

JavaOne Tokyo Hands on Lab

Project Looking Glass (LG3D) プログラミング

このテキストは JavaOne Tokyo 2005 で行われた Project Looking Glass (LG3D) ハンズオンの内容に修正・加筆を行ったものです。

また、ベースとなる LG3D のバージョンを LG3D Release 0.7.0 から LG3D Release 0.7.1 に変更しました。 LG3D のバージョン変更に伴う内容の変更はありません。

このテキストでは LG3D を利用するための準備、LG3D のインストール方法、起動方法 および NetBeans 4.1 によるプログラミング方法を解説しています。

1. グラフィックス環境のセットアップ

1. [Solaris 10 x86 編](#)
2. [Linux & NVIDIA 編](#)
3. [Linux & ATI 編](#)

2. LG3D のインストールと実行

1. [インストール \(Solaris 10 x86 編 \)](#)
2. [インストール \(Linux 編 \)](#)
3. [実行方法](#)
4. [インストールと実行方法 \(Windows XP 編 \)](#)

3. NetBeans による LG3D プログラミング (準備)

1. [NetBeans 4.1 のダウンロードおよびインストール](#)
2. [NetBeans の設定](#)

4. NetBeans による LG3D プログラミング (実践)

1. [プロジェクトの作成](#)
2. [シンプルな 3D オブジェクトの作成](#)
3. [テクスチャを使う](#)
4. [サムネイルの作成](#)
5. [マウスイベント・アニメーションの作成](#)
6. [JAR ベースアプリケーションの作成](#)

5. 応用問題

LG3D ハンズオンで課題が終了した方にお渡ししたものです。

答えは用意していません。

- [参考文献, URL 等](#)

注意: このテキストでのユーザ環境は次の通りです。

ユーザ名	duke
ホームディレクトリ	/home/duke
LG3D ディレクトリ	/home/duke/lg3d
JDK ディレクトリ	/home/duke/lg3d/jdk1.5.0_05
データディレクトリ	/home/duke/Documents
NetBeans ディレクトリ	/opt/netbeans-4.1
画面解像度	1280x1024
LG3D ウィンドウサイズ	1152x864

グラフィックス環境のセットアップ (Solaris 10 x86 編)

LG3D を Solaris 10 x86 で利用するためには必要な環境は次の通りです。

- Pentium 4, Athlon 64 のような比較的新しい CPU (1.0 GHz 以上推奨)
- 512MB 以上のメモリ (利用環境によっては 1GB 以上推奨)
- NVIDIA Quadro シリーズ (GeForce FX シリーズも利用できます)

ドライバの確認

システムに NVIDIA のドライバが入っているかどうかは pkginfo コマンドで調べることができます。以下のように、実行した際にも表示されなければ NVIDIA のドライバはインストールされていません。

```
# pkginfo | grep NVDA
```

ドライバのインストール

NVIDIA のサイト(<http://www.nvidia.com/object/unix.html>) からドライバをダウンロードします。

ドキュメント記述時(2005/10)の最新版のドライバは 1.0-7676 (NVIDIA-Solaris-x86-1.0-7676.run) です。ドライバはインストールはルートユーザで行います。インストール方法は次の通りです。

```
# sh NVIDIA-Solaris-x86-1.0-7676.run
```

```
A sample xorg configuration file has been copied to /etc/X11/xorg.conf.nvidia
Copyright 2005 by NVIDIA Corporation. All rights reserved.
Use is subject to license terms.
```

```
Installation of <NVDAgraphicsr> was successful.
Copyright 2005 by NVIDIA Corporation. All rights reserved.
Use is subject to license terms.
```

```
Installation of <NVDAgraphics> was successful.
```

インストールが終了するとシステムに 2 つのパッケージ(NVDAgraphics,NVDAgraphicsr)が追加されます。以下のように pkginfo コマンドで確認できます。

```
# pkginfo | grep NVD
system    NVDAgraphics          NVIDIA Graphics System Software
system    NVDAgraphicsr         NVIDIA Graphics System Device Driver
```

ドライバのインストールが終了したら X-Window 環境の設定を行います。 (このままでは X-Window は起動しません)

NVIDIA 製ドライバをインストールすると /etc/X11/xorg.conf.nvidia というファイルがインストールされますので、これを利用します。このファイルは NVIDIA 製ドライバ環境での xorg.conf のサンプルファイルです。ほとんどの環境ではこのファイルを /etc/X11/xorg.conf にコピーするだけで良いです。

```
# cp /etc/X11/xorg.conf.nvidia /etc/X11/xorg.conf
```

xorg.conf をコピー後に編集し AllowGLXWithComposite オプションを有効にします。

LG3Dには lg3d-dev, lg3d-app, lg3d-session の 3つのコマンドが用意されていますが、lg3d-session コマンドを利用するためには必要なオプションです。

```
# Device configured by xorgconfig:  
  
Section "Device"  
    Identifier  "** NVIDIA (generic)          [nv]"  
    Driver      "nvidia"  
    #VideoRam   131072  
    # Insert Clocks lines here if appropriate  
    Option      "AllowGLXWithComposite" "yes"  
EndSection
```

UXGA を利用している場合などは /usr/X11/bin/xorgconfig を用いて /etc/X11/xorg.conf を生成します。その後 xorg.conf を編集し、NVIDIA 製ドライバ用の設定を行ってください。

インストール後、リブートします。

```
# init 6
```

NVIDIA Quadro シリーズを利用している場合は NVIDIA のロゴが表示され、ログイン画面が表示されると思います。ドライバのインストールおよび設定は終了です。

NVIDIA GeForce FX シリーズを利用している場合は GUI の表示に失敗します。

NVIDIA 製ドライバのインストール時点では NVIDIA GeForce シリーズのカード情報が Solaris に登録されないのが原因です。

NVIDIA 製ドライバを有効にするためには グラフィックスカード情報を登録します。
登録方法は次の通りです。

- /usr/X11/bin/scanpci コマンドを実行します。PCI 情報を表示されるので、グラフィックスカードの vendor, device の値を確認します (下図の青字(下線)の部分)。
vendor, device の値のうち「0x」と4桁の数値の最初の「0」は無視します。
つまり、例では vendor = 10de, device = 322 となります。
- update_drv コマンドでグラフィックスカード情報を登録します。
- リコンフィグオプション付きでリブートします。

```
# /usr/X11/bin/scanpci  
  
...  
  
pci bus 0x0000 cardnum 0x1f function 0x05: vendor 0x8086 device 0x24d5  
Intel Corp. 82801EB AC'97 Audio Controller  
  
pci bus 0x0001 cardnum 0x00 function 0x00: vendor 0x10de device 0x322  
nVidia Corporation NV34 [GeForce FX 5200]  
  
pci bus 0x0002 cardnum 0x01 function 0x00: vendor 0x8086 device 0x1019  
Intel Corp. 82547EI Gigabit Ethernet Controller (LOM)  
  
...  
# update_drv -a -i "pci10de,322" nvidia  
# reboot -- -r
```

グラフィックス環境のセットアップ(Linux & NVIDIA 編)

LG3D を動かすためには 24bit color, OpenGL 1.3 以上をサポートした環境が必要です。Linux 上でこの環境を構築するには以下のグラフィックスチップを搭載した PC が必要です(NVIDIA 製のグラフィックスチップの場合)。

- NVIDIA GeForce シリーズ
- NVIDIA Quadro FX シリーズ

ドライバのインストール

[NVIDIA のサイト](http://www.nvidia.com/object/unix.html)(<http://www.nvidia.com/object/unix.html>) から NVIDIA 製のグラフィックスドライバをダウンロードします。ドキュメント記述時(2005/10)の最新版のドライバは 1.0-7676(NVIDIA-Solaris-x86-1.0-7676.run)です。ここではファイルを /tmp にダウンロードしたと仮定します。

ドライバはインストールはルートユーザで行います。

インストール方法は次の通りです。 (ユーザーの入力したコマンドは赤(太字)で示します)

1. ルートユーザになりコンソールモードに変更する。 コンソールモードに変わったら再度ログインし、インストーラを起動します。
対話式のインストーラが表示されるので、指示に従いインストールします。

```
> su -  
Password: ← パスワードを入力  
# init 3  
  
console login: root  
Password: ← パスワードを入力  
...  
# sh NVIDIA-Linux-x86_64-1.0-7676-pkg2.run
```

補足:

Fedora Core 4 等を利用している場合、NVIDIA 製のドライバをインストールしてもうまく動かせない場合があります。

そのような場合は、<http://rpm.livna.org/>にある RPM 版(nvidia-glx-1.0.7676-0.lvn.2.4.i386.rpm)を利用してください。
<http://rpm.livna.org/fedora/4/>以下にがあります。

インストール方法はルートユーザで

```
# rpm -ihv nvidia-glx-1.0.7676-0.lvn.2.4.i386.rpm
```

とします。

インストールには依存関係のために他の RPM バイナリが必要な場合があります。

2. ドライバのインストールが終了したら X-Window 環境の設定を行います。 以下のように /etc/X11/xorg.conf を編集します。

赤(太字)は変更部分に関するコメントです。これを参考に変更してください。

```
...(略)...  
Section "Module"          [モジュールの設定]  
    Load      "glx"  
    # Load     "dri"  
    # Load     "GLcore"  
EndSection  
...(略)...  
Section "InputDevice"      [キーボード設定]  
    Identifier "Keyboard1"  
    Driver     "kbd"          ["keyboard"の場合、kbdに変更]  
    Option    "AutoRepeat" "500 30"  
    Option    "XkbRules"   "xfree86"  
    Option    "XkbModel"   "jp106"  
    Option    "XkbLayout"  "jp"  
EndSection  
...(略)...  
Section "Device"           [グラフィックスドライバの設定]  
    Identifier "NVIDIA"  
    Driver     "nvidia"        ["nv"から変更]  
    Option    "AllowGLXWithComposite" "yes" [Ig3d-sessionを利用する場合は追加]  
EndSection  
...(略)...  
Section "Screen"            [スクリーン設定]  
    Identifier "Screen0"  
    Device     "NVIDIA"  
    Monitor   "Monitor0"  
    DefaultDepth 24          [無い場合は追加、24以外の場合は24に変更]  
    Subsection "Display"  
        Depth     24  
        Modes    "1280x1024"  
        ViewPort 0 0 # initial origin if mode is smaller than desktop  
    EndSubsection  
EndSection
```

3. Xを再起動します。

```
# init 5
```

グラフィックス環境のセットアップ(Linux & ATI編)

LG3D を動かすには 24bit color, OpenGL 1.3 以上をサポートした環境が必要です。
Linux の場合、以下の環境が必要となります。

- ATI Radeon 8500 以降

ドライバのインストール

ATI製グラフィックスドライバは ATIのウェブサイト（<http://www.ati.com/jp/index.html>）に公開されています。

トップページから「ドライバ」→「Linux Drivers and Software」→「Radeon 8500 Series and Higher」とリンクをたどっていくと見つかります。

ドキュメント記述時(2005/12)の最新版のドライバは 8.19.10 です。

ATI製のドライバは XFree86/Xorg のバージョン毎に用意されていますのでダウンロードを行う前に XFree86/Xorg のバージョンを確認しておきます。

```
# X -version
X Window System Version 6.8.2
Release Date: 18 December 2003
X Protocol Version 11, Revision 0, Release 6.8
Build Operating System: SuSE Linux [ELF] SuSE
Current Operating System: Linux iacus 2.6.5-7.183-default #1 Tue Jun 7 15:57:09
UTC 2005 i686
Build Date: 10 May 2005
Before reporting problems, check
http://www.sun.com/service/sunjavasystem/jdsserviceguide.html
      to make sure that you have the latest version.
Module Loader present
```

上記の実行結果より Xorg 6.8.2 ということがわかりました。ここでは Xorg 6.8 用のドライバ([fglrx_6_8_0-8.19.10-1.i386.rpm](#))を利用します。

ドライバはインストールはルートユーザで行います。

インストール方法は次の通りです。（ユーザーの入力したコマンドは赤(太字)で示します）

1. ルートユーザになりコンソールモードに変更する。コンソールモードに変わったら再度ログインします。

```
> su -
Password: ←パスワードを入力
# init 3

console login: root
Password: ←パスワードを入力
...
#
```

2. ドライバをインストールします。

```
# rpm -ihv fglrx_6_8_0-8.19.10-1.i386.rpm
```

3. xorg.conf(X-Window 環境の設定ファイル)を作成します。
fglrxconfig と aticonfig という2つのコマンドがあります。
fglrxconfig は対話形式のインストーラを起動するコマンドです。
aticonfig は自動的に xrog.conf を作成するコマンドです。
aticonfig --initial --input=/etc/X11/xorg.conf
のように実行します。

下の例は fglrxcofig を実行したときのものです。

```
# fglrxconfig
```

```
=====
=====
```

This program will create the ATI "xorg.conf" file
- based on your selections - for the following video cards...

- ATI Radeon 8500 / 9100
- ATI FireGL 8700 / 8800 / E1
- ATI FireGL T2
- ATI Radeon 9000
- ATI Radeon 9200
- ATI Radeon 9500
- ATI Radeon 9600
- ATI Radeon 9700
- ATI Radeon 9800
- ATI FireGL Z1 / X1 / X2
- ATI Mobility M9
- ATI Mobility FireGL 9000
- ATI Mobility M9PLUS
- ATI FireGL V3100 / V5100 / V7100

The "xorg.conf" file usually resides in /etc/X11.

Press [Enter] to continue, press 'q'&[Enter] or [Ctrl]+'c' to abort.

4. X を再起動します。

```
# init 5
```

5. X の再起動後、fglrxinfo コマンドで OpenGL 設定が正しく行われたことを確認します。

```
# fglrxinfo
```

```
display: :0.0 screen: 0
OpenGL vendor string: ATI Technologies Inc.
OpenGL renderer string: MOBILITY RADEON 9600 Generic
OpenGL version string: 1.3.5461 (X4.3.0-8.19.10)
```

LG3D のインストール (Solaris 10 x86 編)

準備

Solaris 10 x86 で LG3D を動かすためには以下のバイナリが必要です。 まずはこれらをダウンロードしてください。

以下のファイルは執筆時点での最新版です。できるだけ最新版を利用して下さい。

コンポーネント名	ファイル名	ダウンロード元
JDK 5.0	jdk-1_5_0_05-solaris-i586.sh	http://java.sun.com/j2se/1.5.0/ja/download.html
JAI 1.1.2	jai-1_1_2_01-lib-solaris-i586-jdk.bin	http://java.sun.com/products/java-media/jai/downloads/download-1_1_2_01.html
Java 3D 1.4.0 beta2	java3d-1_4_0-beta2-solaris-x86.zip	https://java3d.dev.java.net/binary-builds.html#Stable_builds_1.4.0
LG3D Release 0.7.1	lg3d-fcs-rel-0-7-1-solaris-i86pc-0510281634.tar.gz	https://lg3d-core.dev.java.net/binary-builds.html
LG3D Release 0.7.1 Javadoc	javadoc-lg3d-fcs-rel-0-7-1-generic-0510281819.tar.gz	https://lg3d-core.dev.java.net/binary-builds.html

Java 環境のインストール

「準備」の項目でダウンロードしたファイルは /tmp にあるものと仮定します。

ここではユーザーのホームディレクトリに lg3d というディレクトリを作成し、インストールを行います。

以下、ユーザによる入力を赤(太字)で示します。

- ホームディレクトリに lg3d というディレクトリを作成します。

```
# cd  
# mkdir lg3d
```

- JDK 1.5 をインストールします。

```
# cd lg3d  
# sh /tmp/jdk-1_5_0_05-solaris-i586.sh
```

Sun Microsystems, Inc. Binary Code License Agreement

for the JAVA 2 PLATFORM STANDARD EDITION DEVELOPMENT KIT 5.0

SUN MICROSYSTEMS, INC. ("SUN") IS WILLING TO LICENSE THE SOFTWARE IDENTIFIED BELOW TO YOU ONLY UPON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS BINARY CODE LICENSE AGREEMENT AND SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT"). PLEASE READ THE AGREEMENT CAREFULLY. BY DOWNLOADING OR INSTALLING THIS SOFTWARE, YOU ACCEPT THE TERMS OF THE AGREEMENT. INDICATE ACCEPTANCE BY SELECTING THE "ACCEPT" BUTTON AT THE BOTTOM OF THE AGREEMENT. IF YOU ARE NOT WILLING TO BE BOUND BY ALL THE TERMS, SELECT THE "DECLINE" BUTTON AT THE BOTTOM OF THE AGREEMENT AND THE DOWNLOAD OR INSTALL PROCESS WILL NOT CONTINUE.

....

For inquiries please contact: Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. (LFI#141623/Form

ID#011801)

Do you agree to the above license terms? [yes or no]

yes

Unpacking...

...

Creating jdk1.5.0_05/jre/lib/plugin.jar

Creating jdk1.5.0_05/jre/lib/javaws.jar

Creating jdk1.5.0_05/jre/lib/deploy.jar

Done.

- JAI をインストールします。

```
# cd jdk1.5.0_05  
# sh /tmp/jai-1_1_2_01-lib-solaris-i586-jdk.bin
```

Sun Microsystems, Inc.

Binary Code License Agreement

JAVA ADVANCED IMAGING API, VERSION 1.1.2

READ THE TERMS OF THIS AGREEMENT AND ANY PROVIDED SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT") CAREFULLY BEFORE OPENING THE SOFTWARE MEDIA PACKAGE.? BY OPENING THE SOFTWARE MEDIA PACKAGE, YOU AGREE TO THE TERMS OF THIS AGREEMENT.? IF YOU ARE ACCESSING THE SOFTWARE ELECTRONICALLY,

INDICATE YOUR ACCEPTANCE OF THESE TERMS BY SELECTING THE "ACCEPT" BUTTON AT

THE END OF THIS AGREEMENT.? IF YOU DO NOT AGREE TO ALL THESE TERMS, PROMPTLY RETURN THE UNUSED SOFTWARE TO YOUR PLACE OF PURCHASE FOR A REFUND OR,

IF THE SOFTWARE IS ACCESSED ELECTRONICALLY, SELECT THE "DECLINE" BUTTON AT THE END OF THIS AGREEMENT.

...

For inquiries please contact: Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A
(LFI#143342/Form ID#011801)

Do you agree to the above license terms? [yes or no]

yes

Unpacking...

Checksumming...

0

0

UnZipSFX 5.31 of 31 May 1997, by Info-ZIP (Zip-Bugs@lists.wku.edu).

inflating: COPYRIGHT-jai.txt

inflating: INSTALL-jai.txt

inflating: LICENSE-jai.txt

inflating: README-jai.txt

inflating: UNINSTALL-jai

inflating: jre/lib/i386/libmlib_jai.so

inflating: jre/lib/ext/jai_core.jar

inflating: jre/lib/ext/jai_codec.jar

inflating: jre/lib/ext/mlibwrapper_jai.jar

Done.

- Java 3D をインストールします。

```
# cd /tmp
# unzip java3d-1_4_0-beta2-solaris-x86.zip
Archive: java3d-1_4_0-beta2-solaris-x86.zip
  creating: java3d-1_4_0-beta2-solaris-x86/
  inflating: java3d-1_4_0-beta2-solaris-x86/HOW-TO-INSTALL.txt
  inflating: java3d-1_4_0-beta2-solaris-x86/README.txt
  inflating: java3d-1_4_0-beta2-solaris-x86/COPYRIGHT.txt
  inflating: java3d-1_4_0-beta2-solaris-x86/LICENSE-JRL.txt
  inflating: java3d-1_4_0-beta2-solaris-x86/LICENSE.txt
extracting: java3d-1_4_0-beta2-solaris-x86/j3d-140-beta2-sol-x86.zip
# cd
# cd lg3d/jdk1.5.0_05/jre
# unzip /tmp/java3d-1_4_0-beta2-solaris-x86/j3d-140-beta2-sol-x86.zip
Archive: ../../java3d-1_4_0-beta2-solaris-x86/j3d-140-beta2-sol-x86.zip
  inflating: lib/ext/vecmath.jar
  inflating: lib/ext/j3dcore.jar
  inflating: lib/ext/j3dutils.jar
  inflating: lib/i386/libj3dcore-ogl.so
  inflating: lib/i386/libj3dutils.so
  creating: lib/amd64/
  inflating: lib/amd64/libj3dcore-ogl.so
  inflating: lib/amd64/libj3dutils.so
```

LG3D のインストール

LG3D のインストールはダウンロードしたファイルを解凍します。

```
# cd
# gunzip -c /tmp/lg3d-fcs-rel-0-7-1-solaris-i86pc-0510281634.tar.gz | tar xf -
```

LG3D Javadoc のインストール

LG3D 0.7.1 から Javadoc は本体からはずされました。

LG3D を利用するだけであれば Javadoc は必要ありませんが、 LG3D プログラミングを行う場合には Javadoc は有用なのでインストールします。

インストール方法は LG3D と同様にファイルを解凍のみです。 解凍時に警告が出ますが無視してかまいません。解凍後 @LongLink という不要なファイルを削除します。

```
# cd
# gunzip -c /tmp/javadoc-lg3d-fcs-rel-0-7-1-generic-0510281819.tar.gz | tar xf -
tar: ./@LongLink: タイプフラグ 'L' を認識できません。通常のファイルに変換して います
...
# rm @LongLink
```

LG3D のインストール (Linux 編)

準備

Linux で LG3D を動かすためには以下のバイナリが必要です。 まずはこれらをダウンロードしてください。

以下のファイルは執筆時点での最新版です。 できるだけ最新版を利用してください。

コンポーネント名	ファイル名	ダウンロード元
JDK 5.0	jdk-1_5_0_05-linux-i586.bin	http://java.sun.com/j2se/1.5.0/ja/download.html
JAI 1.1.2	jai-1_1_2_01-lib-linux-i586-jdk.bin	http://java.sun.com/products/java-media/jai/downloads/download-1_1_2_01.html
Java 3D 1.4.0 beta2	java3d-1_4_0-beta2-linux-i586.zip	https://java3d.dev.java.net/binary-builds.html#Stable_builds_1.4.0
LG3D Release 0.7.1	lg3d-fcs-rel-0-7-1-linux-i686-0510281810.tar.gz	https://lg3d-core.dev.java.net/binary-builds.html
LG3D Release 0.7.1 Javadoc	javadoc-lg3d-fcs-rel-0-7-1-generic-0510281819.tar.gz	https://lg3d-core.dev.java.net/binary-builds.html

Java 環境のインストール

「準備」の項目でダウンロードしたファイルは /tmp にあるものと仮定します。

ここではユーザーのホームディレクトリに lg3d というディレクトリを作成し、インストールを行います。

以下、ユーザによる入力を赤(太字)で示します。

- ホームディレクトリに lg3d というディレクトリを作成します。

```
# cd  
# mkdir lg3d
```

- JDK 1.5 をインストールします。

```
# cd lg3d  
# sh /tmp/jdk-1_5_0_05-linux-i586.bin
```

Sun Microsystems, Inc. Binary Code License Agreement

for the JAVA 2 PLATFORM STANDARD EDITION DEVELOPMENT KIT 5.0

SUN MICROSYSTEMS, INC. ("SUN") IS WILLING TO LICENSE THE SOFTWARE IDENTIFIED BELOW TO YOU ONLY UPON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS BINARY CODE LICENSE AGREEMENT AND SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT"). PLEASE READ THE AGREEMENT CAREFULLY. BY DOWNLOADING OR INSTALLING THIS SOFTWARE, YOU ACCEPT THE TERMS OF THE AGREEMENT. INDICATE ACCEPTANCE BY SELECTING THE "ACCEPT" BUTTON AT THE BOTTOM OF THE AGREEMENT. IF YOU ARE NOT WILLING TO BE BOUND BY ALL THE TERMS, SELECT THE "DECLINE" BUTTON AT THE BOTTOM OF THE AGREEMENT AND THE DOWNLOAD OR INSTALL PROCESS WILL NOT CONTINUE.

