleUniverse.

```
Canvas3D canvas3D = new Canvas3D(SimpleUniverse.getPreferredConfiguration());
```

Création d'un univers 3D permettant d'intégrer le canvas3D que nous venons de créer

```
SimpleUniverse simpleU = new SimpleUniverse(canvas3D)
```

Positionnement d'un point d'observation et du canvas 3D pour avoir une vue d'ensemble correcte de la scène 3D

```
simpleU.getViewingPlatform().setNominalViewingTransform()
```

Création de la scène 3D contenant tous les objets que l'on veut visualiser

C'est la méthode createSceneGraph() de notre classe ColorCube3D qui va créer tous les objets 3D. Il s'agit d'abord de créer un objet "maître" ou "parent" auquel nous allons rattacher le cube 3D. Cet objet est une instance de la classe BranchGroup.

```
BranchGroup parent = new BranchGroup()
```

Ensuite, nous allons lui ajouter le cube 3D en couleur. Java 3D fournit une classe prête à l'emploi permettant de créer des couleurs : `ColorCube`.

```
parent.addChild(new ColorCube(0.4));
```

L'argument 0.4 est un facteur d'échelle afin que le cube puisse être vu en entier dans la fenêtre de notre application. Nous avons volontairement occulté la partie de cet exemple permettant d'animer ce cube.

Vous vous reporterez aux chapitres Transformations géométriques et Animation (en construction) pour plus de détails sur la manière de gérer les transformations et les animations.

Compilation de la scène

Une fois la scène créée par la méthode `createSceneGraph()`, il faut la compiler afin de la transformer en une série d'instructions très performantes qui seront envoyées au moteur de rendu de la carte graphique chargé de l'affichage.


```
scene.compile();
```

Ajouter la scène 3D à l'univers 3D

C'est la dernière étape de la création de notre application 3D

```
simpleU.addBranchGraph(scene);
```

Ecriture de la méthode `main()`

 Pour exécuter une applet au sein d'un navigateur Internet, la méthode `main()` n'est pas requise, il suffit de surcharger la méthode `init()` de la classe `Applet`. Cependant, le fait de rajouter cette méthode nous permettra d'exécuter notre classe à la fois comme une applet et comme une application indépendante.

```
public static void main(String[] args) {
    Frame frame = new MainFrame(new ColorCube3D(), 256, 256);
}
```

Le fichier [ColorCube3D.java](#) présente le code source complet de ce premier exemple :

```
// Etape 1 :
// Importation des packages Java 2
import java.applet.Applet;
import java.awt.*;

// Etape 2 :
// Importation des packages Java 3D
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.geometry.ColorCube;
import com.sun.j3d.utils.universe.*;
import javax.media.j3d.*;

public class ColorCube3D extends Applet {

    public ColorCube3D() {
        this.setLayout(new BorderLayout());

        // Etape 3 :
        // Creation du Canvas 3D
        Canvas3D canvas3D = new Canvas3D(SimpleUniverse.getPreferredConfiguration());
```

```

    this.add(canvas3D, BorderLayout.CENTER);

    // Etape 4 :
    // Creation d'un objet SimpleUniverse
    SimpleUniverse simpleU = new SimpleUniverse(canvas3D);

    // Etape 5 :
    // Positionnement du point d'observation pour avoir une vue correcte
    // de la scene 3D
    simpleU.getViewingPlatform().setNominalViewingTransform();

    // Etape 6 :
    // Creation de la scene 3D qui contient tous les objets 3D que l'on
    // veut visualiser
    BranchGroup scene = createSceneGraph();

    // Etape 7 :
    // Compilation de la scene 3D
    scene.compile();

    // Etape 8:
    // Attachement de la scene 3D a l'objet SimpleUniverse
    simpleU.addBranchGraph(scene);
}

/**
 * Creation de la scene 3D qui contient tous les objets 3D
 * @return scene 3D
 */
public BranchGroup createSceneGraph() {
    // Creation de l'objet parent qui contiendra tous les autres objets 3D
    BranchGroup parent = new BranchGroup();

    /***** Partie de code concernant l'animation du cube *****/
    /* Elle sera expliquée en details dans les chapitres relatifs aux
       transformations geometriques et aux animations */
    TransformGroup objSpin = new TransformGroup();
    objSpin.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    Alpha rotationAlpha = new Alpha(-1, 4000);
    RotationInterpolator rotator = new RotationInterpolator(rotationAlpha, objSpin);
    BoundingSphere bounds = new BoundingSphere();
    rotator.setSchedulingBounds(bounds);
    objSpin.addChild(rotator);
    /***** Fin de la partie relative a l'animation *****/

    // Construction du cube couleur
    objSpin.addChild(new ColorCube(0.4));
    parent.addChild(objSpin);

    return parent;
}

/**
 * Etape 9 :
 * Methode main() nous permettant d'utiliser cette classe comme une applet
 * ou une application.
 * @param args
 */
public static void main(String[] args) {
    Frame frame = new MainFrame(new ColorCube3D(), 256, 256);
}
}

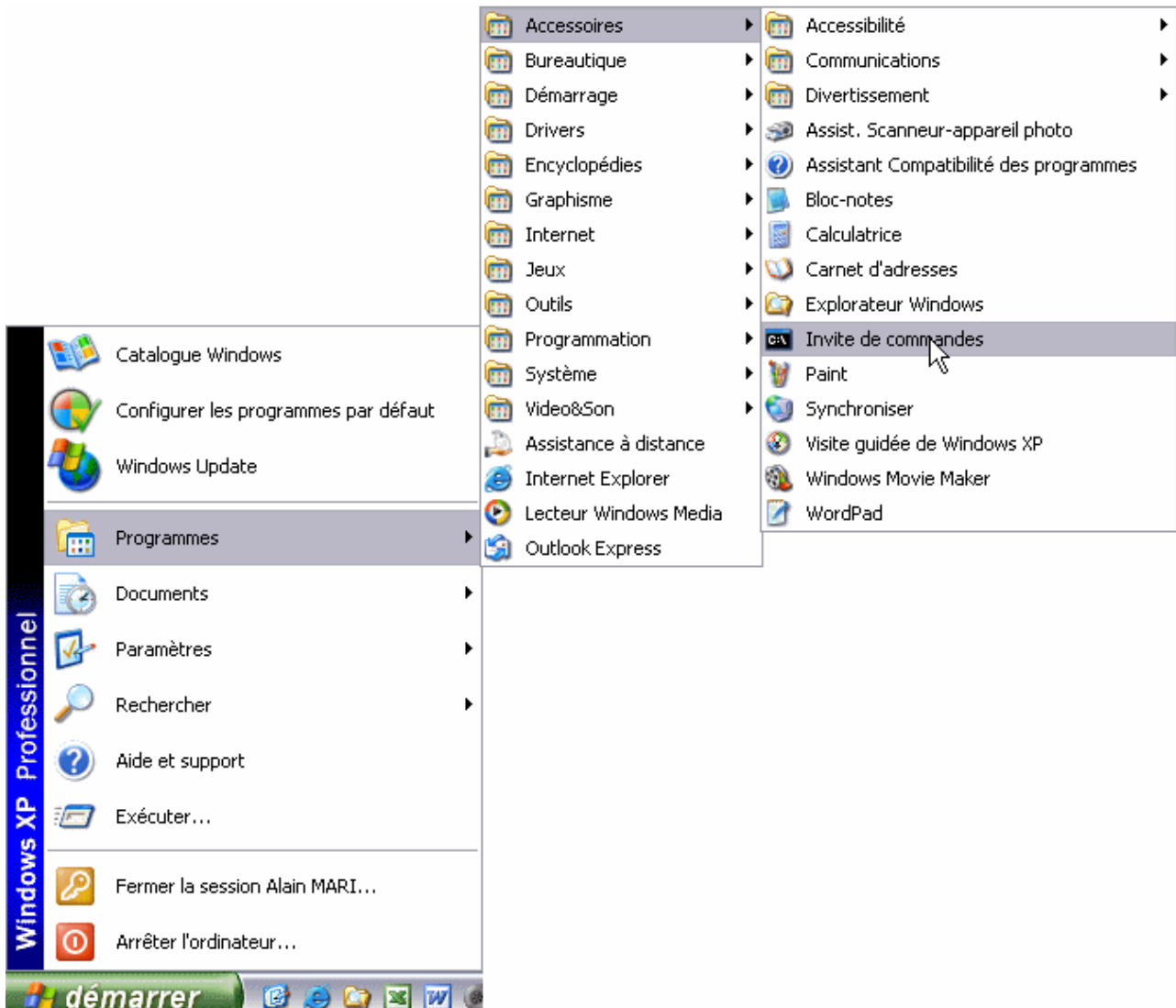
```

2.2 - Exécution du programme

Avant de pouvoir exécuter ce programme, il faut le compiler.

Pour cela, cliquer sur le menu Démarrer puis sélectionner Programmes et Accessoires.

Sélectionner ensuite le menu Invite de commandes.



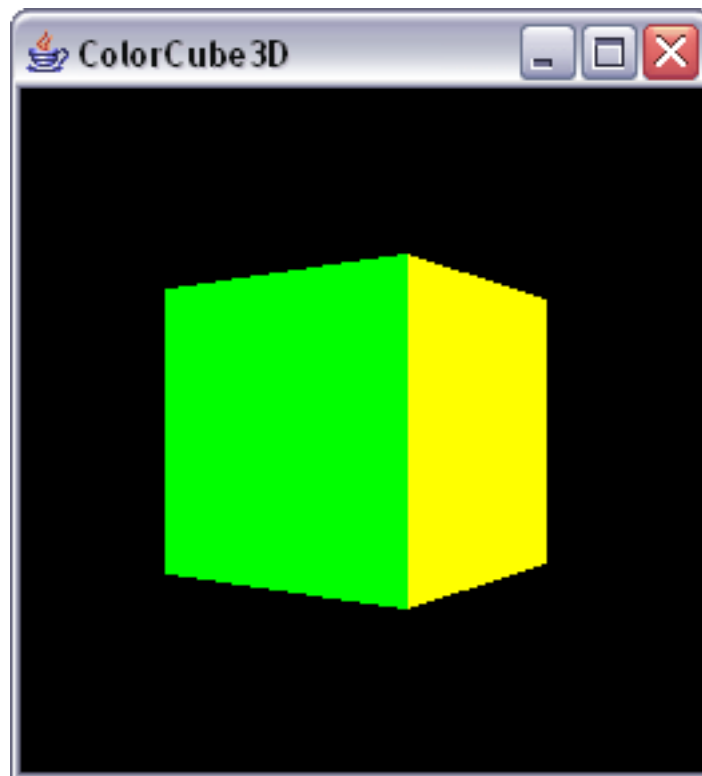
Dans la ligne de commandes, positionnez - vous là où se trouve le fichier ColorCube3D.java puis lancez la commande `javac ColorCube3D.java`. Un fichier ColorCube.class est alors automatiquement créé.