....

For inquiries please contact: Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. (LFI#141623/Form ID#011801)

Do you agree to the above license terms? [yes or no]

yes

Unpacking...

....

Creating jdk1.5.0_05/jre/lib/plugin.jar

Creating jdk1.5.0_05/jre/lib/javaws.jar

Creating jdk1.5.0_05/jre/lib/deploy.jar

Done.

- JAI をインストールします。

```
# cd jdk1.5.0_05  
# sh /tmp/jai-1_1_2_01-lib-linux-i586-jdk.bin
```

Sun Microsystems, Inc.

Binary Code License Agreement

JAVA ADVANCED IMAGING API, VERSION 1.1.2

READ THE TERMS OF THIS AGREEMENT AND ANY PROVIDED SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT") CAREFULLY BEFORE OPENING THE SOFTWARE MEDIA PACKAGE.? BY OPENING THE SOFTWARE MEDIA PACKAGE, YOU AGREE TO THE TERMS OF THIS AGREEMENT.? IF YOU ARE ACCESSING THE SOFTWARE ELECTRONICALLY,

INDICATE YOUR ACCEPTANCE OF THESE TERMS BY SELECTING THE "ACCEPT" BUTTON AT

THE END OF THIS AGREEMENT.? IF YOU DO NOT AGREE TO ALL THESE TERMS, PROMPTLY RETURN THE UNUSED SOFTWARE TO YOUR PLACE OF PURCHASE FOR A REFUND OR,

IF THE SOFTWARE IS ACCESSED ELECTRONICALLY, SELECT THE "DECLINE" BUTTON AT THE END OF THIS AGREEMENT.

....

For inquiries please contact: Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A
(LFI#143342/Form ID#011801)

Do you agree to the above license terms? [yes or no]

yes

Unpacking...

Checksumming...

0

0

UnZipSFX 5.31 of 31 May 1997, by Info-ZIP (Zip-Bugs@lists.wku.edu).

inflating: COPYRIGHT-jai.txt

inflating: INSTALL-jai.txt

inflating: LICENSE-jai.txt

inflating: README-jai.txt

inflating: UNINSTALL-jai

inflating: jre/lib/i386/libmllib_jai.so

inflating: jre/lib/ext/jai_core.jar

inflating: jre/lib/ext/jai_codec.jar

inflating: jre/lib/ext/mlibwrapper_jai.jar

Done.

- Java 3D をインストールします。

```
# cd /tmp
# unzip java3d-1_4_0-beta2-linux-i586.zip
Archive: java3d-1_4_0-beta2-linux-i586.zip
  creating: java3d-1_4_0-beta2-linux-i586/
  inflating: java3d-1_4_0-beta2-linux-i586/LICENSE-JRL.txt
  inflating: java3d-1_4_0-beta2-linux-i586/HOW-TO-INSTALL.txt
  inflating: java3d-1_4_0-beta2-linux-i586/COPYRIGHT.txt
  extracting: java3d-1_4_0-beta2-linux-i586/j3d-140-beta1-linux-x86.zip
  inflating: java3d-1_4_0-beta2-linux-i586/LICENSE.txt
  inflating: java3d-1_4_0-beta2-linux-i586/README.txt
# cd
# cd lg3d/jdk1.5.0_05/jre
# unzip /tmp/java3d-1_4_0-beta2-linux-i586/j3d-140-beta2-linux-x86.zip
Archive: /tmp/java3d-1_4_0-beta2-linux-i586/j3d-140-beta2-linux-x86.zip
  inflating: lib/ext/j3dcore.jar
  inflating: lib/ext/j3dutils.jar
  inflating: lib/ext/vecmath.jar
  inflating: lib/i386/libj3dcore-ogl.so
  inflating: lib/i386/libj3dutils.so
  inflating: lib/i386/libj3dcore-ogl-cg.so
```

LG3D のインストール

LG3D のインストールはダウンロードしたファイルを解凍するだけです。

```
# cd
# tar xzf /tmp/lg3d-fcs-rel-0-7-1-linux-i686-0510281810.tar.gz
```

Javadoc のインストール

LG3D 0.7.1 から Javadoc は本体からはずされました。

LG3D を利用するだけであれば Javadoc は必要ありませんが、 LG3D プログラミングを行う場合には Javadoc は有用なのでインストールします。

インストール方法は LG3D と同様にファイルを解凍のみです。

```
# cd
# tar xzf /tmp/javadoc-lg3d-fcs-rel-0-7-1-generic-0510281819.tar.gz
```

LG3D の実行(Solaris/Linux 編)

LG3D の 3 つの実行コマンド

LG3D をインストールすると lg3d/bin 内に lg3d-dev, lg3d-app, lg3d-session という 3 つのコマンドができます。これらが LG3D を実行するためのコマンドです。

その他のファイルは無視して構いません。

lg3d-dev, lg3d-app はウィンドウアプリケーションとして LG3D を実行するためのコマンドです。
開発者用のコマンドと位置づけられています。

lg3d-dev と lg3d-app の違いは X-Window アプリケーションが実行できるかどうかです。 lg3d-dev は LG3D アプリケーションのみ実行でき、 lg3d-app は X-Window アプリケーションの実行も行えます。

一方、 lg3d-session は X-Window のセッションごと起動するモードです。

各環境により利用できるコマンドは異なります。各環境で利用できるコマンドは次の通りです。

	lg3d-dev	lg3d-app	lg3d-session
Solaris 10 x86 + NVIDIA	○	○	○
Linux + NVIDIA	○	○	○
Linux + ATI	○	○	×
Windows	○	×	×

開発者向けモード(lg3d-dev/lg3d-app)の実行

lg3d-dev, lg3d-app の実行方法は次の通りです。

コマンドの実行前に JAVA_HOME 環境変数を設定します。

(インストールしたユーザーのホームディレクトリを /home/duke とします。)

sh, bash の場合

```
# JAVA_HOME=/home/duke/lg3d/jdk1.5.0_05 ; export JAVA_HOME  
# cd lg3d  
# bin/lg3d-dev (または bin/lg3d-app)
```

csh, tcsh の場合

```
# setenv JAVA_HOME /home/duke/lg3d/jdk1.5.0_05  
# cd lg3d  
# bin/lg3d-dev (または bin/lg3d-app)
```

補足 1:

Fedora Core 4 の標準状態では lg3d-app コマンドは動きません。

/etc/X11/gdm/gdm.conf に DisallowTCP=false を追加し、X-Window の再起動が必要です。

下図は Solaris 10 x86 3/05 上で lg3d-dev を実行したものです。



補足 2:

インストールディレクトリの etc/lg3d/displayconfig/j3d1x1 (ここでは /home/duke/lg3d/etc/lg3d/displayconfig/j3d1x1) ファイルの設定を変更することにより LG3D ウィンドウの大きさを変更できます。

下の例では画面サイズを 1152x864 に設定しています。 画面サイズの代わりに NoBorderFullScreen と指定すればフルスクリーン表示も可能です。

変更箇所を **赤(太字)** で示します。

```
/*
*****
*
* * Java 3D Calibration file for single-screen desktop configuration with
* neither head tracking nor 6DOF sensor tracking.
*
*****
*/
// Create a new screen object and associate it with a logical name and a
// number. This number is used as an index to retrieve the AWT GraphicsDevice
// from the array that GraphicsEnvironment.getScreenDevices() returns.
//
```

```
(NewScreen center 0)
```

```
// Set the available image area for a full screen. This is important when
// precise scaling between objects in the virtual world and their projections
// into the physical world is desired through use of an explicit ScreenScale
// view attribute. The defaults are 0.365 meters for width and 0.292 meters
// for height.
//
(ScreenAttribute center PhysicalScreenWidth 0.360)
(ScreenAttribute center PhysicalScreenHeight 0.288)
//(ScreenAttribute center WindowSize NoBorderFullScreen)
//(ScreenAttribute center WindowSize (800 600))
(ScreenAttribute center WindowSize (1152 864))

// Create a view using the defined screen.
//
(NewView view0)
(ViewAttribute view0 Screen center)
(ViewAttribute view0 FrontClipDistance 0.01)
(ViewAttribute view0 BackClipDistance 10.0)
(ViewAttribute view0 WindowEyepointPolicy RELATIVE_TO_COEXISTENCE)
(ViewAttribute view0 WindowMovementPolicy VIRTUAL_WORLD)
(ViewAttribute view0 WindowResizePolicy VIRTUAL_WORLD)
(ViewAttribute view0 ScreenScalePolicy SCALE_EXPLICIT)

// For debugging this will give us the standard scale world and view
// but in a window. Obviously if the window is reduced in size less of
// the world is visible.
(ViewAttribute view0 CoexistenceCenteringEnable true)
(ViewAttribute view0 WindowEyepointPolicy RELATIVE_TO_WINDOW)
(ViewAttribute view0 WindowMovementPolicy PHYSICAL_WORLD)

// Enable stereo viewing if desired
// (ViewAttribute view0 StereoEnable true)
```

セッションモード(lg3d-session コマンド)の実行

lg3d-session を実行するためにはあらかじめ X-Window を終了しておきます。

```
Solaris 10 x86 の場合
```

```
# /etc/init.d/dtlogin stop
```

```
Linux の場合
```

```
# init 3
```

X-Window を終了後は lg3d-dev/lg3d-app と同様の方法で lg3d-session を実行します。

```
sh, bash の場合
```

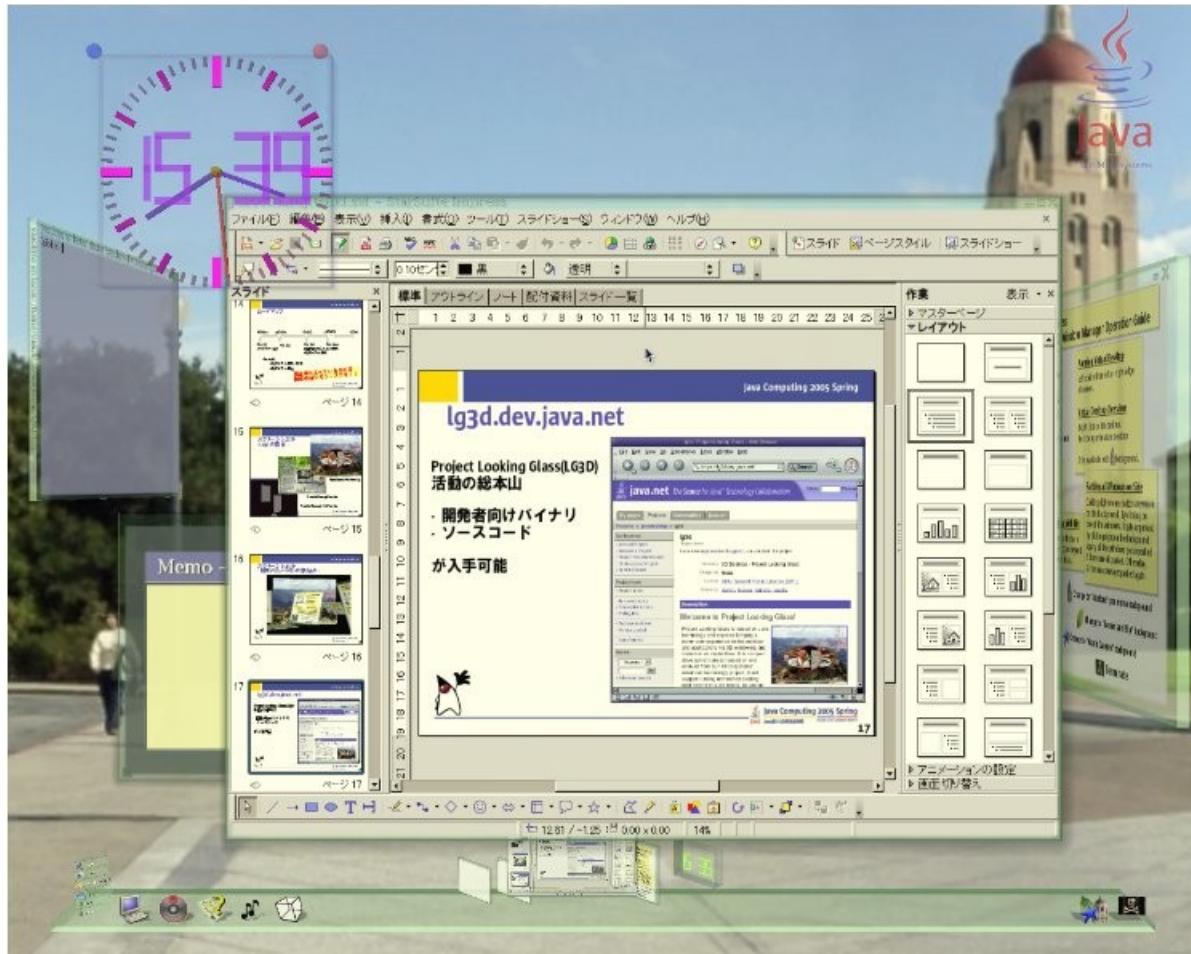
```
# JAVA_HOME=/home/duke/lg3d/jdk1.5.0_05 ; export JAVA_HOME
# cd lg3d
# bin/lg3d-session
```

csh, tcsh の場合

```
# setenv JAVA_HOME /home/duke/lg3d/jdk1.5.0_05  
# cd lg3d  
# bin/lg3d-session
```

下図は lg3d-session を実行したものです。

X-Window アプリケーションの例として StarSuite 8 を起動しています。



LG3D のインストールと実行方法 (Windows XP 編)

準備

Windows XP で LG3D を動かすためには以下のバイナリが必要です。 まずはこれらをダウンロードしてください。

以下のファイルは執筆時点での最新版です。 できるだけ最新版を利用してください。

コンポーネント名	ファイル名	ダウンロード元
JDK 5.0	jdk-1_5_0_05-windows-i586-p.exe	http://java.sun.com/j2se/1.5.0/ja/download.html
JAI 1.1.2	jai-1_1_2_01-lib-windows-i586-jdk.exe	http://java.sun.com/products/java-media/jai/downloads/download-1_1_2_01.html
Java 3D 1.4.0 beta2	java3d-1_4_0-beta2-windows-i586.zip	https://java3d.dev.java.net/binary-builds.html#Stable_builds_1.4.0
LG3D Release 0.7.1	lg3d-fcs-rel-0-7-1-generic-0510281811.zip	https://lg3d-core.dev.java.net/binary-builds.html

補足:

LG3D 0.7.1 から Javadoc が独立したパッケージになりました。 ([javadoc-lg3d-fcs-rel-0-7-1-generic-0510281819.tar.gz](#))

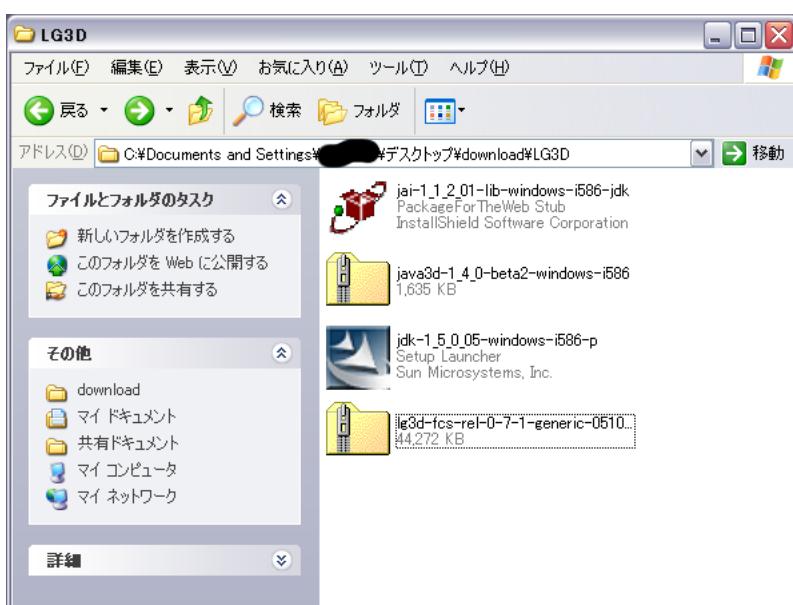
ただ、現状の配布形式は tar.gz 形式のため、Windows 環境で解凍するためには 別途 tar.gz に対応した解凍ツールが必要になります。

そのため、このテキストでは Javadoc のインストール部分は解説していません。

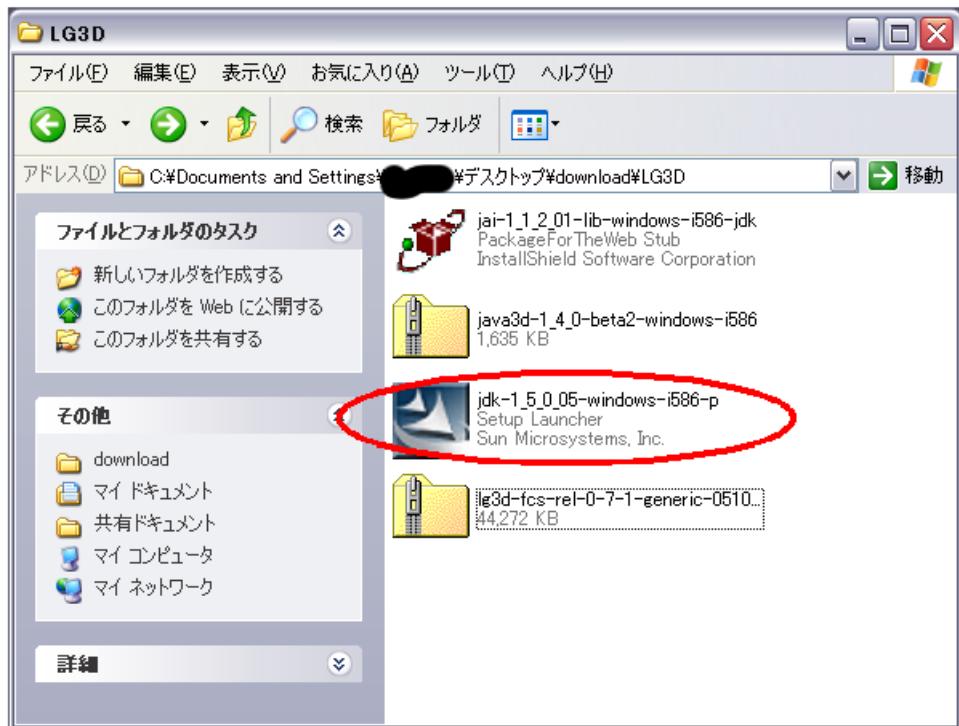
JDK のインストール

インストールは "Administrator" 権限を持つユーザで行う必要があります。

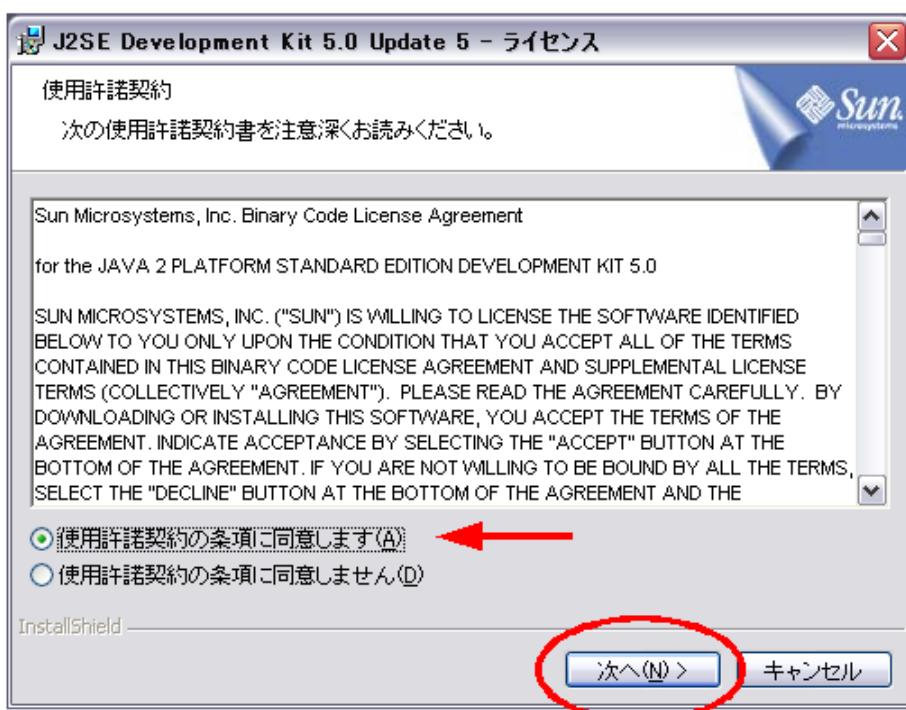
ここでは、デスクトップ上に **download\lg3d** というディレクトリを作成し、必要なファイルをダウンロードしています。



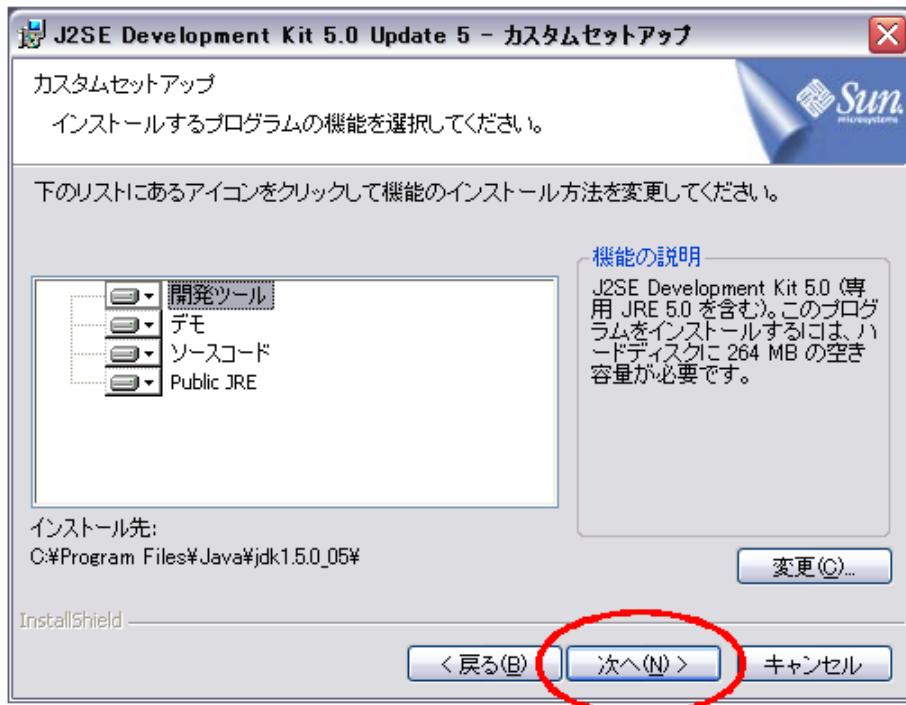
- jdk-1_5_0_05-windows-i586-p.exe をダブルクリックし、インストーラを起動します。



- しばらく待つとインストーラが起動します。
ライセンスが表示されるので確認してください。
インストールを継続する場合は「**使用許諾契約の条項に同意します**」を選択し、「次へ」をクリックします。



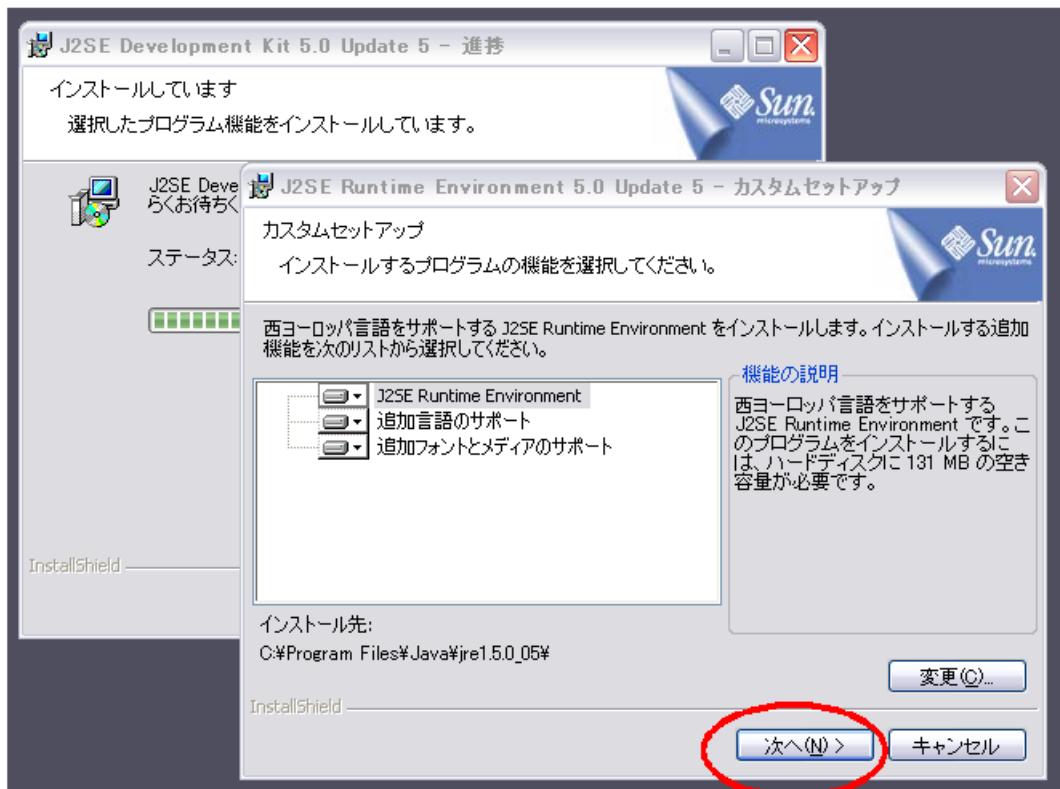
- カスタムセットアップの項目に移ります。ここではデフォルト設定のままインストールします。インストール先は「C:\Program Files\Java\jdk1.5.0_05」になります。続けるには「次へ」をクリックします。



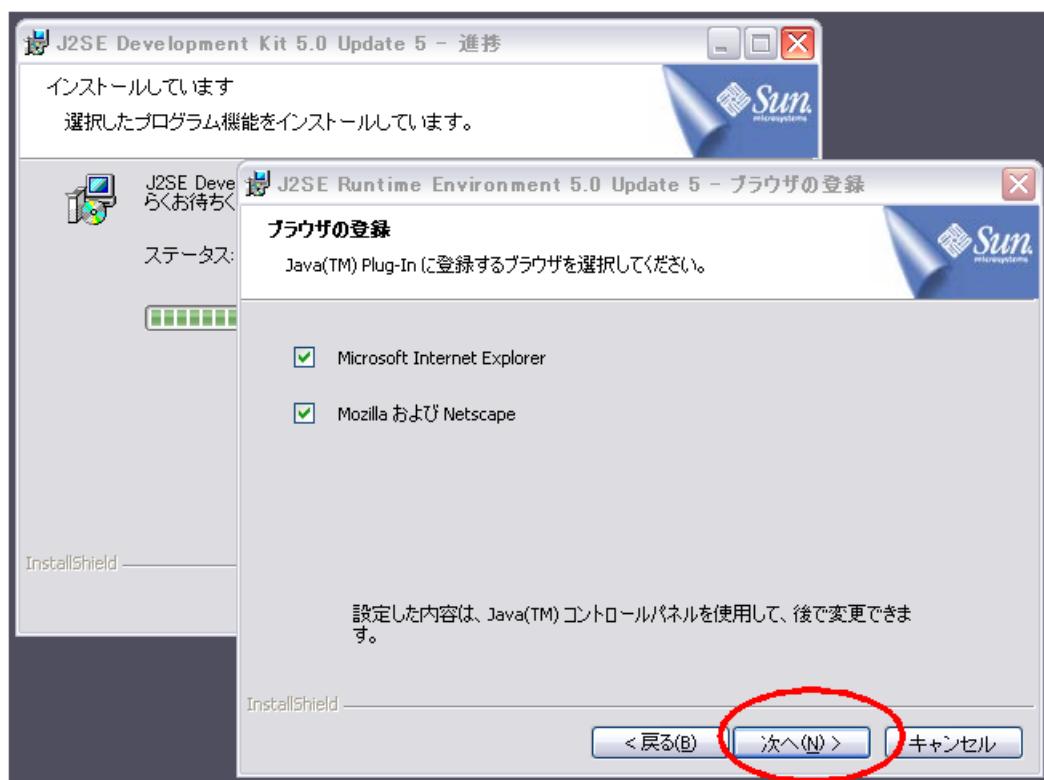
- JDKがインストールされます。



- JDK のインストール途中で、JRE のインストールダイアログが表示されます。JRE のカスタムセットアップの項目になります。ここではデフォルト設定のままインストールします。「次へ」をクリックします。



- Java Plug-In のブラウザへの登録選択をします。Java Plug-In をブラウザに登録したくない場合は選択を解除します。ここではデフォルト設定とします。インストールを継続するには「次へ」をクリックします。



- JRE のインストールが行われます。
JRE のダイアログはインストールが終了すると自動的に終了します。

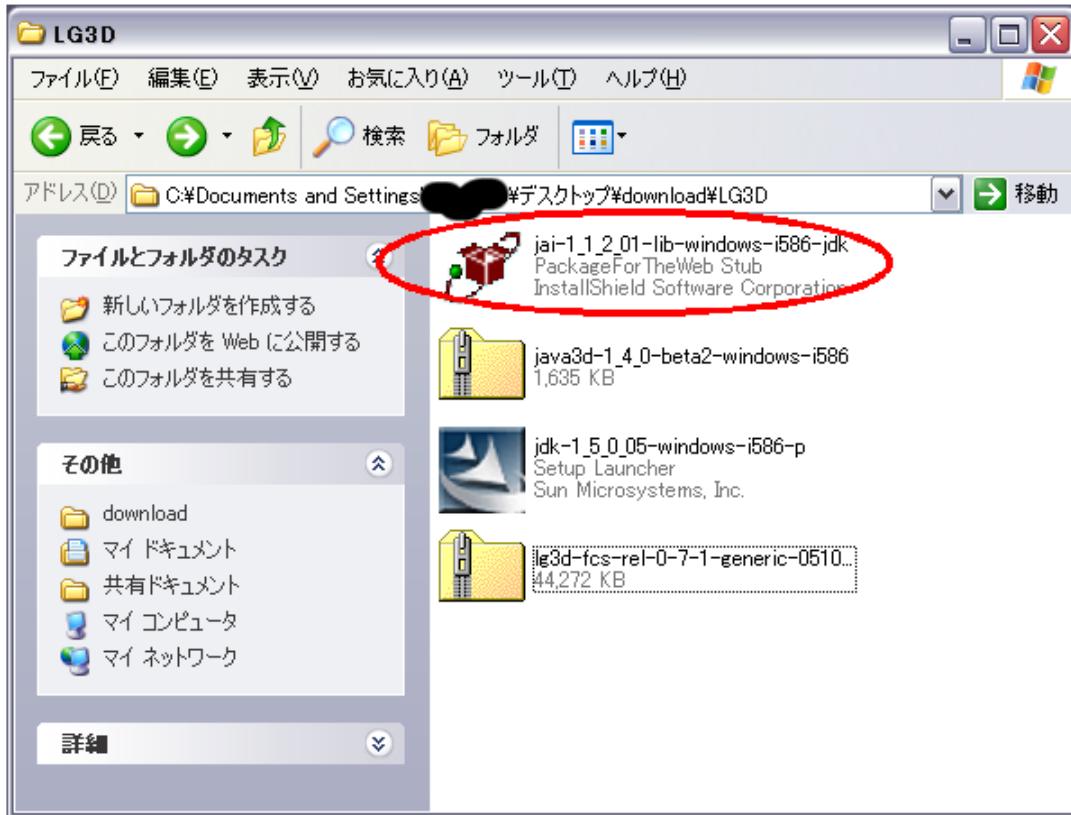


- JDK のインストールの終了が表示されるので、「完了」をクリックします。

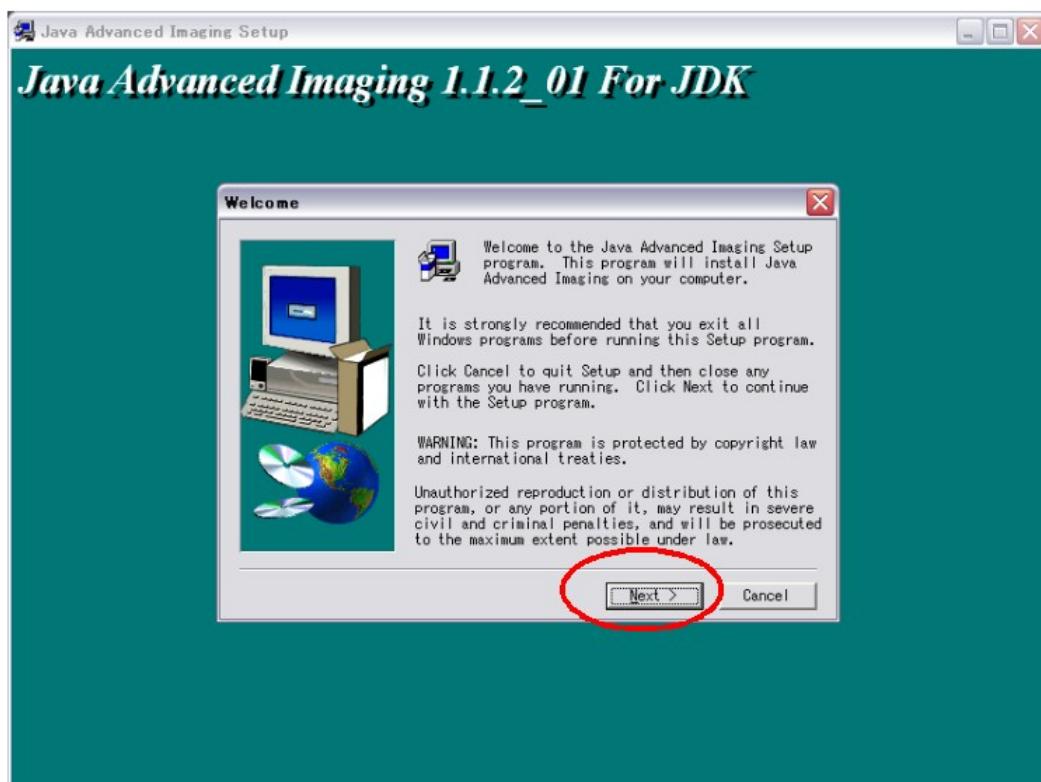


JAI のインストール

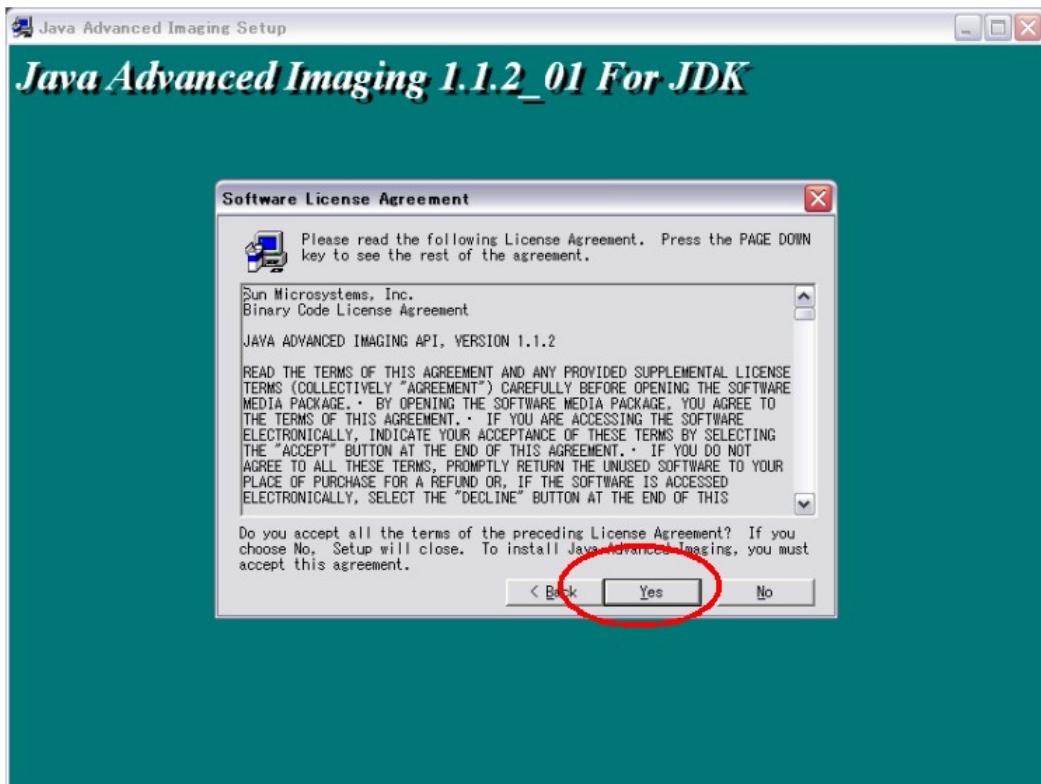
- jai-1_1_2_01-lib-windows-i586-jdk.exe をダブルクリックします。



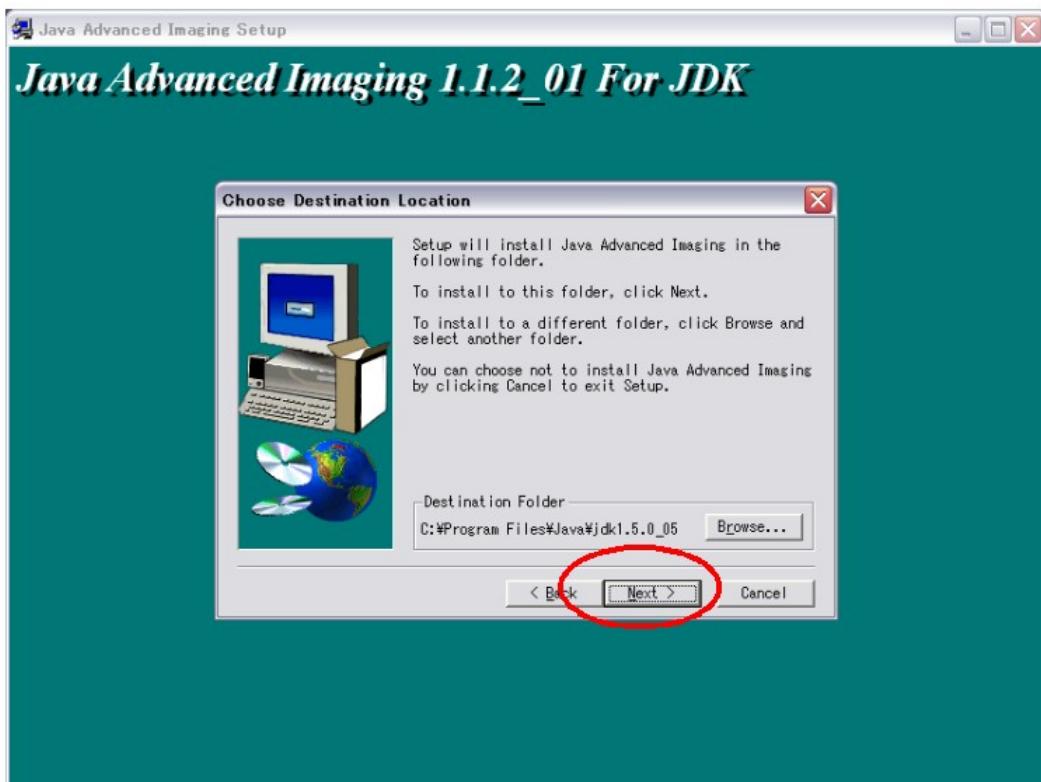
- インストーラが起動します。
「Next」をクリックし、次に進みます。



- ライセンス条項が表示されますので確認してください
「Yes」をクリックするとインストールを継続します。

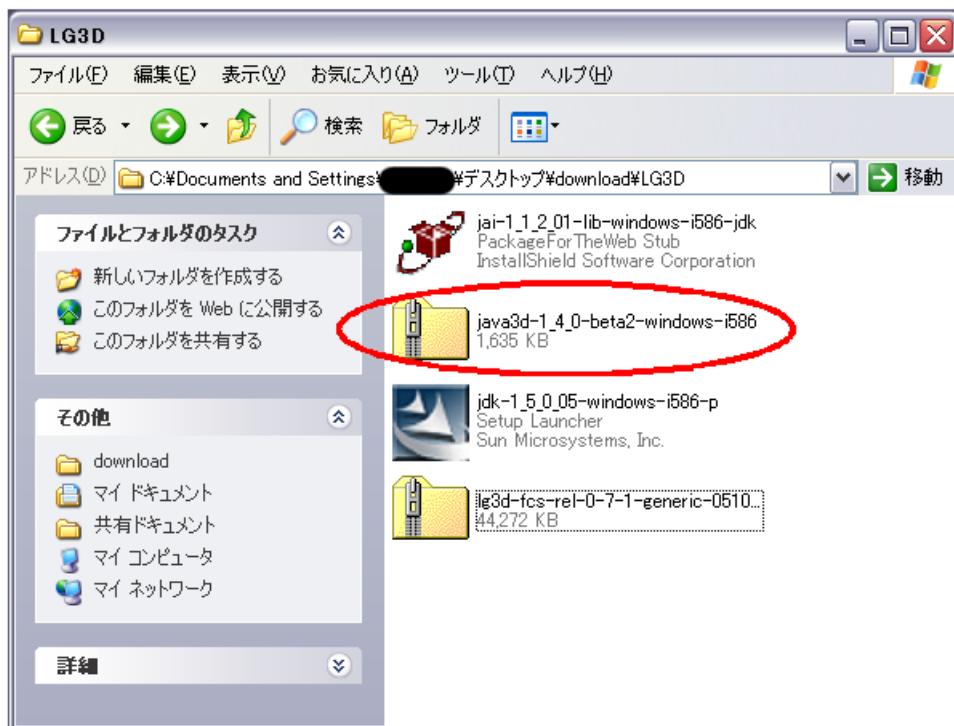


- インストール先の選択画面になるので、JDK のディレクトリが正しいか確認します。
「Next」をクリックするとインストールを開始します。
インストールが終了するとインストーラは自動的に終了します。