Pour exécuter le programme en tant qu'application indépendante, il suffit de lancer la commande : `java ColorCube3D`.

Pour exécuter le programme en tant qu'applet avec un navigateur web, il suffit de créer un fichier .html avec une balise APPLET comme suit :

```
<APPLET CODE="ColorCube3D.class" WIDTH=256 HEIGHT=256>
</APPLET>
```

Ouvrir ensuite le fichier [ColorCube3D.html](#) avec votre navigateur préféré et le cube en couleurs animé apparaîtra aussitôt :



Vous pouvez aussi exécuter l'application avec Java Web Start (cela évite d'installer Java 3D au préalable) en répondant sans crainte "oui" aux éventuels avertissements de sécurité : [ColorCube3D.jnlp](#)

3 - Utilisation de Java 3D avec JBuilder

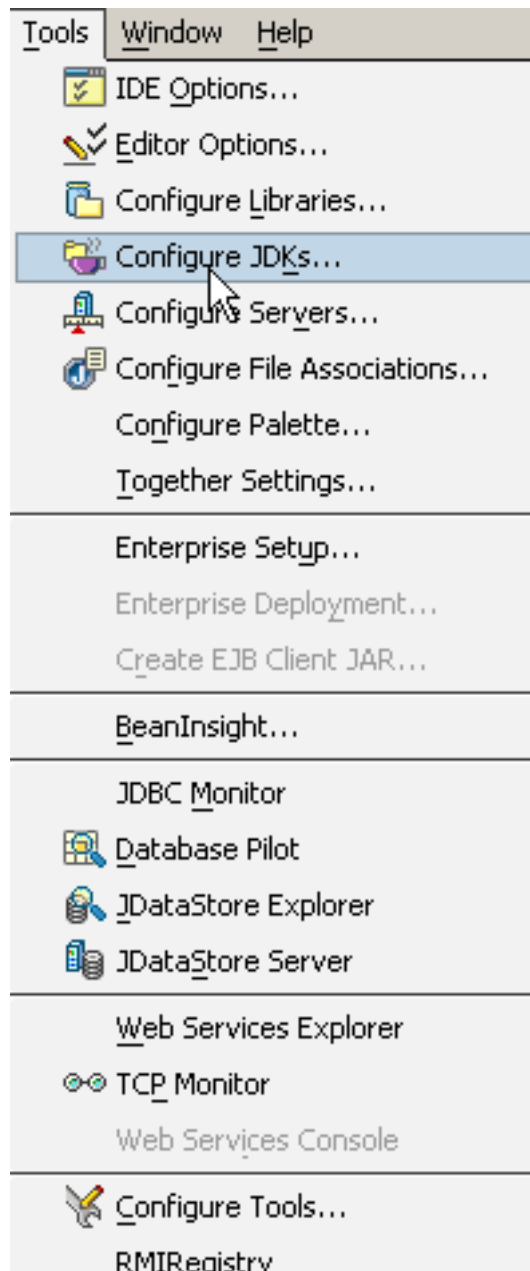
Comme nous l'avons vu dans le paragraphe précédent, tous les exemples de ce chapitre peuvent être écrits à l'aide d'un simple éditeur de texte comme le bloc notes de Windows, compilés grâce à la console Windows, puis exécutés en ligne de commande ou à l'aide d'un navigateur internet. Cependant, toutes ces étapes peuvent se révéler fastidieuses à la longue surtout pour les exemples les plus longs.

Aussi nous proposons d'intégrer Java 2 et Java 3D dans un environnement de développement, c'est-à-dire un logiciel capable de détecter les fautes de syntaxe en cours de frappe, de compiler et d'exécuter automatiquement les applications en un simple clic de souris. Un des plus utilisés et conviviaux est sans doute JBuilder de la société Borland dont une version d'essai pleinement fonctionnelle pendant 30 jours peut être téléchargée à l'adresse suivante : www.borland.com/products/downloads/download_jbuilder.html

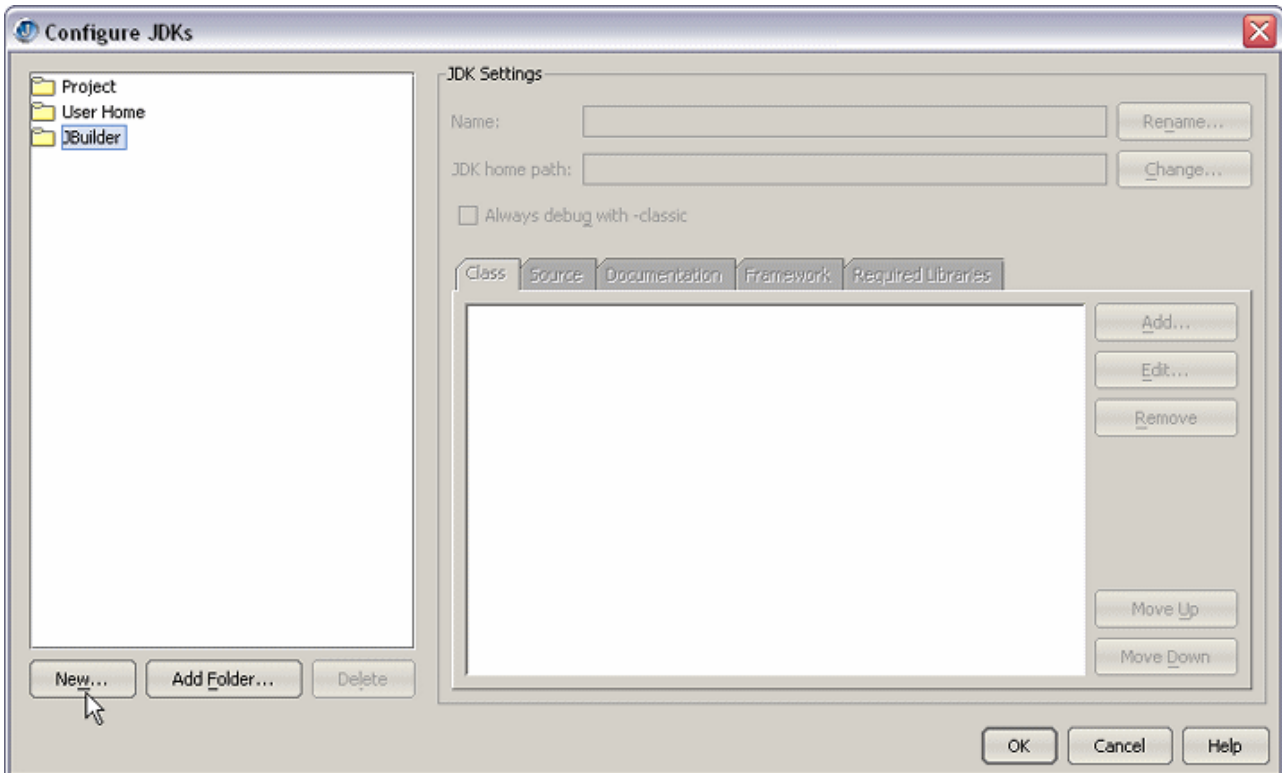
Après s'être armé de patience et une fois l'installation de JBuilder terminée, il s'agit de le configurer de telle sorte qu'il prenne correctement en compte les librairies Java 2 et 3D installées sur le système.

Les étapes suivantes décrivent la marche à suivre avec la version 9 de JBuilder.

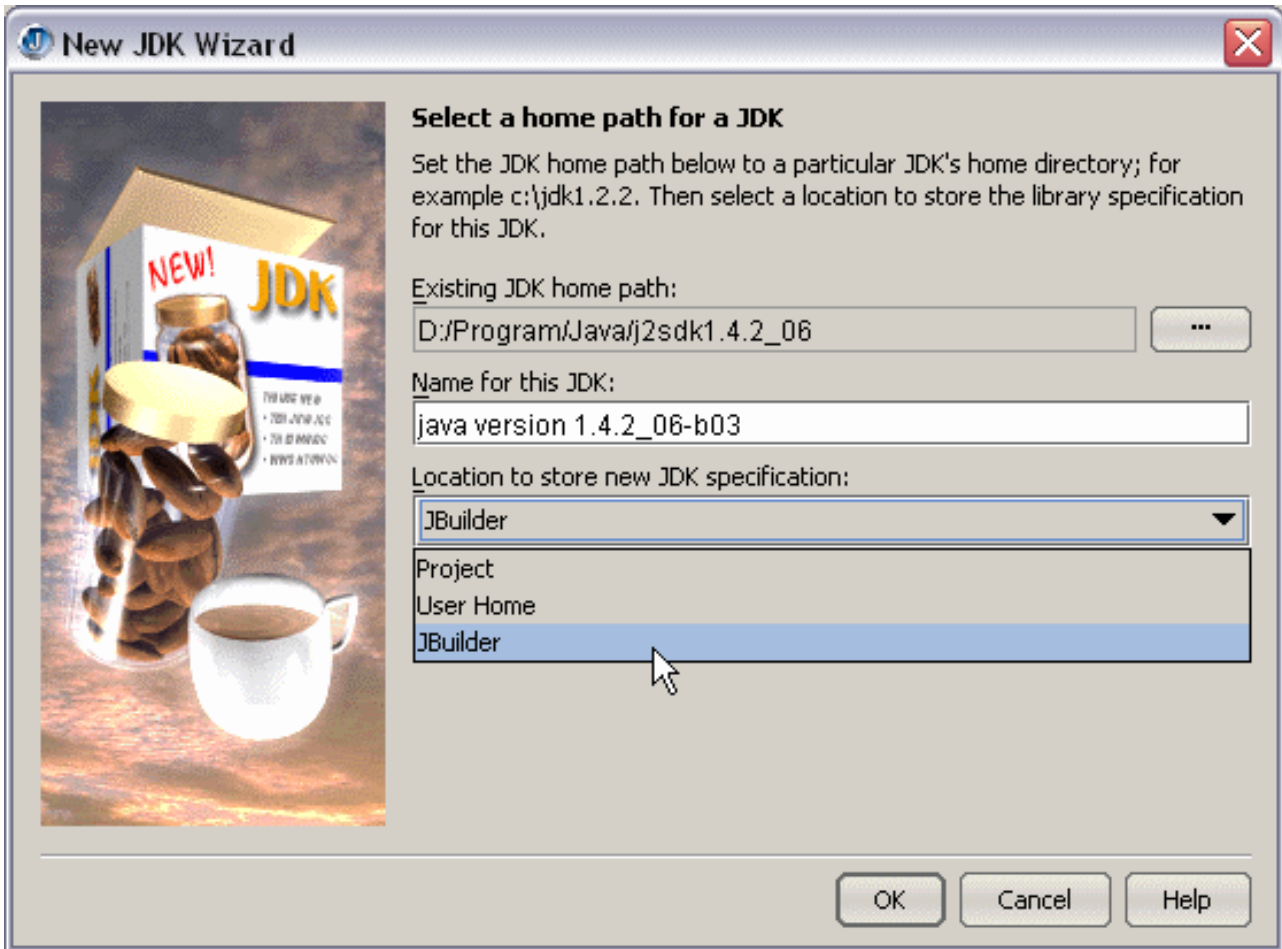
Exécutez JBuilder, cliquez sur le menu Tools, puis choisissez Configure JDKs.