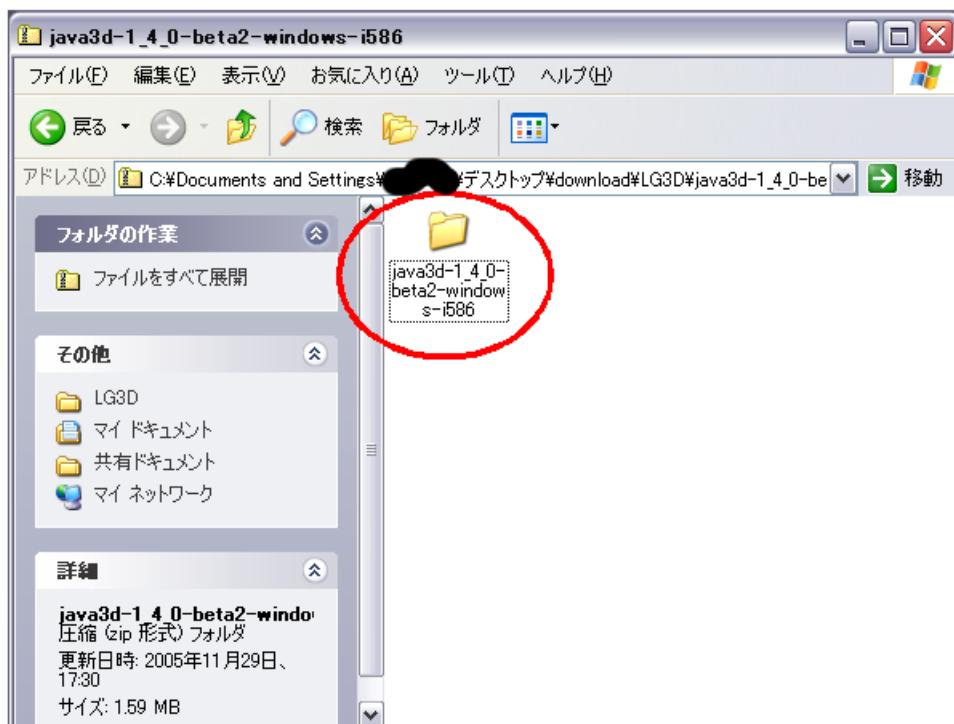


Java3D のインストール

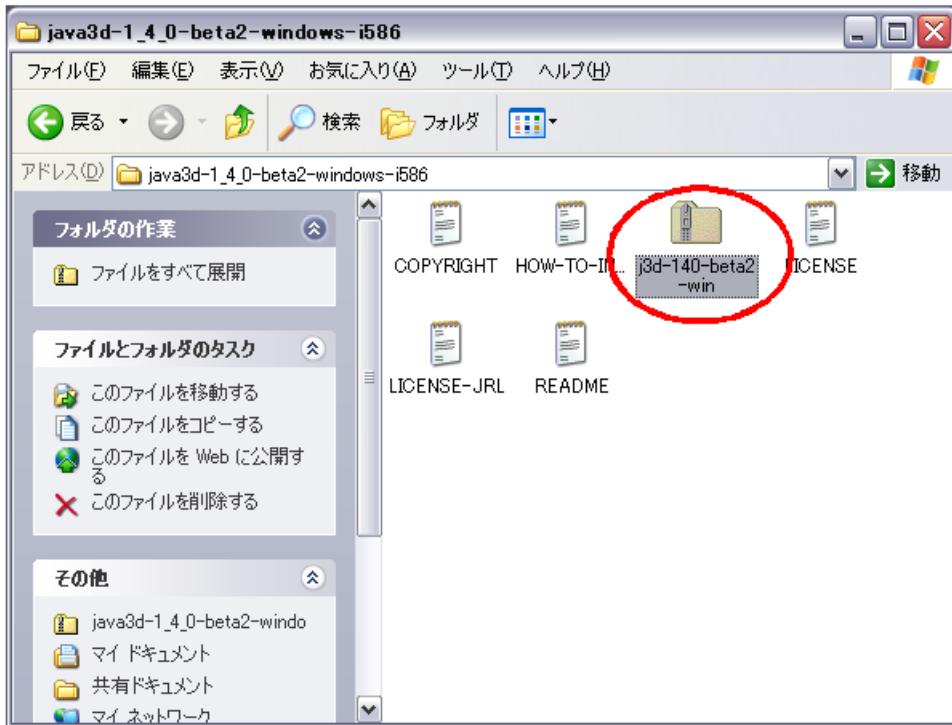
- java3d-1_4_0-beta2-windows-i586.zip をダブルクリックします。



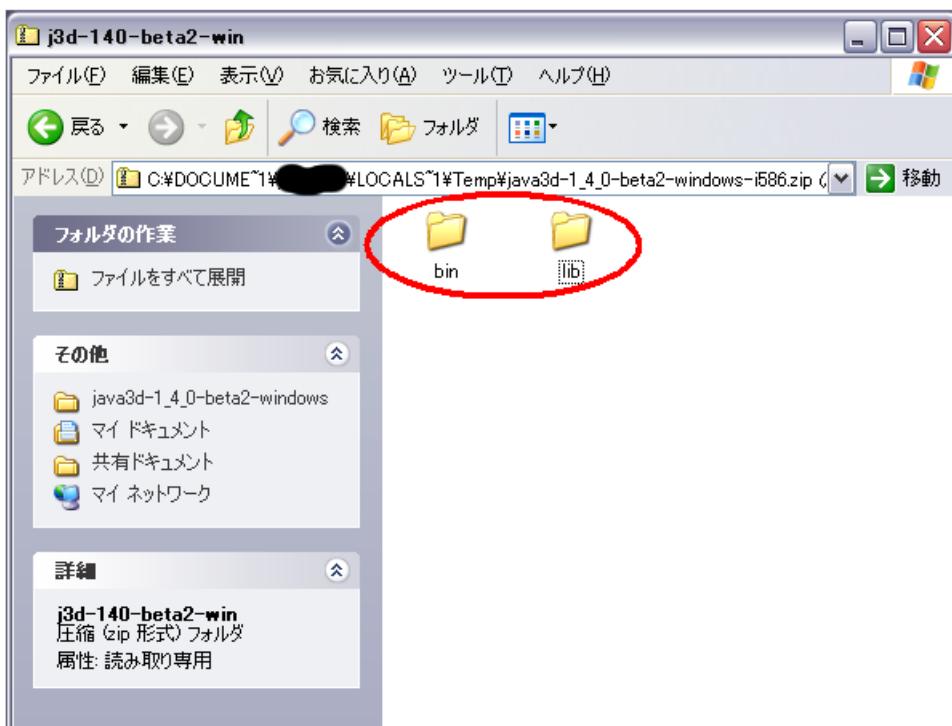
- zip ファイル内のフォルダが表示されます。フォルダをダブルクリックします。



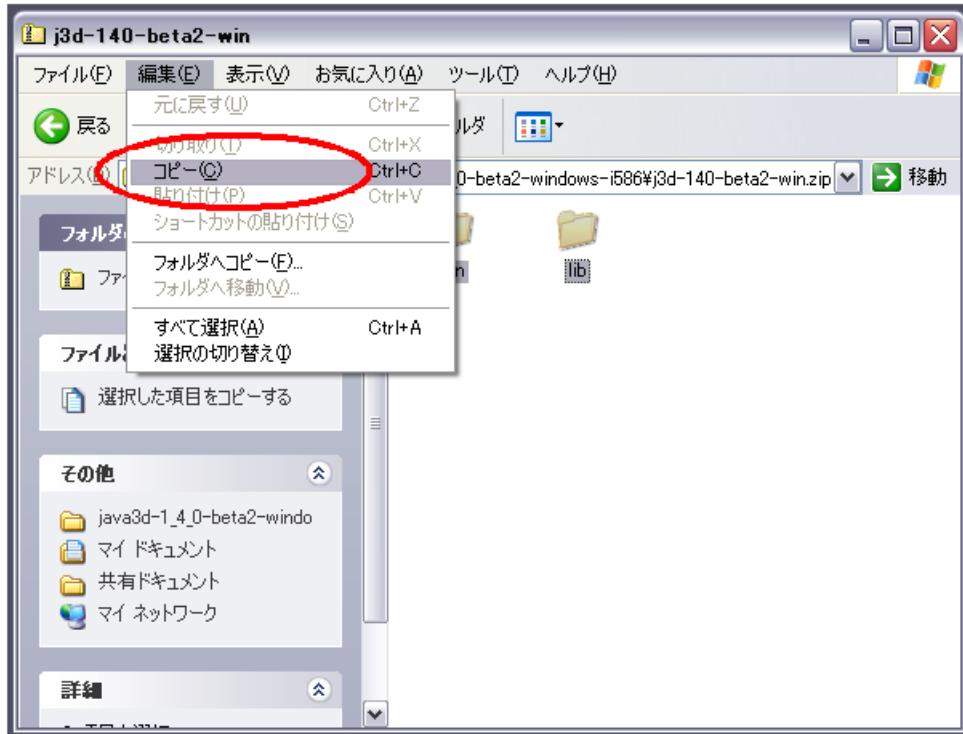
- **j3d-140-beta2-windows-i586.zip** 圧縮フォルダをダブルクリックします。



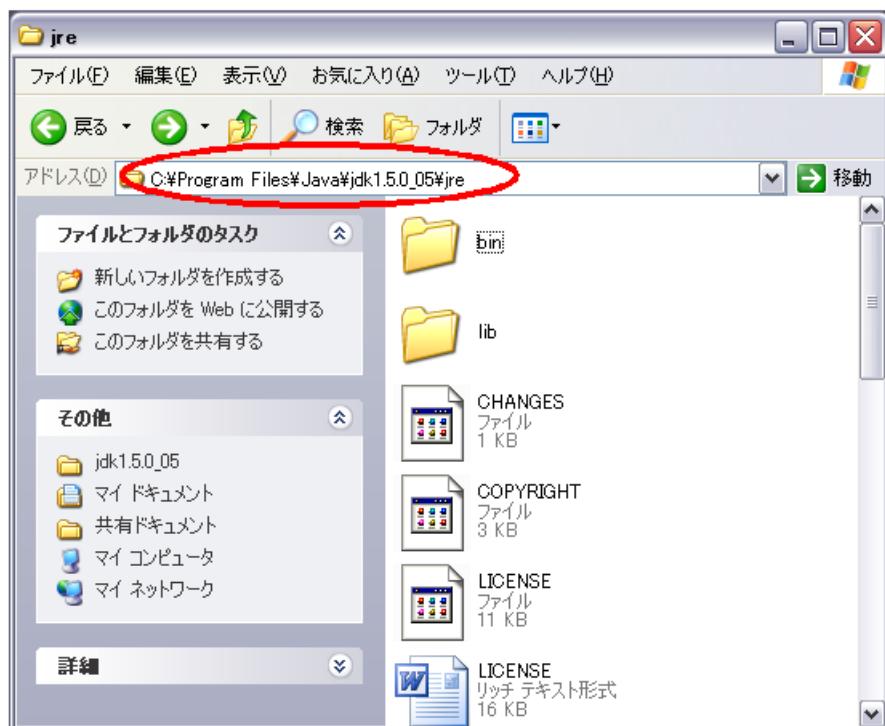
- **bin** と **lib** が表示されるので、両方選択します。



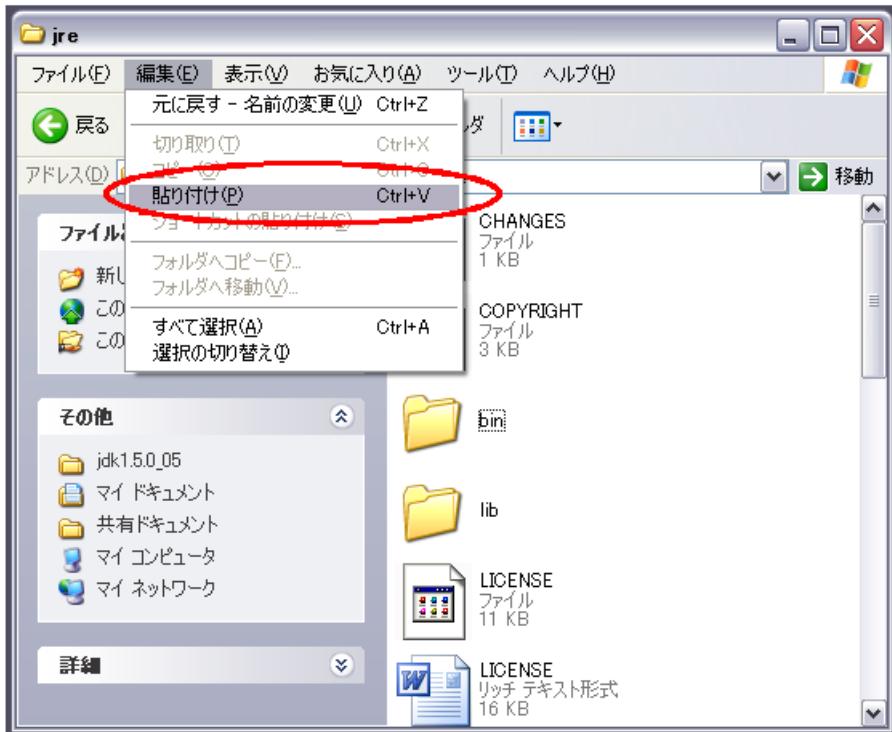
- 「編集」→「コピー」を選択します。



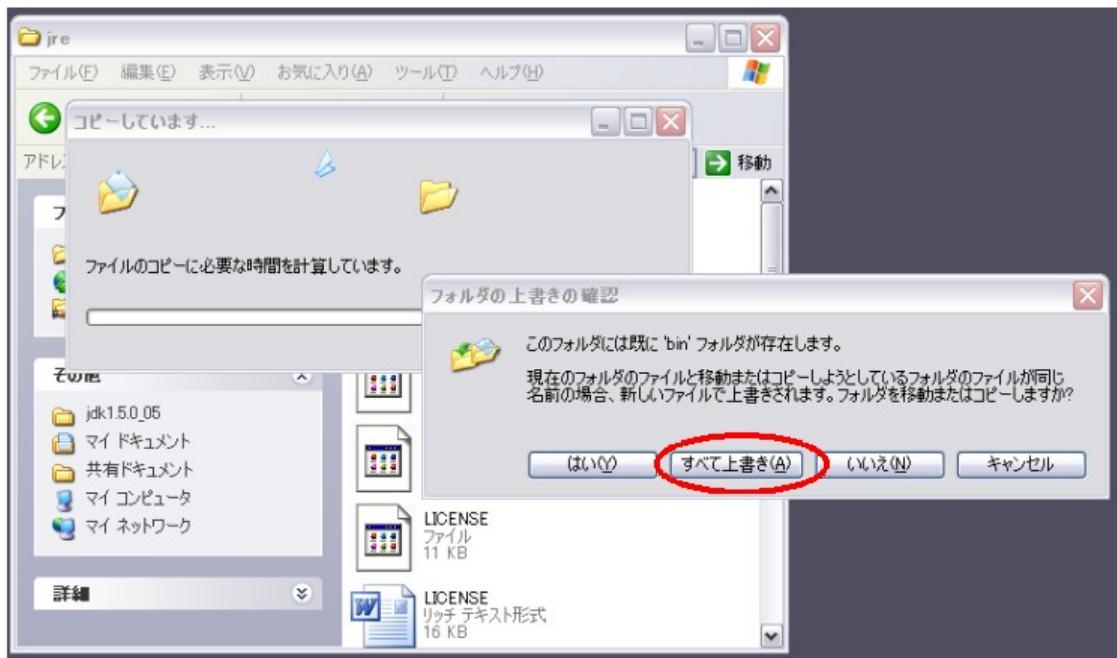
- JDKをインストールしたディレクトリにある jre ディレクトリ (**C:\Program Files\Java\jdk1.5.0_05\jre**) をエクスプローラで開きます。



- 「編集」→「貼り付け」を選択します。



- フォルダの上書き確認が出ますので、「すべて上書き」を選択します。



環境変数の設定

LG3D を実行するために必要となる環境変数(JAVA_HOME)を設定します。

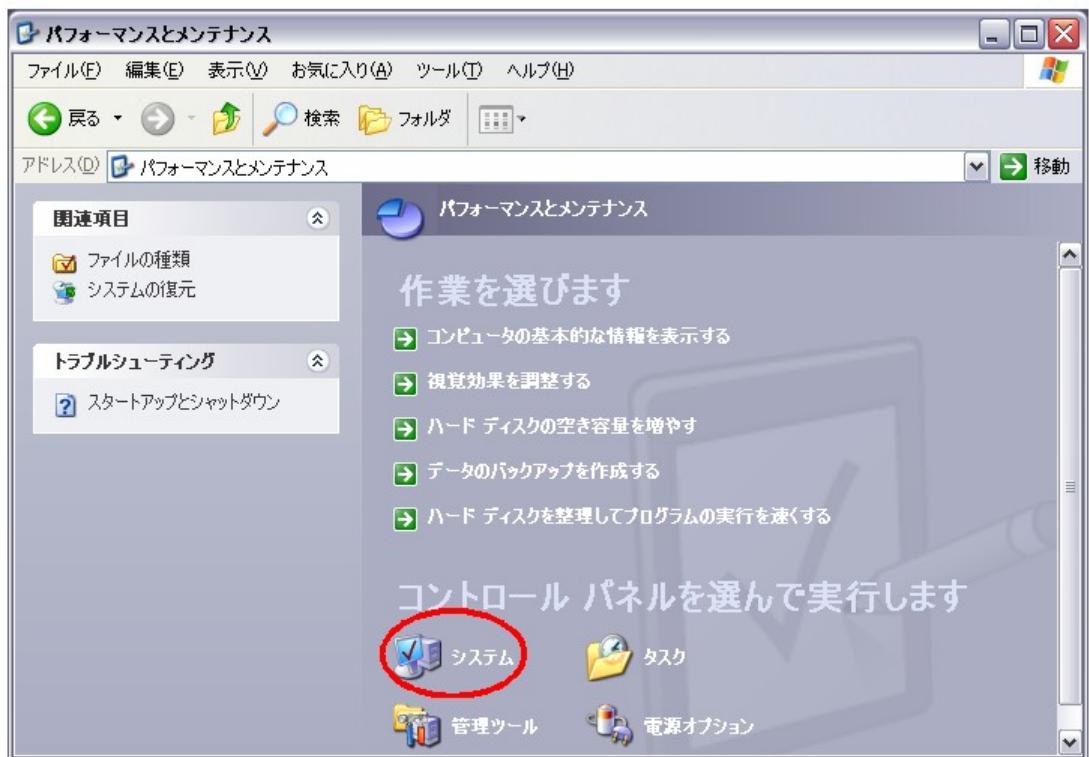
- 「スタートメニュー」から 「コントロールパネル」を選択します。



- 「パフォーマンスとメンテナンス」を選択します。



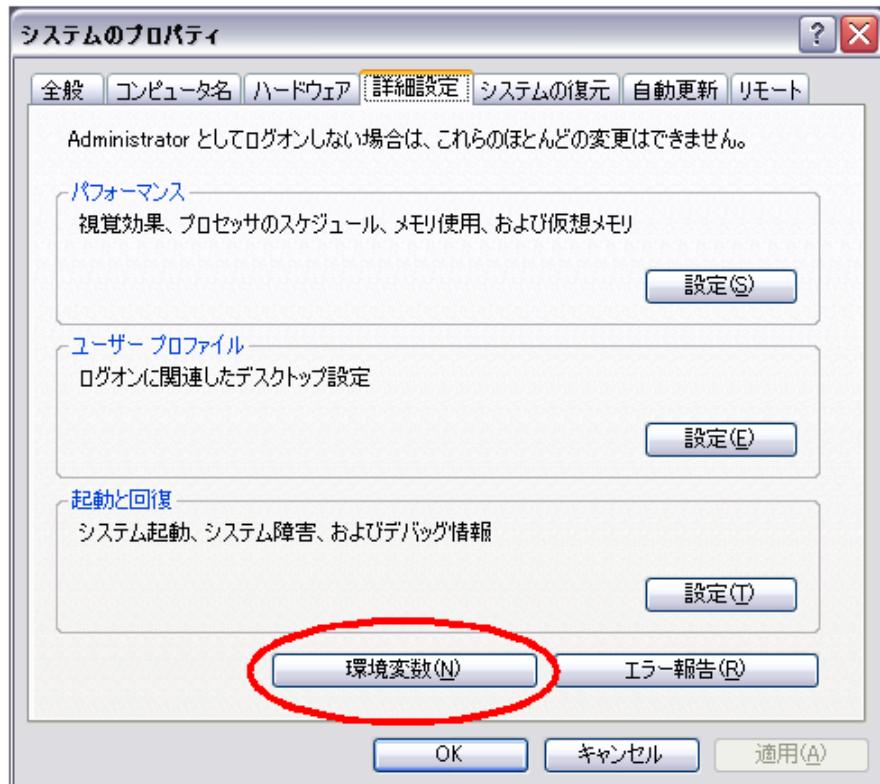
- 「システム」を選択します。



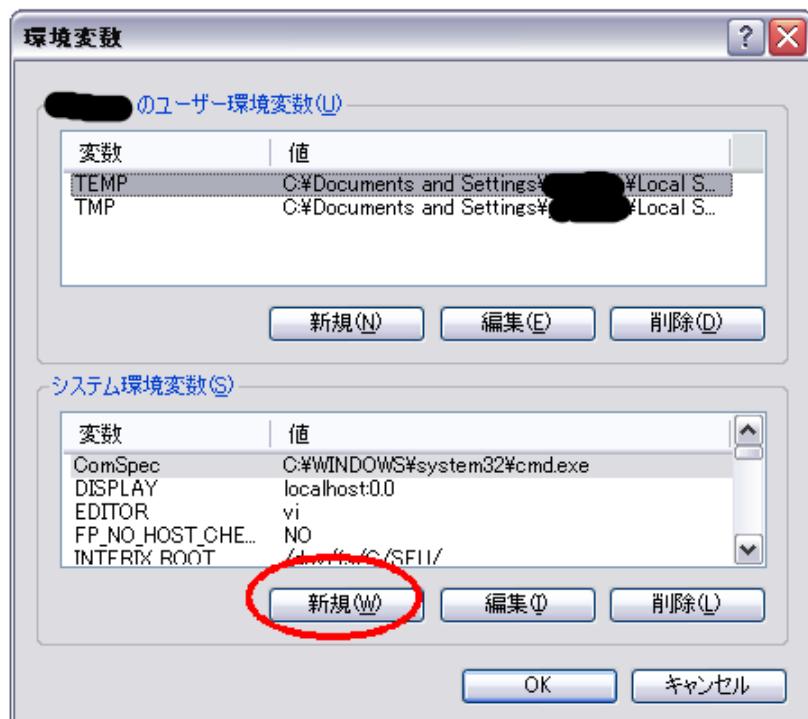
- システムのプロパティダイアログが表示されるので、「詳細設定」を選択します。



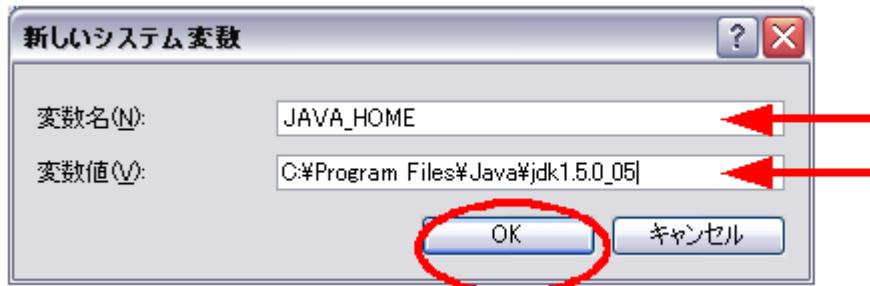
- 「環境変数」を選択します。



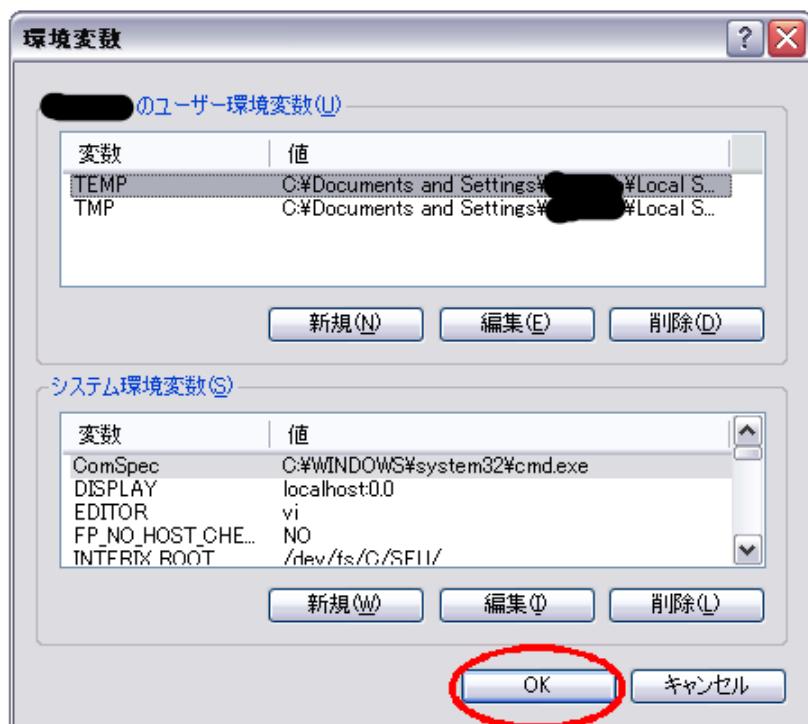
- 「新規」を選択します。



- 変数名に「JAVA_HOME」を、変数値に「C:\Program Files\Java\jdk1.5.0_05」を入力し、「OK」をクリックします。



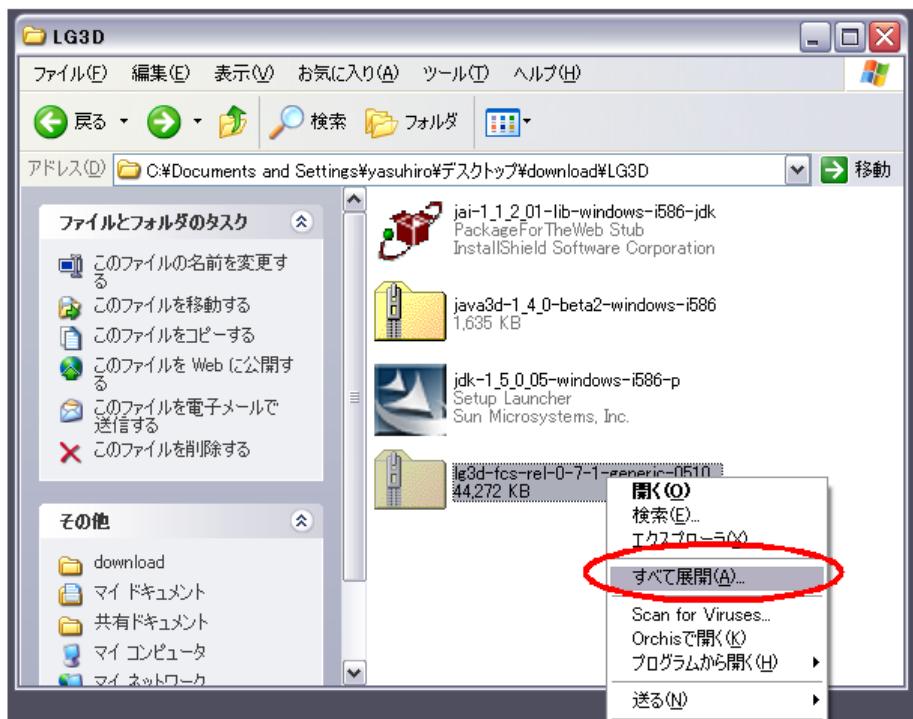
- 「OK」をクリックします。



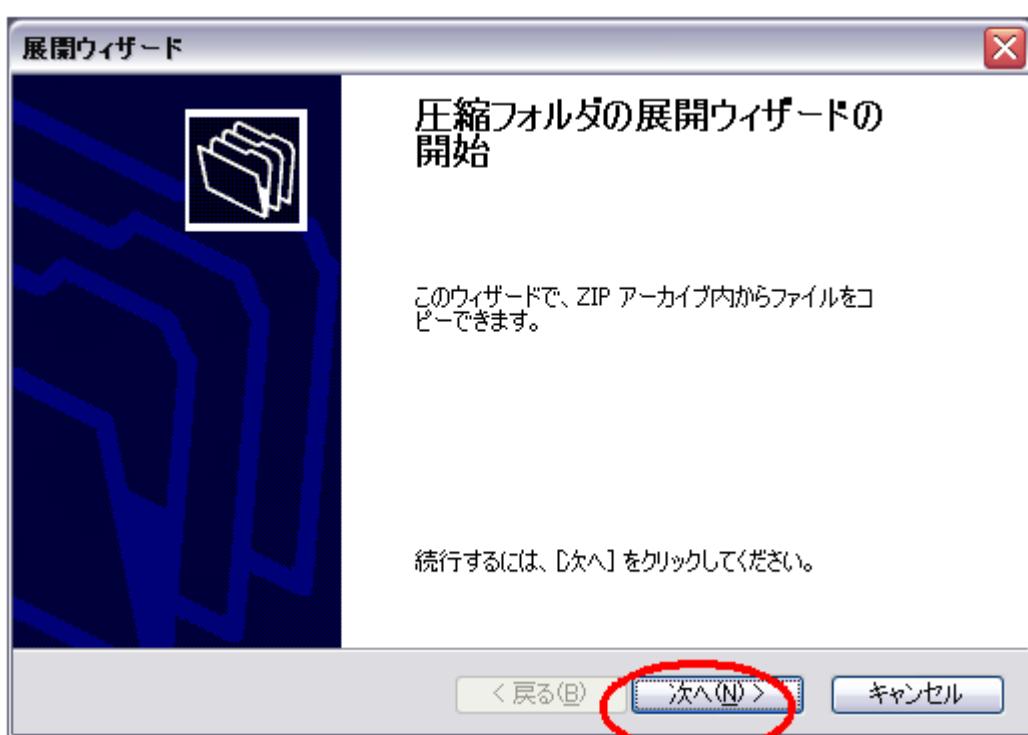
LG3D のインストール

LG3D のインストールはダウンロードしたファイルを解凍するだけです。
ここでは **d:\lg3d** にインストールします。

- **lg3d-fcs-rel-0-7-1-generic-0510281811.zip** 上で右クリックをするとメニューが出ます。
このメニューから「**すべて展開**」を選択します。



- 圧縮フォルダの展開ウィザードが開始します。「**次へ**」をクリックします。



- ファイル展開先フォルダを「d:\」に変更し、「次へ」をクリックします。



- 「完了」をクリックし、ウィザードを終了します。



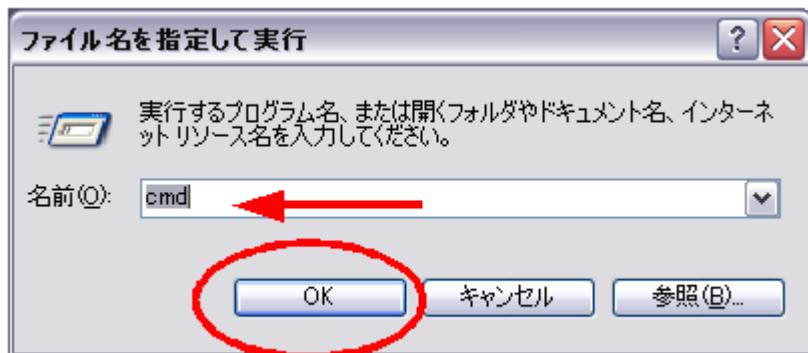
LG3D の実行

LG3D の実行はコマンドプロンプトから行います。
実行手順は以下の通りです。

- 「スタートメニュー」から 「ファイル名を指定して実行」を選択します。



- 名前のフィールドに「cmd」と入力し、「OK」をクリックします。



- lg3d-dev.batのあるディレクトリに移動し、lg3d-dev コマンドを実行します。
d:\lg3d にインストールした場合は次のようにになります。

1. ディスク d: に移動
2. lg3d-dev.batのあるディレクトリ(**lg3d\bin**)に移動
3. lg3d-dev.bat を実行(.batは省略可能)

```

C:\>コマンド プロンプト - lg3d-dev
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\>Documents and Settings\          >d: 1

D:\>cd lg3d\bin 2

D:\>lg3d\bin>lg3d-dev 3
Using config file lgconfig_1p_nox.xml

```

- 下図は実行画面です。



補足

インストールディレクトリの `etc\lg3d\displayconfig\j3d1x1` ファイルの設定を変更することにより LG3D ウィンドウの大きさを変更することが出来ます。 (ここでは `d:\lg3d\etc\lg3d\displayconfig\j3d1x1` になります)
下の例では画面サイズを `1152x864` にしています。 画面サイズの代わりに `NoBorderFullScreen` と指定することでフルスクリーン表示も可能です。
変更する箇所を赤(太字)で示します。

```
/*
*****
*
* Java 3D Calibration file for single-screen desktop configuration with
* neither head tracking nor 6DOF sensor tracking.
*
*****
*/
// Create a new screen object and associate it with a logical name and a
// number. This number is used as an index to retrieve the AWT GraphicsDevice
// from the array that GraphicsEnvironment.getScreenDevices() returns.
//
(NewScreen center 0)

// Set the available image area for a full screen. This is important when
// precise scaling between objects in the virtual world and their projections
// into the physical world is desired through use of an explicit ScreenScale
// view attribute. The defaults are 0.365 meters for width and 0.292 meters
// for height.
//
(ScreenAttribute center PhysicalScreenWidth 0.360)
(ScreenAttribute center PhysicalScreenHeight 0.288)
//(ScreenAttribute center WindowSize      NoBorderFullScreen)
//(ScreenAttribute center WindowSize      (800 600))
(ScreenAttribute center WindowSize      (1152 864))

// Create a view using the defined screen.
//
(NewView view0)
(ViewAttribute view0 Screen center)
(ViewAttribute view0 FrontClipDistance    0.01)
(ViewAttribute view0 BackClipDistance     10.0)
(ViewAttribute view0 WindowEyepointPolicy RELATIVE_TO_COEXISTENCE)
(ViewAttribute view0 WindowMovementPolicy VIRTUAL_WORLD)
(ViewAttribute view0 WindowResizePolicy   VIRTUAL_WORLD)
(ViewAttribute view0 ScreenScalePolicy   SCALE_EXPLICIT)

// For debugging this will give us the standard scale world and view
// but in a window. Obviously if the window is reduced in size less of
// the world is visible.
(ViewAttribute view0 CoexistenceCenteringEnable true)
(ViewAttribute view0 WindowEyepointPolicy RELATIVE_TO_WINDOW)
(ViewAttribute view0 WindowMovementPolicy PHYSICAL_WORLD)

// Enable stereo viewing if desired
// (ViewAttribute view0 StereoEnable       true)
```

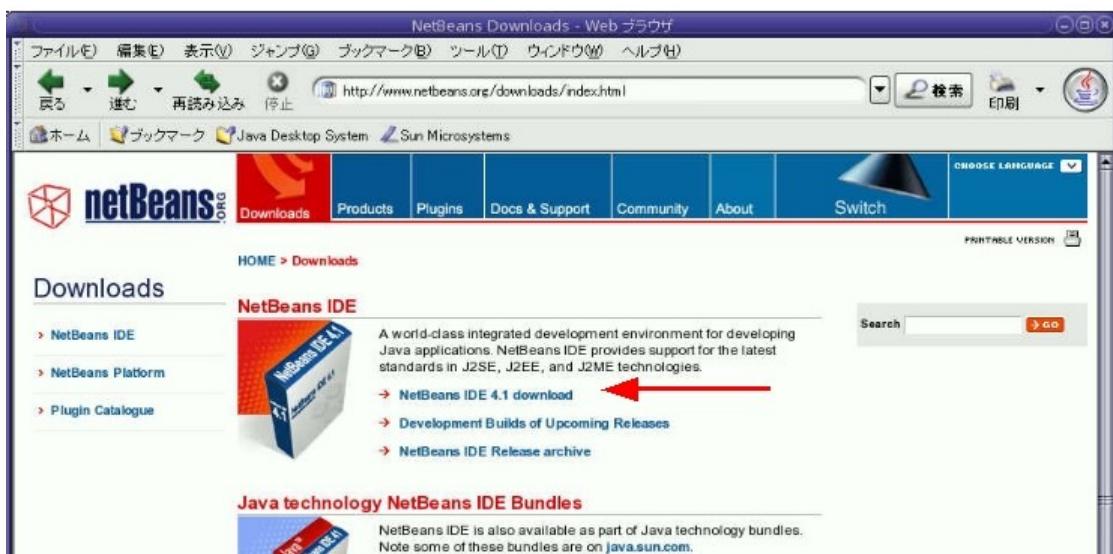
NetBeans のインストール

NetBeans のダウンロード

NetBeans の最新版の <http://www.netbeans.org/> からダウンロードします。

NetBeans 4.1 のダウンロードの手順は次の通りです。

1. ブラウザで <http://www.netbeans.org/downloads/index.html> を開きます。
NetBeans IDE のダウンロードのリンクがいくつか表示されるので、「**NetBeans IDE 4.1 download**」のリンクを選択します。

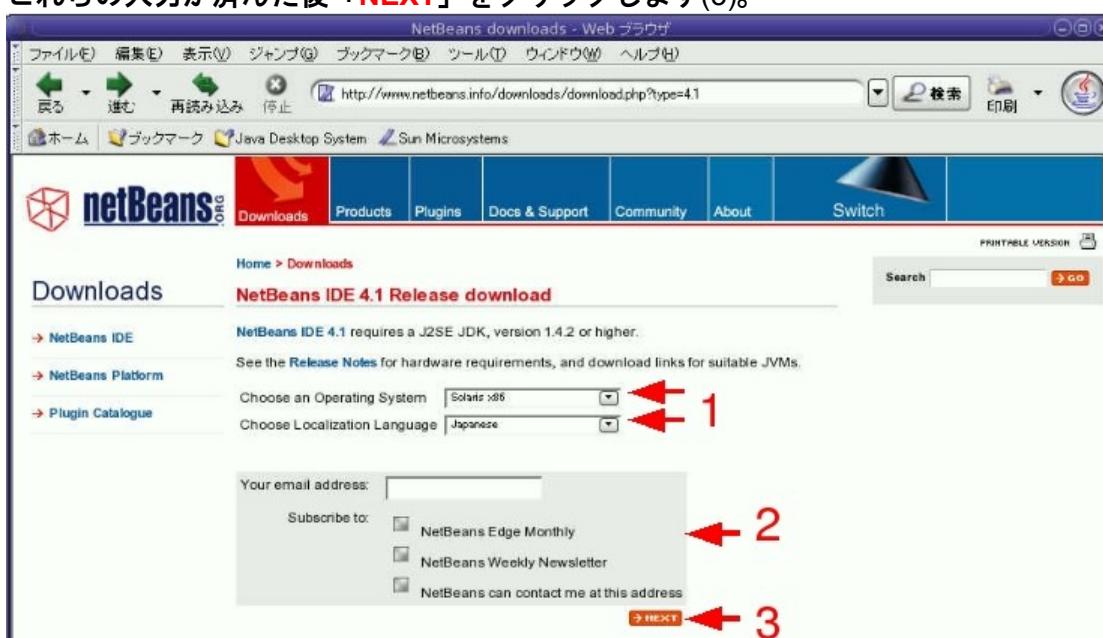


2. 次に NetBeans を利用する環境と対応言語を選択します。

例の場合は「Solaris x86」と「Japanese」を選択しています。

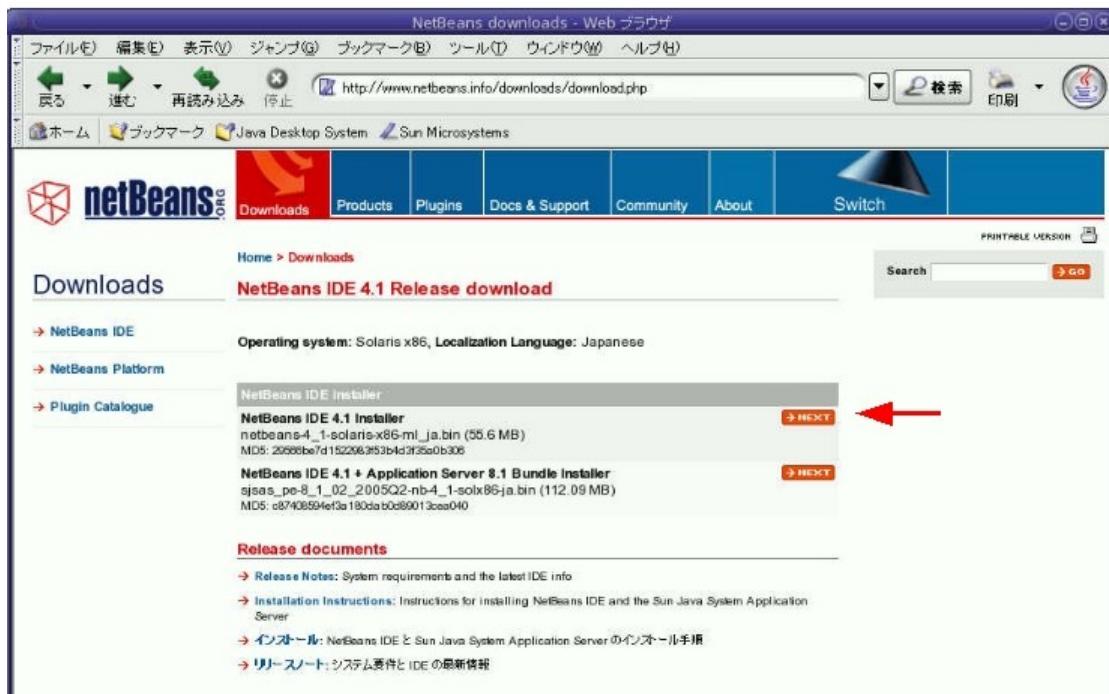
選択後、グレーの部分(2)の入力を行います。NetBeans のメーリングリストなどに登録したい場合はメールアドレスを入力し、チェックボックスをチェックします。例ではメールアドレスをせず、チェックボックスも全て外しています。

これらの入力が済んだ後「NEXT」をクリックします(3)。

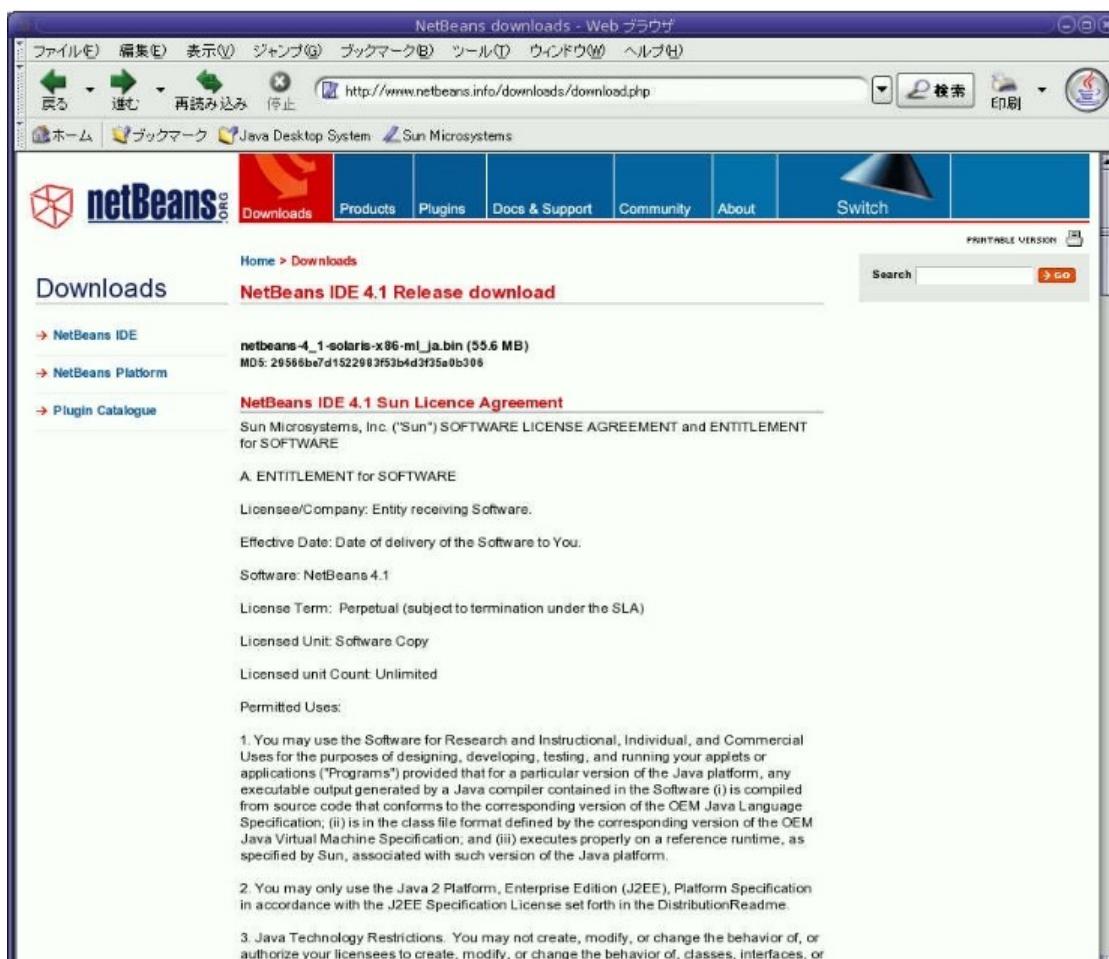


3. NetBeans 単体のインストーラと NetBeans に Application Server がバンドルされたものがあります。

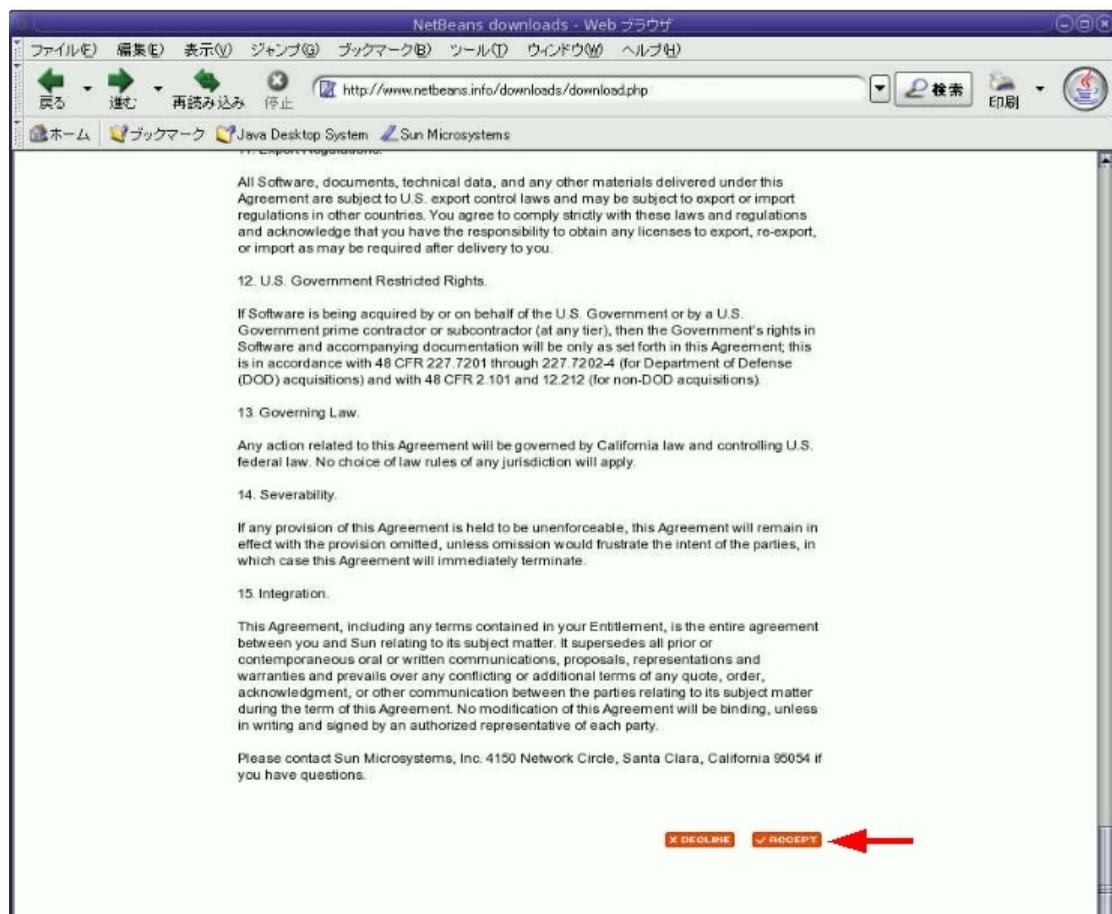
ここでは NetBeans 単体のインストーラ(NetBeans IDE 4.1 Installer)を選択します。