Dans la colonne de gauche de la boîte de dialogue qui apparaît, sélectionnez JBuilder puis cliquez sur le bouton New.



Cliquez sur le bouton ... et choisir le répertoire d'installation de Java 2 puis sélectionnez l'option JBuilder dans la liste déroulante Location to store new JDK specification.

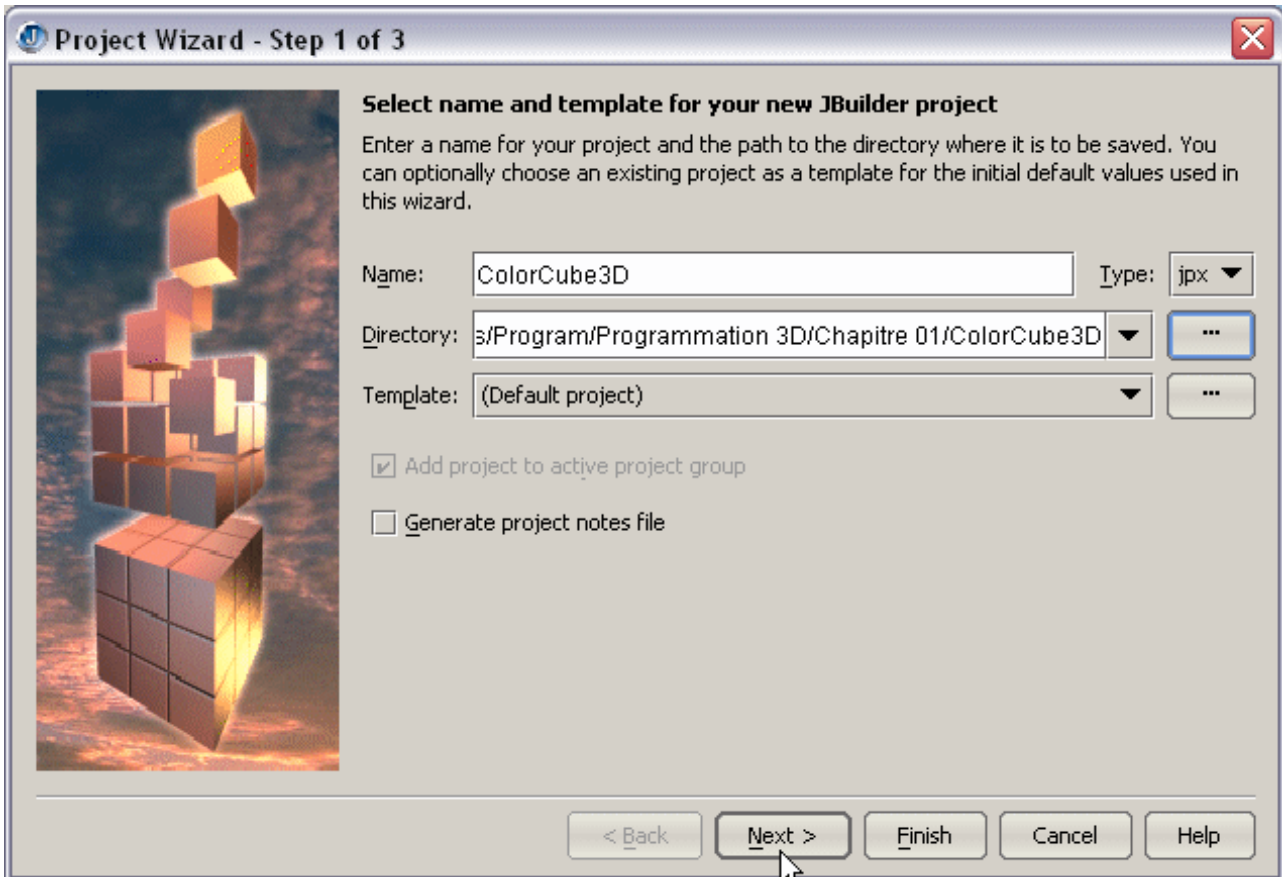


Pour valider, cliquez sur le bouton OK des boîtes de dialogues restées ouvertes.

L'opération est terminée, comme Java 3D a été installé dans le même répertoire que Java 2, la librairie 3D est elle aussi automatiquement intégrée à JBuilder. Désormais vous pouvez écrire et exécuter tous les exemples de ce chapitre en profitant de la convivialité de ce très bon produit.

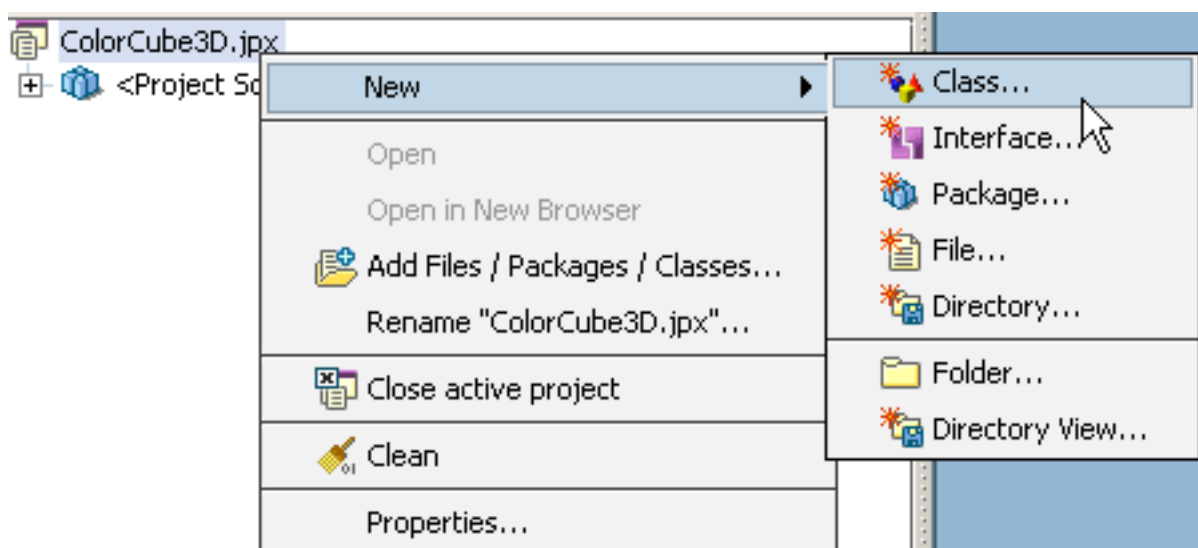
Reprenons notre premier exemple ColorCube3D.java et exécutons le grâce à JBuilder.

Exécutez JBuilder, cliquez sur le menu File puis choisissez New Project... afin de créer un projet de développement vide. Nommez le ColorCube3D dans le répertoire de votre choix pour cliquez sur le bouton Next pour passer à l'étape suivante



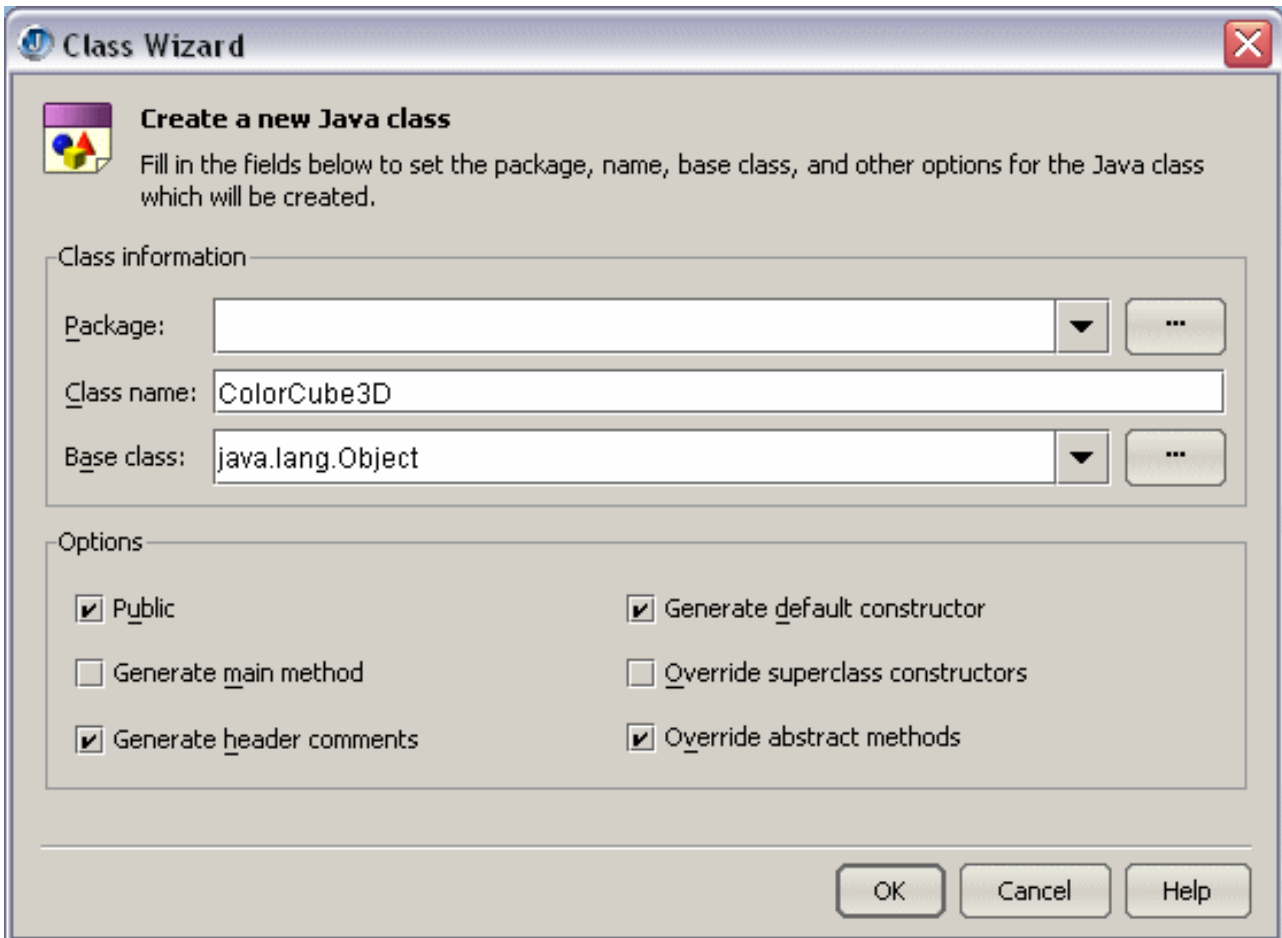
Appuyez deux fois consécutives sur le bouton Next en laissant les réglages par défaut pour les deuxièmes et troisièmes étapes de la création du projet, puis appuyer sur le bouton Finish pour valider le projet.