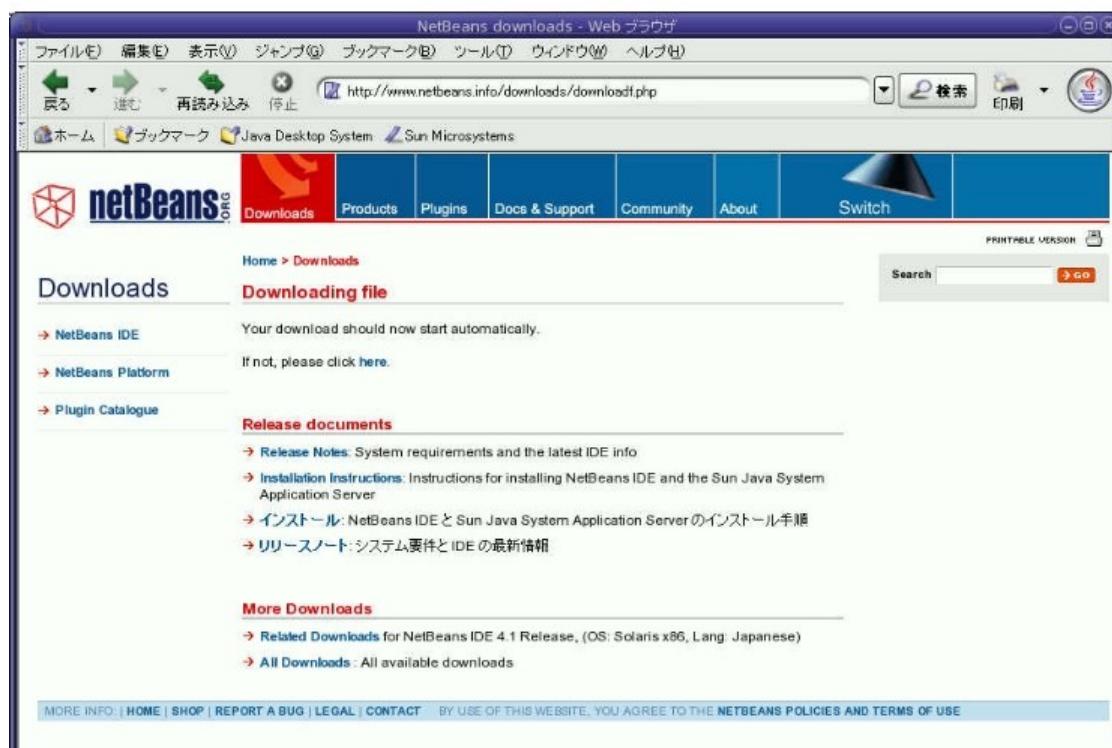
4. ダウンロードを行う前にライセンス条項が表示されますので確認します。



5. 一番下まで進むと「Decline」「Accept」が表示されます。ライセンス条項を受諾してダウンロードする場合は「Accept」をクリックします。



6. 以下の画面が表示されると同時にダウンロードが開始します。



NetBeans のインストール

NetBeans 4.1 のインストール手順は以下の通りです。

ルートユーザでインストーラを起動した場合 /opt がデフォルトのインストール先になります。
一般ユーザで起動した場合はホームディレクトリがデフォルトのインストール先になります。
ここではルートユーザでインストールを行います。

1. ダウンロードしたファイル([netbeans-4_1-solaris-x86-ml_ja.bin](#))の実行権限を変更し、実行します。

```
# chmod 755 netbeans-4_1-solaris-x86-ml_ja.bin
# ./netbeans-4_1-solaris-x86-ml_ja.bin
InstallShield Wizard

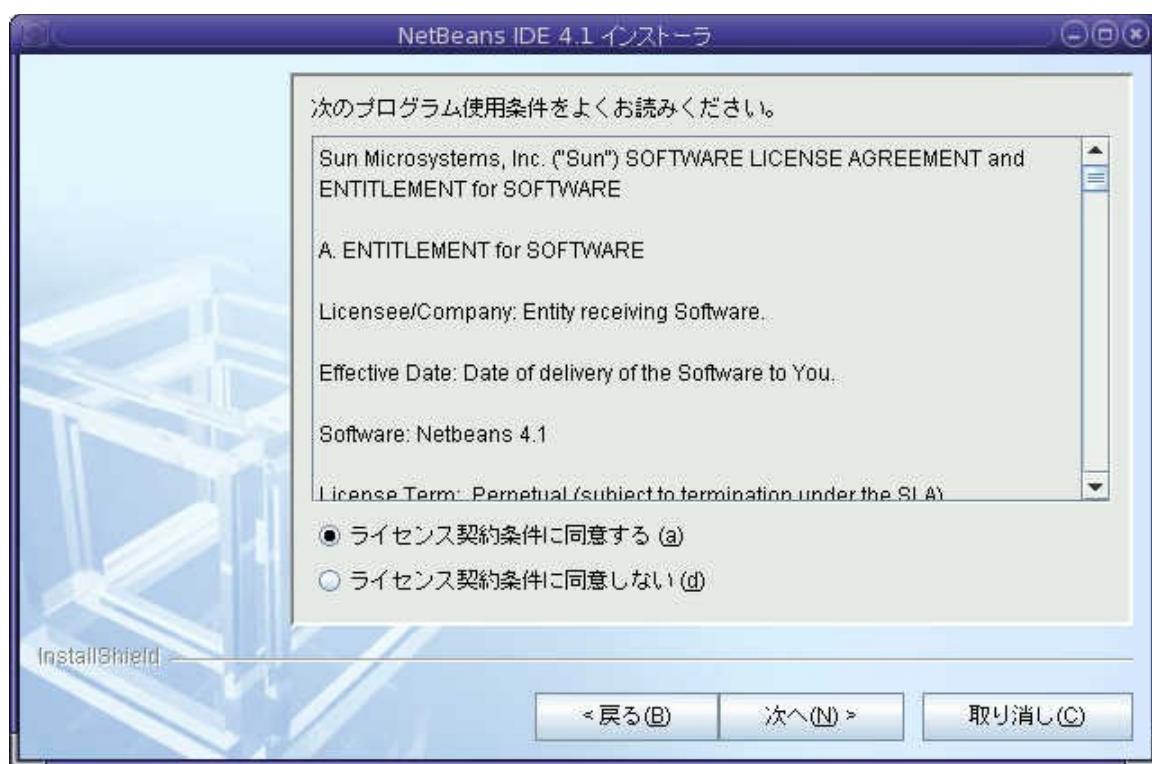
InstallShield Wizard を初期化中です...

Java(tm) 仮想マシンを検索中です...
.....
```

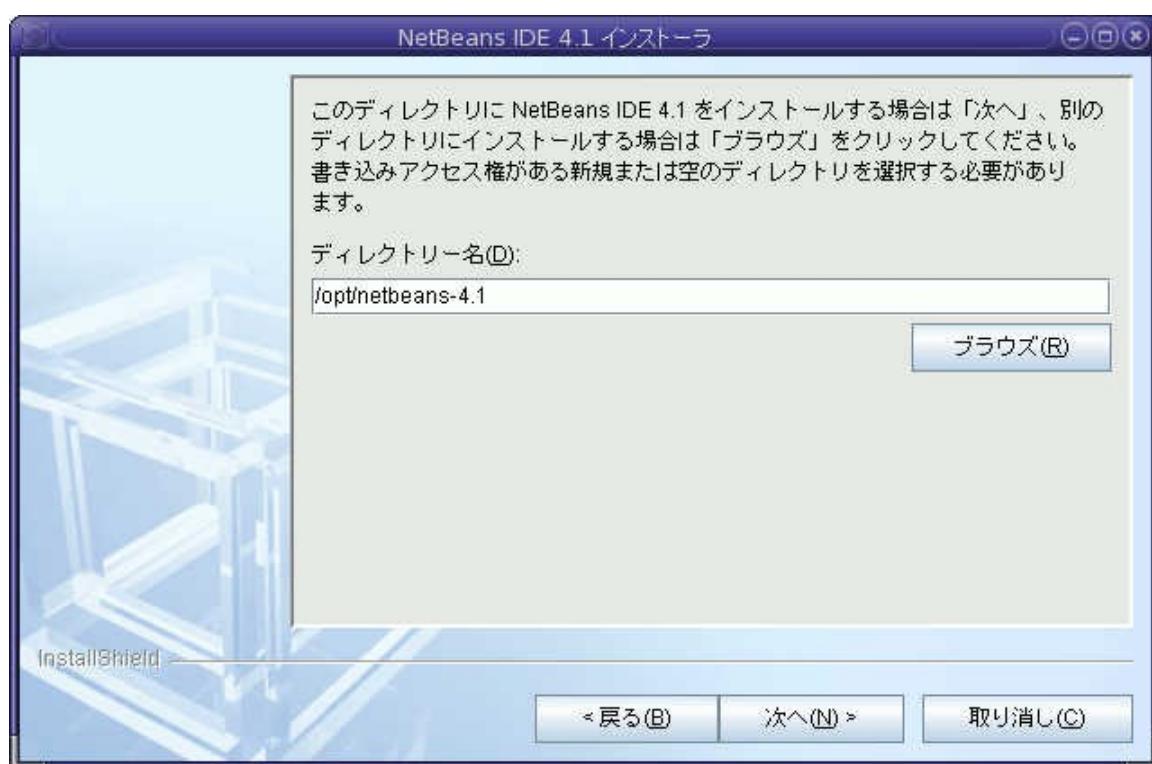
1. インストーラが起動しますので、「**次へ**」をクリックします。



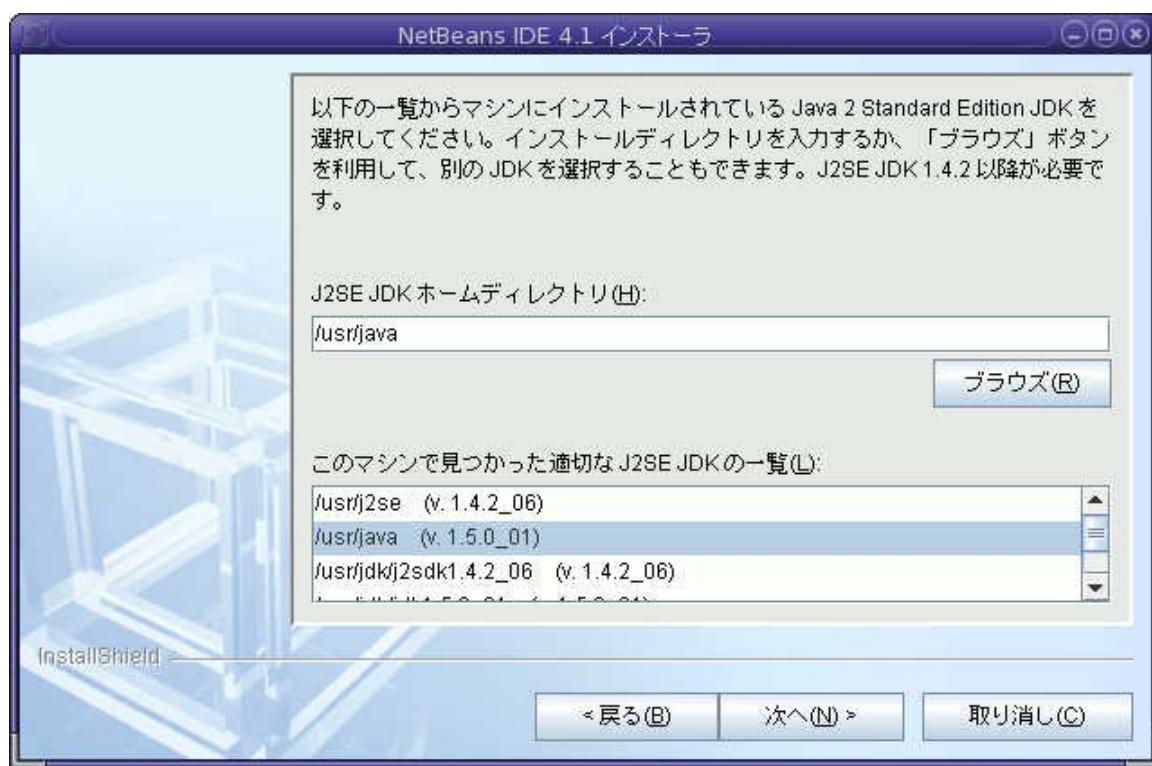
2. ライセンス条項を確認します。
「**ライセンス契約条件に同意する**」のチェックボックスを選択後、「次へ」をクリックします。



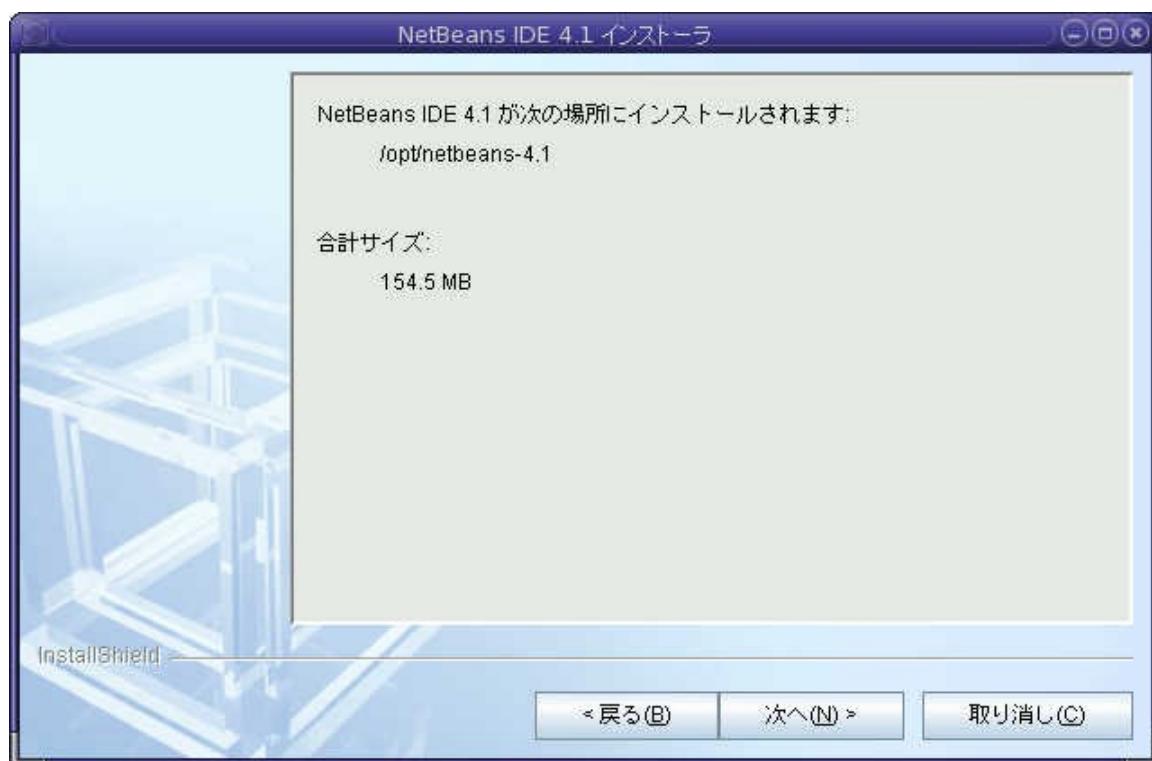
3. インストールするディレクトリを入力します。
ここではデフォルト(**/opt/netbeans-4.1**)をインストール先とします。
「次へ」をクリックします。



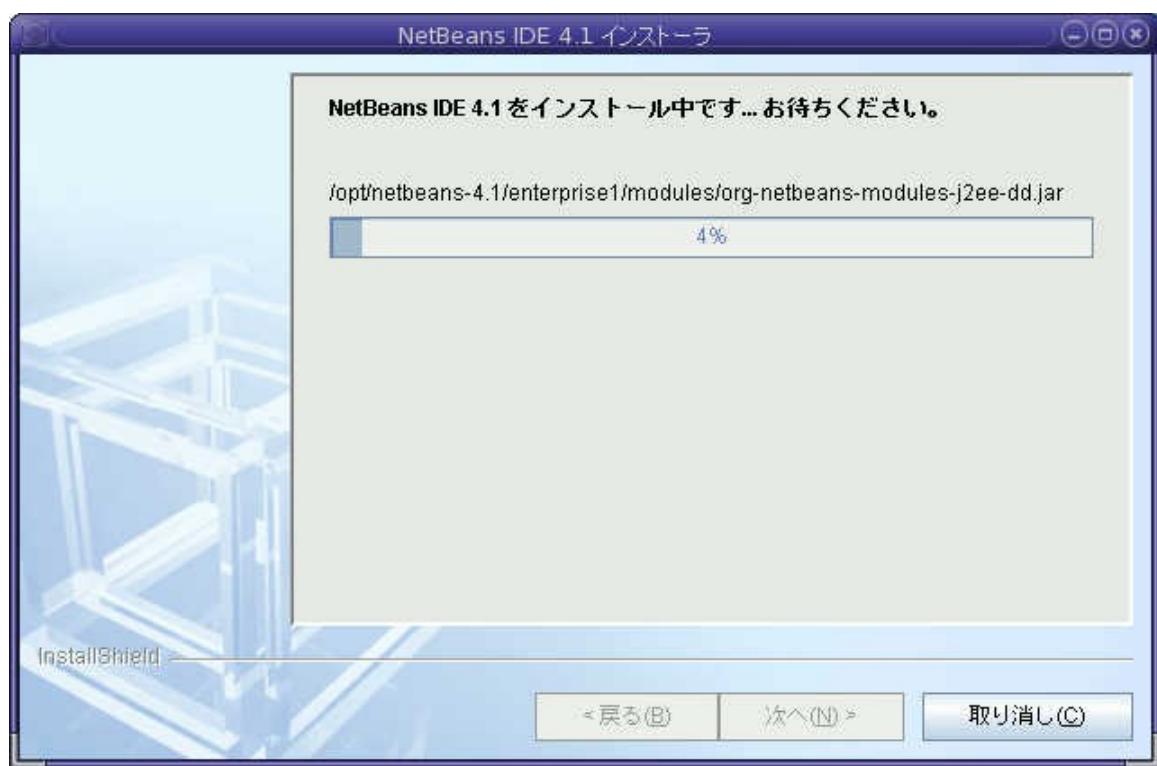
4. 使用する JDK を指定します。 選択後「次へ」をクリックします。



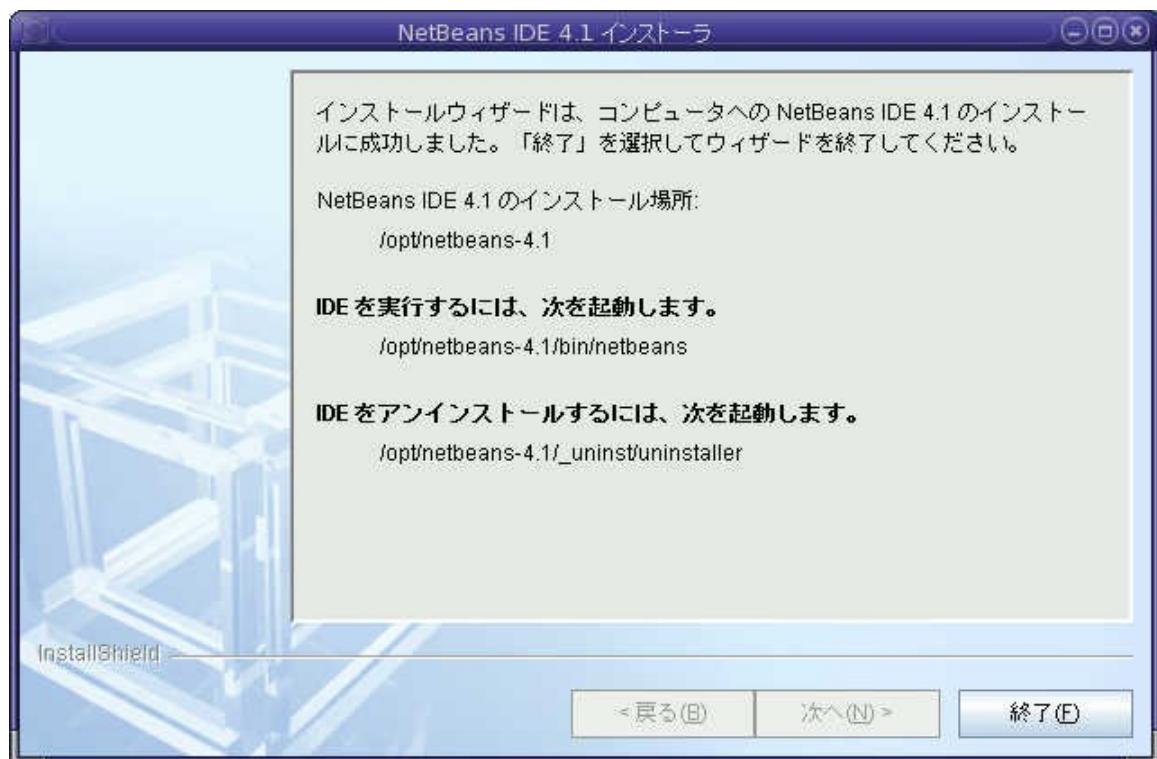
5. インストールの確認画面が表示されるので、「次へ」をクリックします。



6. インストール中の画面です。



7. 「終了」をクリックし、インストーラを終了します。



NetBeans のセットアップ

NetBeans のインストールが終了したら、次に LG3D でプログラミングを行うための準備を行います。準備の内容は「[LG3D 用のコンパイラ環境の登録](#)」と「[LG3D API の登録](#)」です。