Dans la colonne de gauche, cliquez avec le bouton droit de la souris sur le nom du projet ColorCube3D.jpx puis choisissez New puis Class ... afin de créer un fichier vide.



Dans la zone de texte Class name, entrez ColorCube3D, ce sera le nom de la classe principale de démarrage du

programme. Veillez à laisser vide la zone de texte relative au Package.



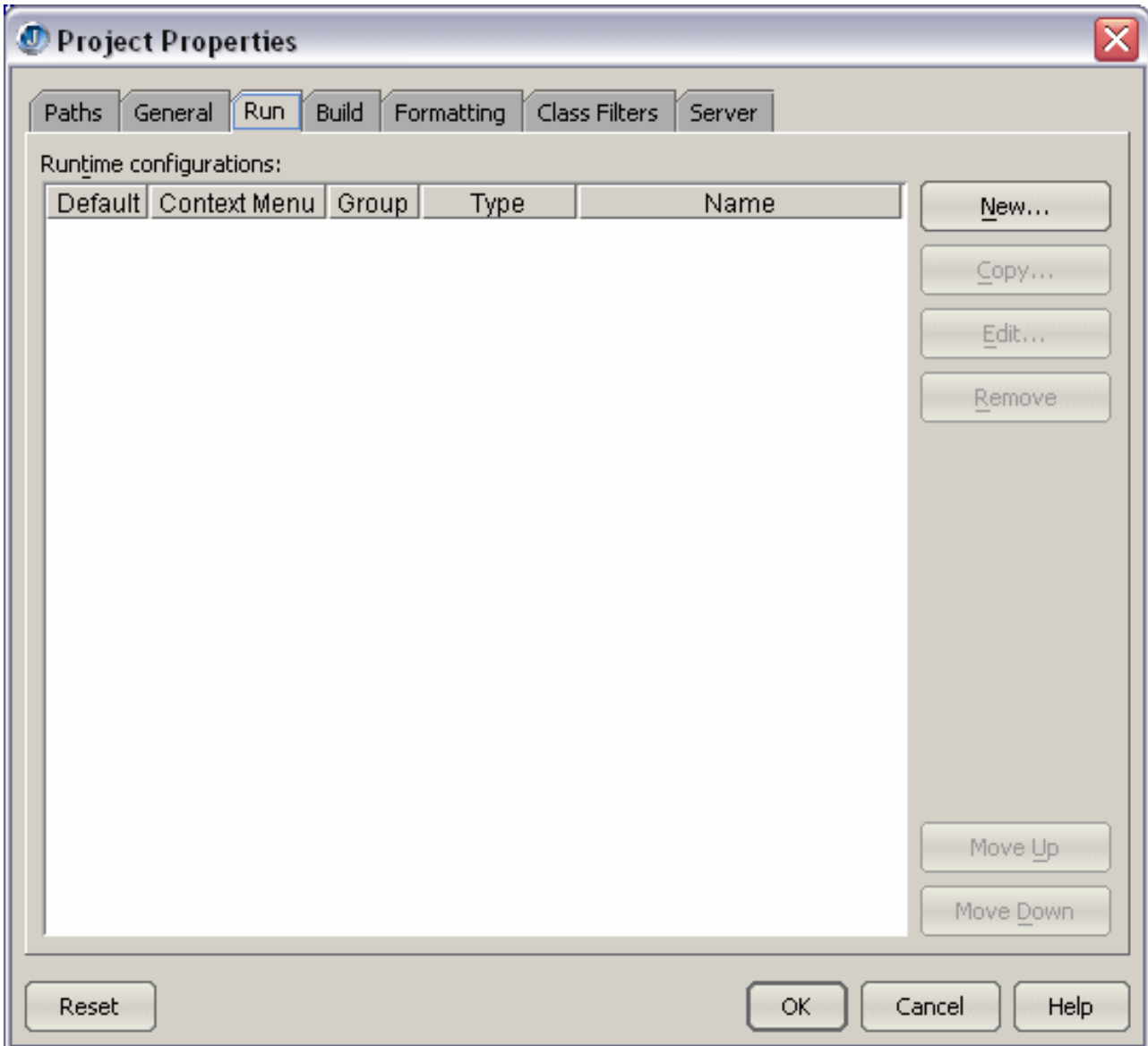
Cliquez sur le bouton OK pour valider la création de la nouvelle classe.

JBuilder fournit alors automatiquement le code source de la nouvelle classe avec un constructeur par défaut. Il suffit de remplacer ce code source par celui du fichier exemple ColorCube3D.java avec un simple copier-coller.