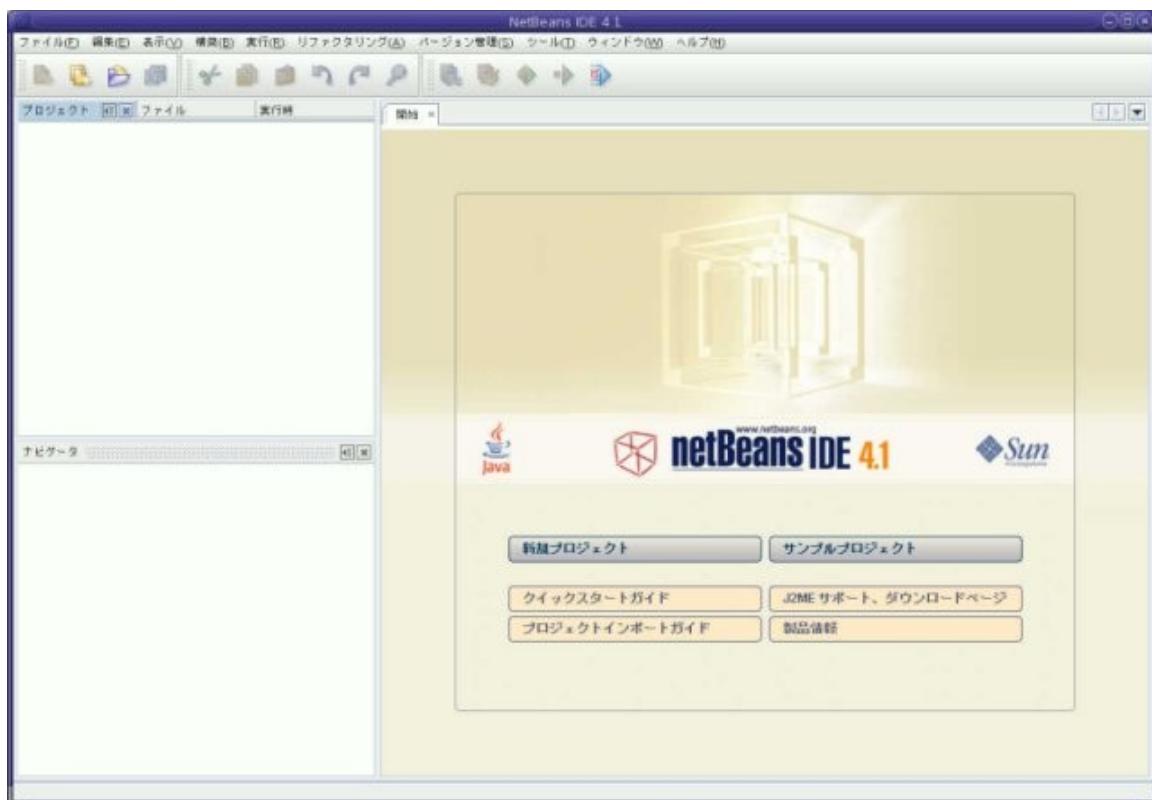
NetBeans の起動

NetBeans の起動を行います。

NetBeans は端末エミュレータ上で次のコマンドを実行します。

```
# /opt/netbeans-4.1/bin/netbeans
```

コマンドを入力すると NetBeans 4.1 の起動画面が表示されます。しばらく待つと IDE が起動します。



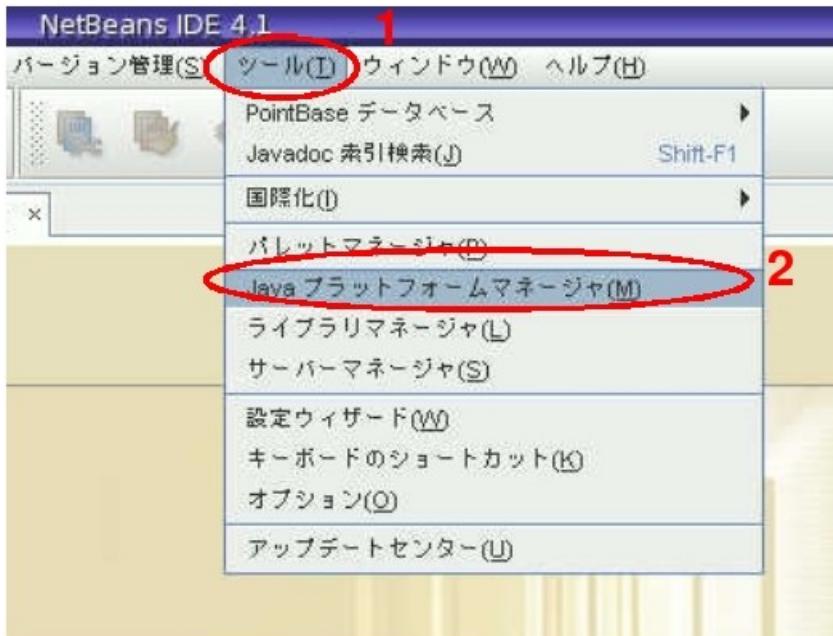
Platform Manager

Platform Manager は JDK 環境を管理しています。

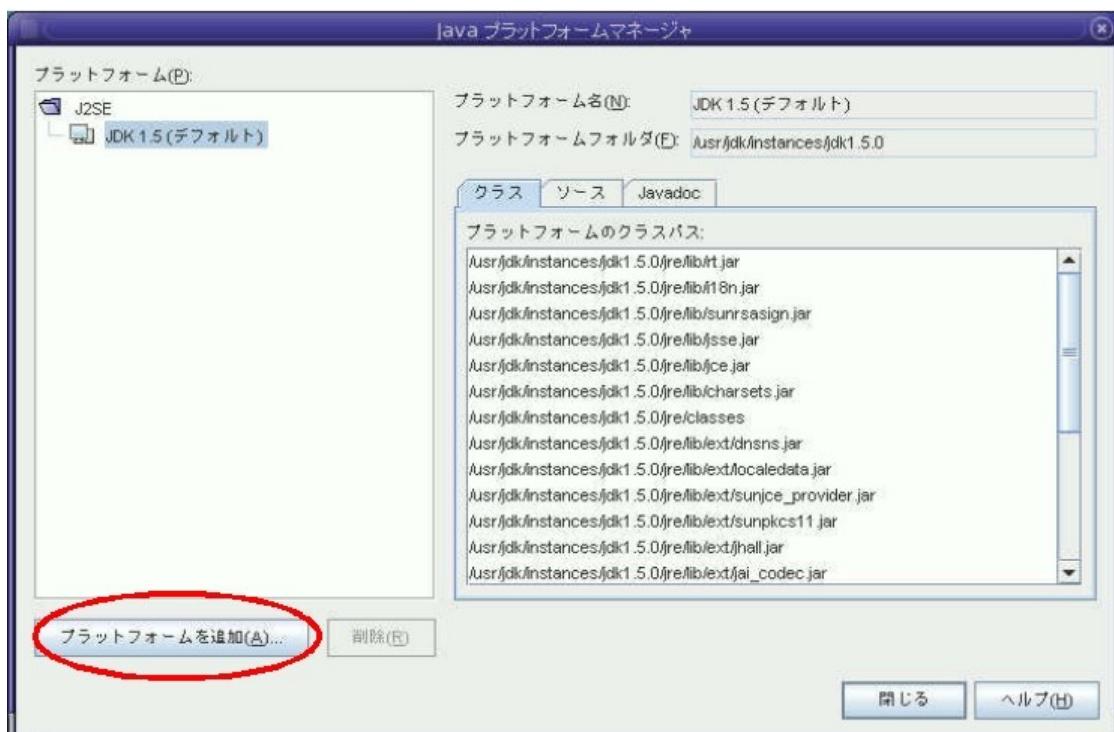
ここでは、システム標準の JDK とは別に あらかじめインストールしておいた LG3D 専用の JDK(+JAI,Java3D)環境を登録します。

登録手順は次のようになります。

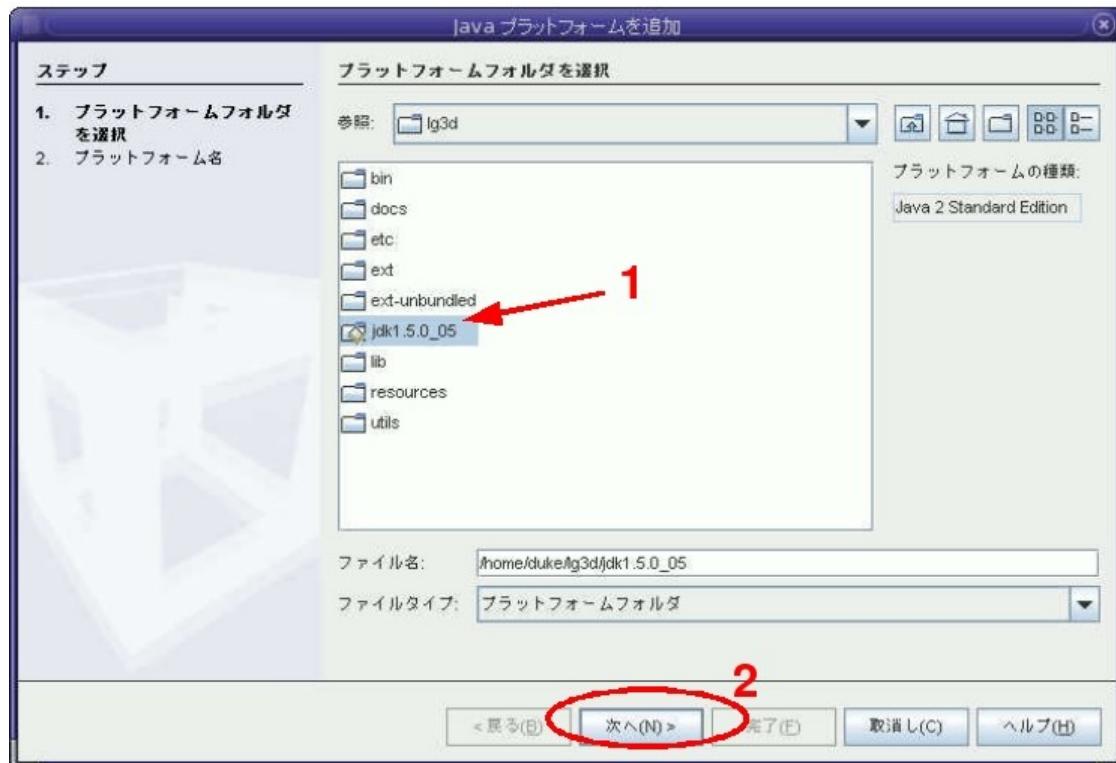
1. NetBeans のメニューから 「ツール」 → 「Java プラットフォームマネージャ」を選択します。



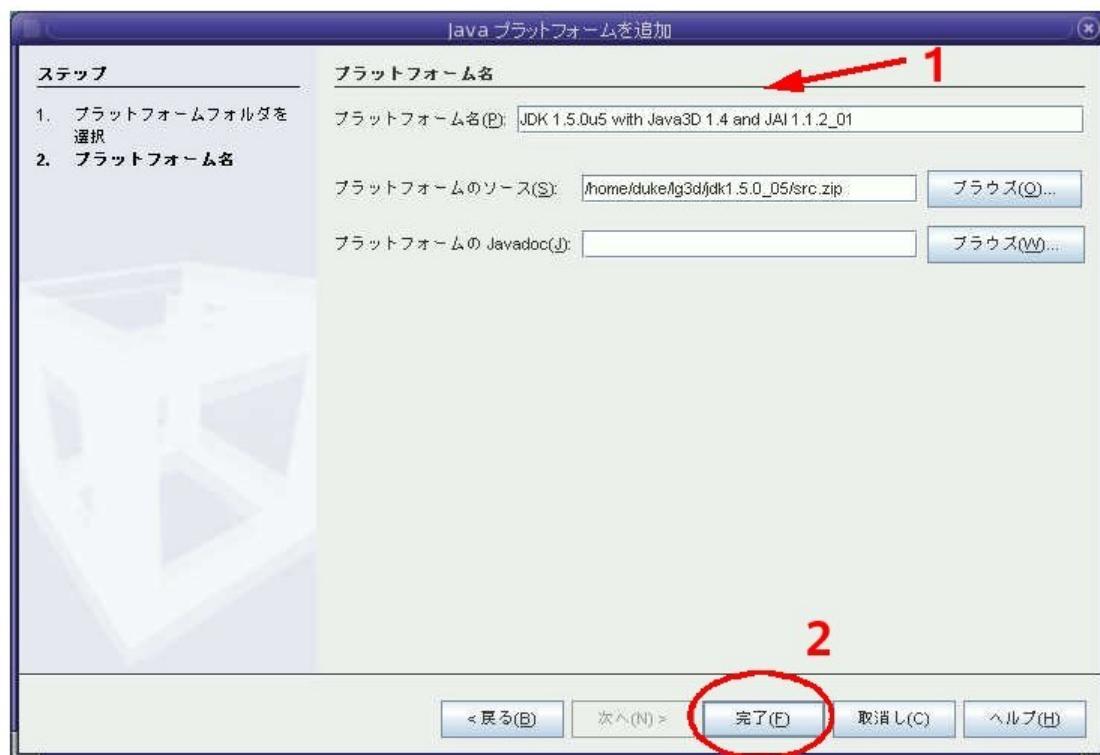
2. Java プラットフォームマネージャダイアログが表示されるので、「**プラットフォームを追加**」をクリックします。



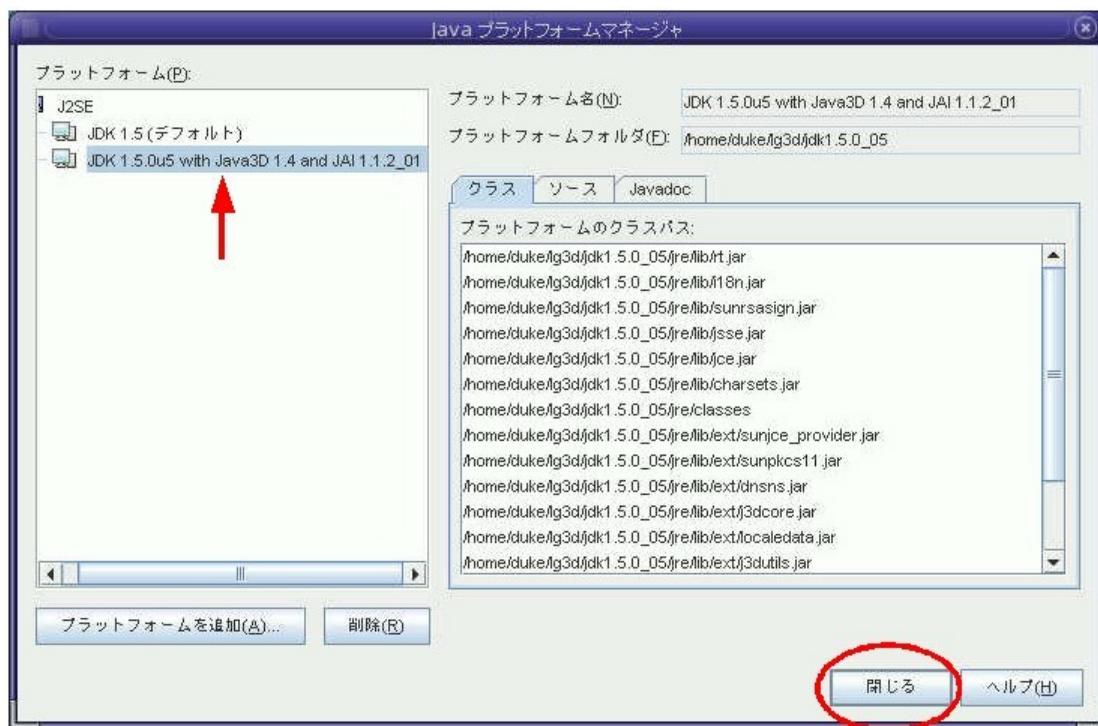
3. ファイラが表示されるので JDK のフォルダ(ディレクトリ)を選択します。
ここでは **/home/duke/lg3d/jdk1.5.0_05** となります。
JDK のディレクトリを選択後「次へ」をクリックします。



4. プラットフォーム名の確認が出ますので適宜変更します。
ここでは「**JDK 5.0u5 with Java3D 1.4 and JAI 1.1.2_01**」とします。
プラットフォーム名を入力後「完了」を押します。



5. Java プラットフォームマネージャに追加されていることを確認し、ウィンドウを閉じます。

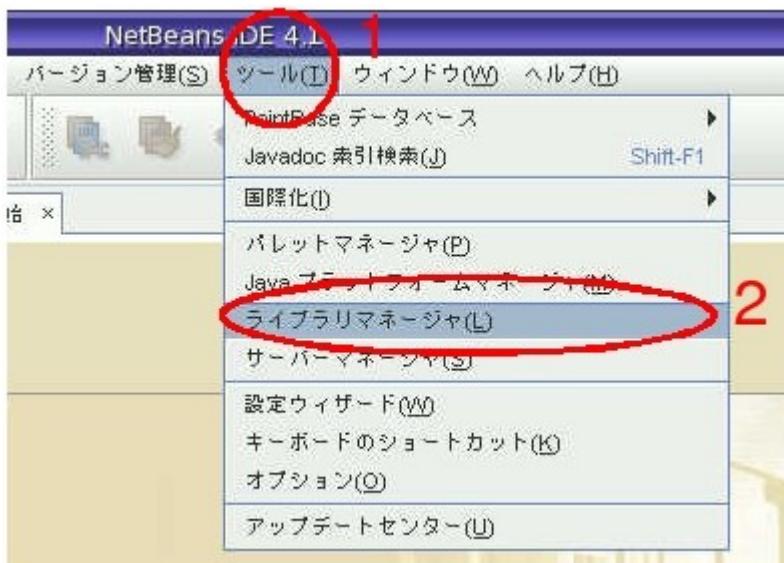


Library Manager

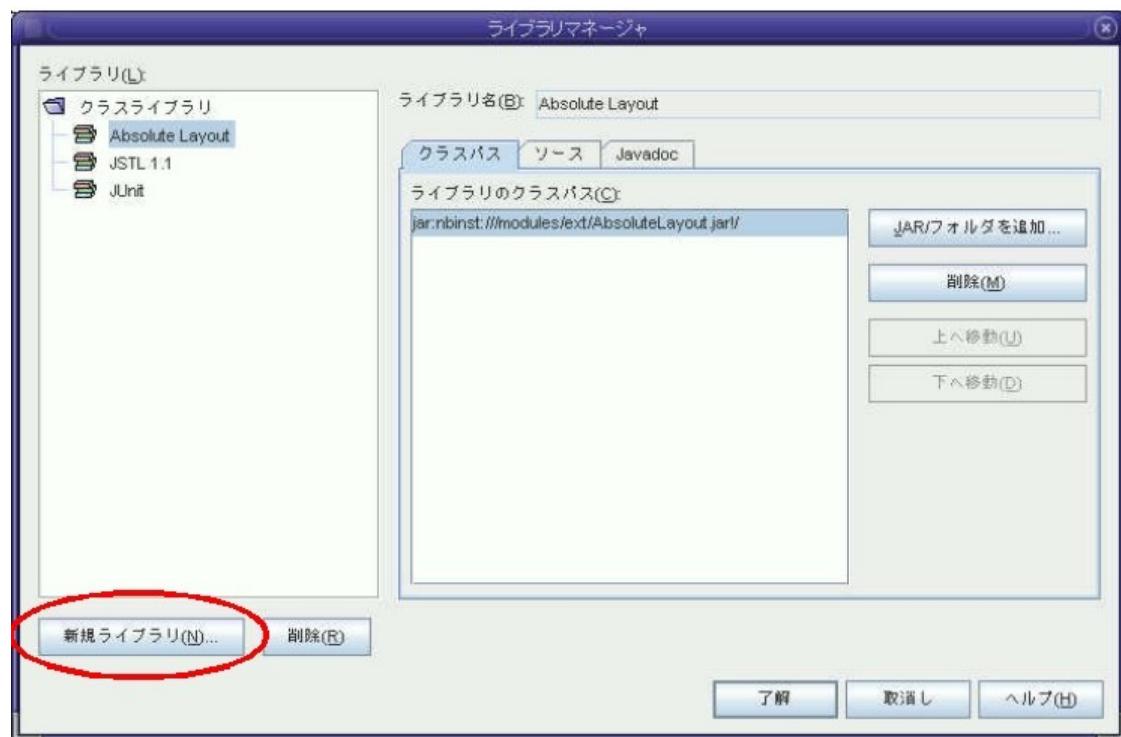
Library Manager は外部の API(ライブラリ)の管理を行うためのツールです。