Appuyez sur F9 pour lancer le programme

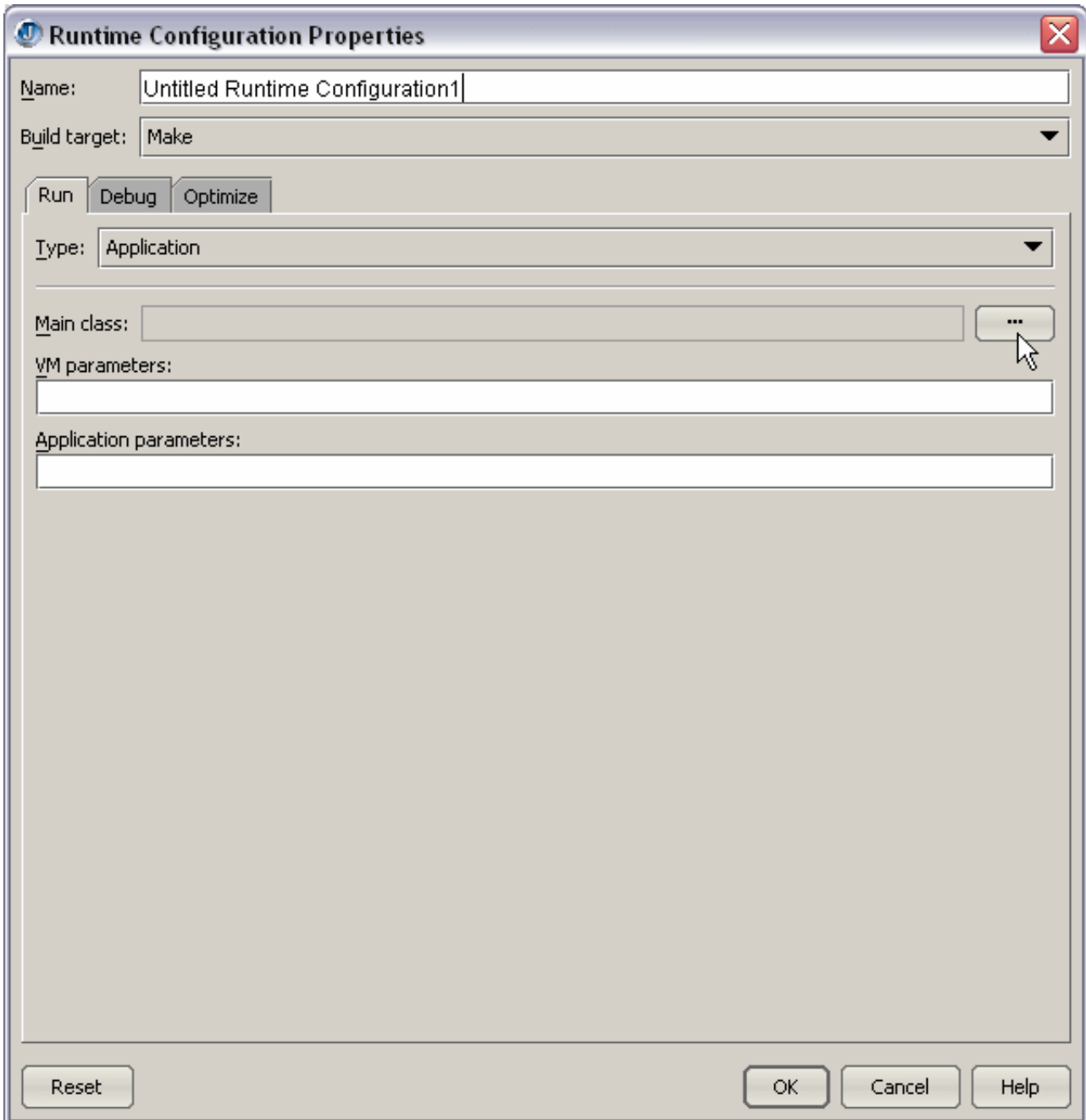
A ce moment là, on peut penser que c'est gagné et que la fenêtre 3D va immédiatement apparaître sous nos yeux, mais il reste juste un dernier détail à régler afin que tout se passe bien.

En effet, JBuilder affiche une fenêtre intitulée Project Properties qui va nous permettre de rentrer le nom de la classe de démarrage qui va être lancée par la machine virtuelle Java. Un même projet peut contenir plusieurs fichiers sources .java possédant chacun une classe de démarrage (c'est-à-dire une classe avec une méthode statique main()) afin de pouvoir exécuter un programme avec des configurations de lancement différentes. Les exemples Java décrits dans ce livre posséderont toujours une classe de démarrage unique afin de simplifier la compréhension du code source.



Cliquez sur le bouton New, la fenêtre de choix de la classe de démarrage apparaît.

Cliquez sur le bouton ... situé à droite de la zone de texte Main class.



Une liste de classes de démarrage utilisables par JBuilder apparaît, il suffit de sélectionner ColorCube3D puis de cliquer sur OK dans les boîtes de dialogues restées ouvertes afin de valider le choix.

Appuyez de nouveau sur F9, et l'application 3D devrait s'exécuter sans problème.

Nous avons eu besoin de 9 étapes pour créer une configuration JBuilder capable d'exécuter notre exemple, ce qui peut paraître un peu long. Cependant, en cas de modification du code source, toute erreur de syntaxe serait détectée en cours de frappe avec une aide à la saisie pour le nom des méthodes applicables à chaque objet. Ensuite, un simple appui sur la touche F9 se charge de compiler et de lancer le programme en une seule passe, le gain de temps n'est alors plus négligeable par rapport à la console et au bloc notes Windows !

4 - En préparation (Eclipse, chap 2, ...)

- Complément à ce premier chapitre : Utilisation de Java 3D avec Eclipse
- Chapitre 2 : Les bases de la construction d'une application 3D

5 - Téléchargements

[Article au format PDF \(mirroir\)](#)

[Article HTML/ZIP \(mirroir\)](#)

[Codes sources \(Java, Applet, JNLP, etc.\) \(mirroir\)](#)