ここでは、LG3D プログラミングを行うために lg3d-core.jar(lg3d-core の API 群) を登録します。
登録手順は次の通りです。

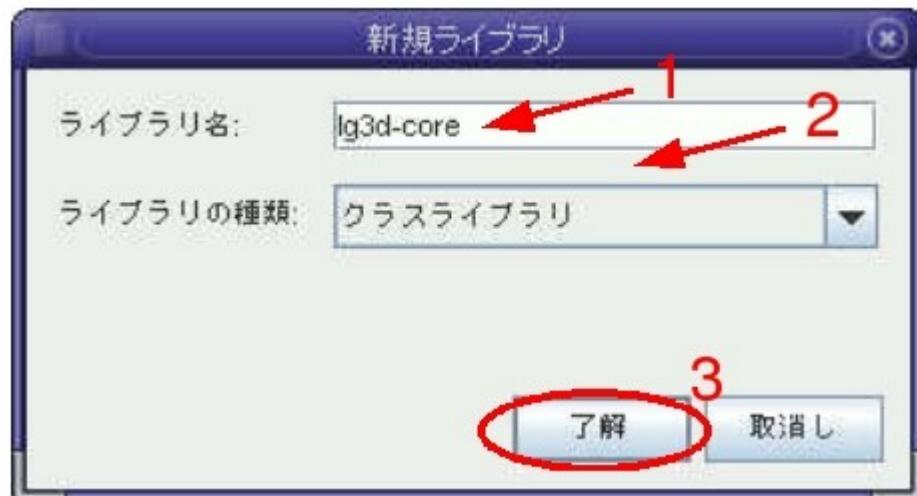
1. NetBeans のメニューから 「ツール」 → 「ライブラリマネージャ」を選択します。



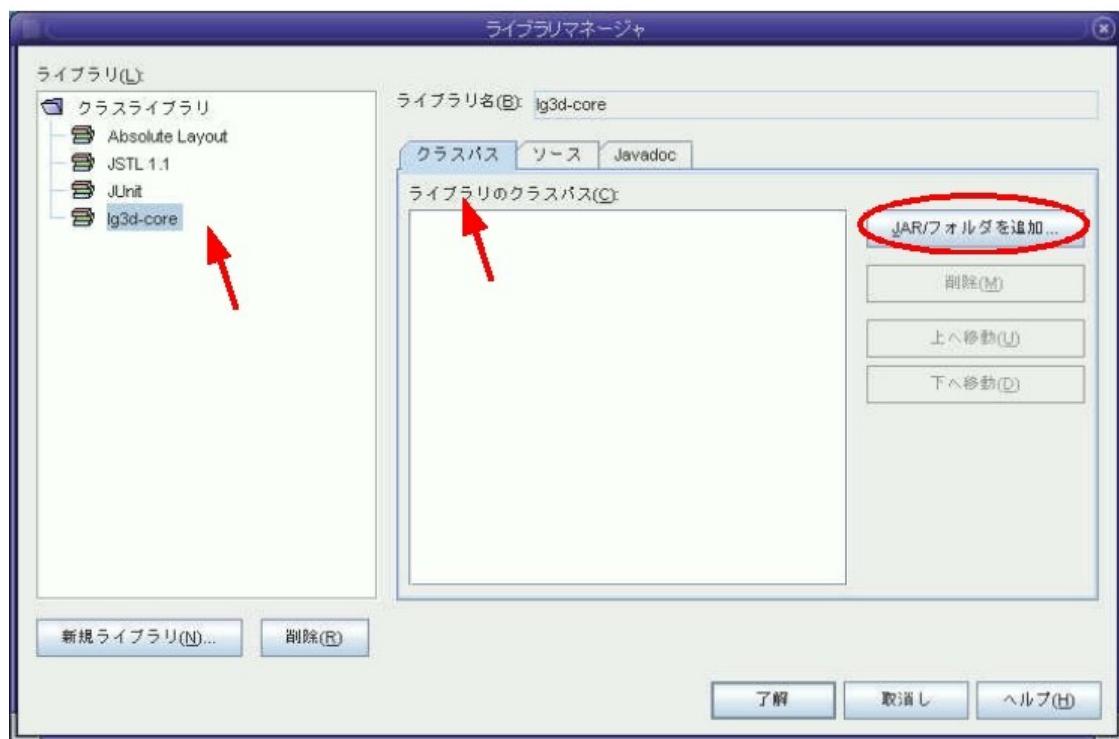
2. ライブラリマネージャダイアログが表示されますので、「新規ライブラリ」を選択します。



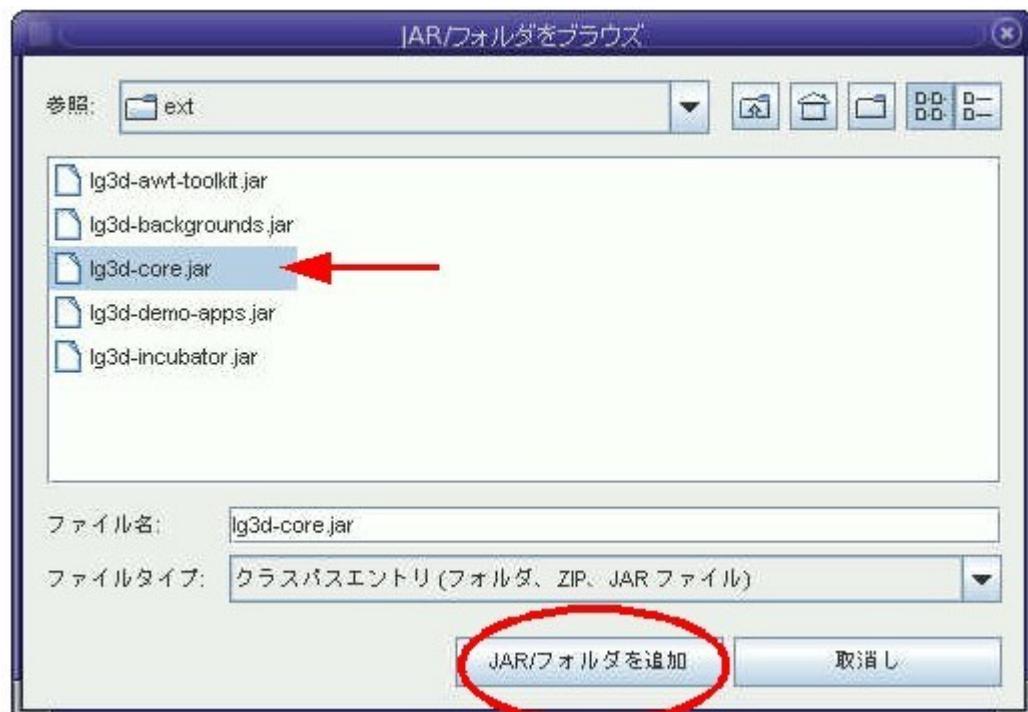
3. 新規ライブラリダイアログが表示されるのでライブラリ名を入力します。ここでは **Ig3d-core** にします。
また、ライブラリの種類を「**クラスライブラリ**」にします。
入力後「**了解**」をクリックします。



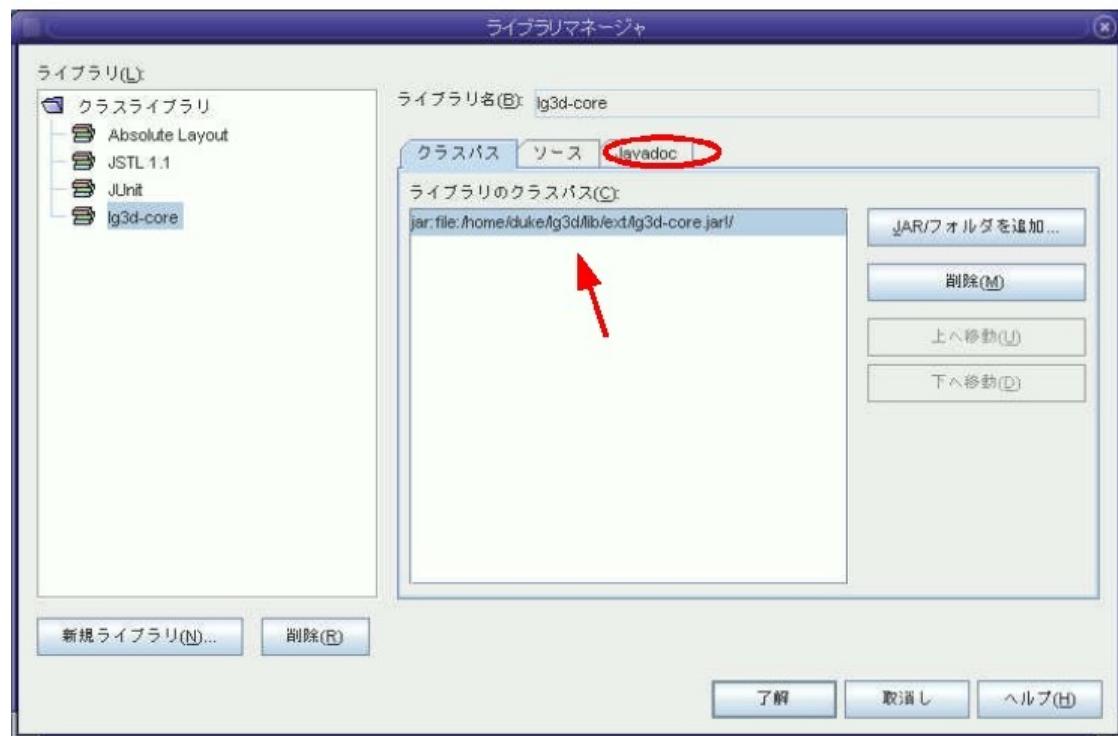
4. ライブラリマネージャダイアログに「lg3d-core」があることを確認し、それを選択します。
右側が「**クラスライブラリ**」となっていることを確認し、「**JAR/フォルダの追加**」を選択します。



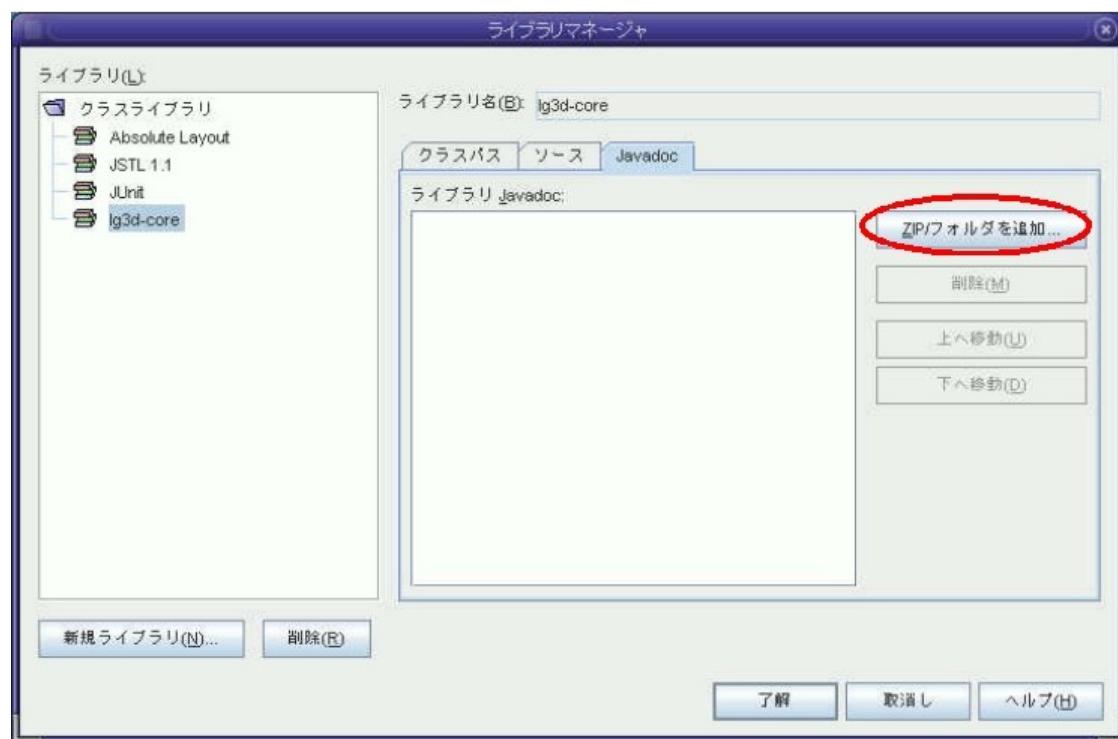
5. ファイルが表示されるので、LG3D をインストールしたディレクトリ
(`/home/duke/lg3d/lib/ext`)に移動します。
lg3d-core.jar を選択し、「**JAR/フォルダの追加**」をクリックします。



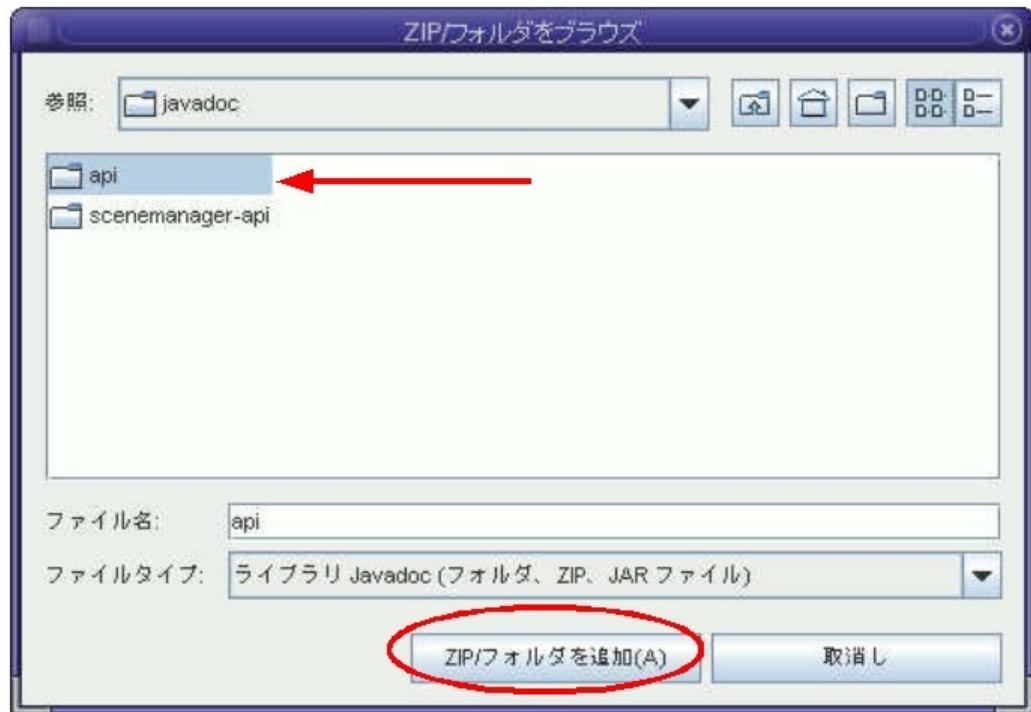
6. ライブラリマネージャにJARファイルが追加されたことを確認します。
確認後、「**Javadoc**」を選択して、クラスパスの表示からJavadocの表示に切替えます。



7. 「ZIP/フォルダを追加する」をクリックします。



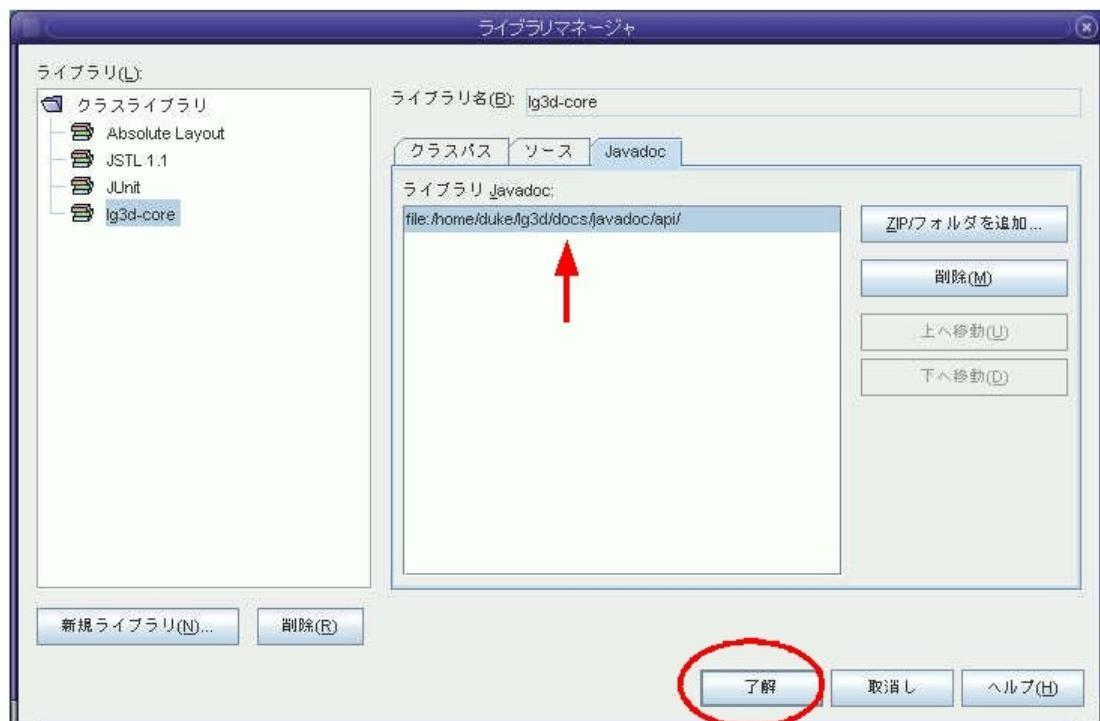
8. LG3D の API ドキュメント(</home/duke/lg3d/docs/javadoc/api>) を選択し、「ZIP/フォルダを追加」をクリックします。



9. Javadoc にフォルダ(ディレクトリ)が登録されたことを確認します。確認後、「了解」をクリックします。

補足:

/home/duke/lg3d/docs/javadoc/scenemanager-api に含まれる内容は LG3D のシステムレベルの開発 (シーンマネージャの開発)で用います。
通常の LG3D アプリケーションの開発では使わないので **登録する必要はありません。**



4.1 プロジェクトの作成

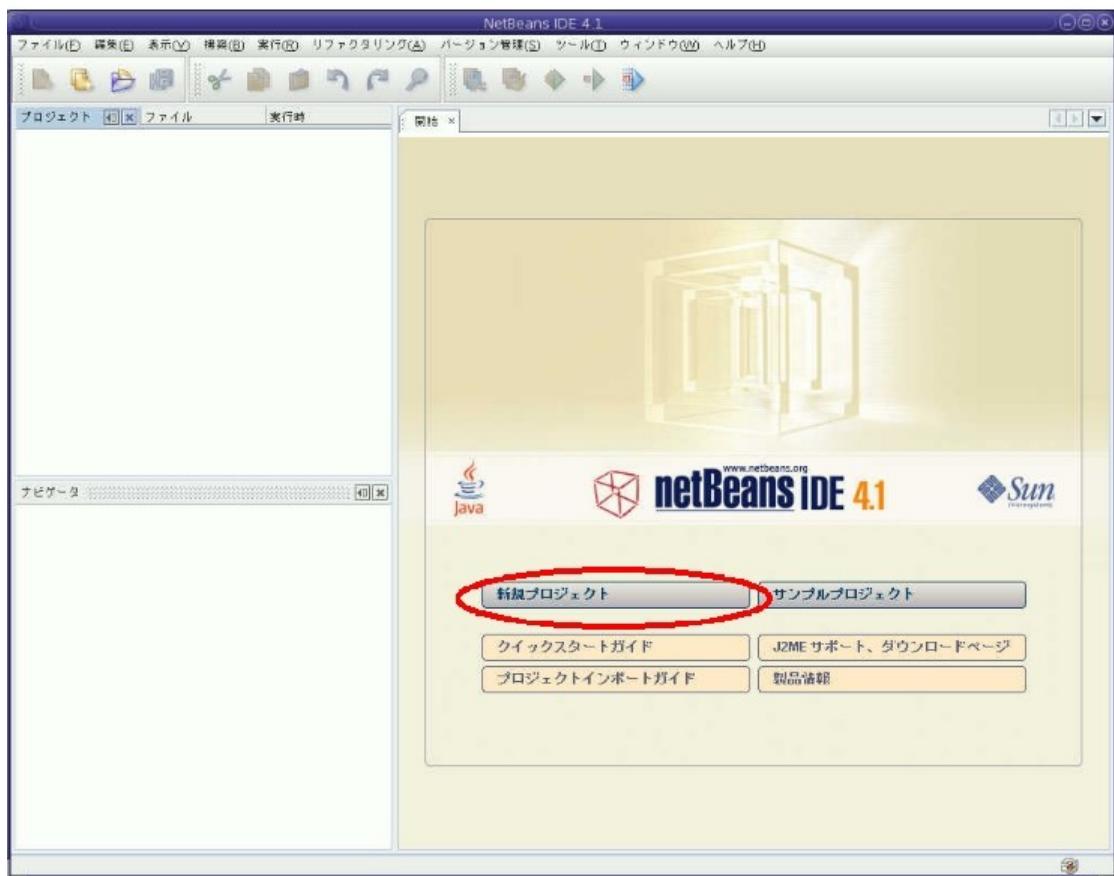
この章で行うこと：

NetBeans を用いて LG3D アプリケーションを作成するために 以下のことを行います。

- ・ LG3D アプリケーションを作成するためのプロジェクトの作成
- ・ プログラミング環境のセットアップ
 - ・ Java プラットフォームの設定
 - ・ ライブラリの設定

プロジェクトの作成

1. NetBeans 4.1 の起動画面で 「新規プロジェクト」を選択するか、「ファイル」→「新規プロジェクト」を選択します。



2. プロジェクト作成用のダイアログが表示されますので、

- ・ カテゴリ : **一般**
- ・ プロジェクト : **Java アプリケーション**

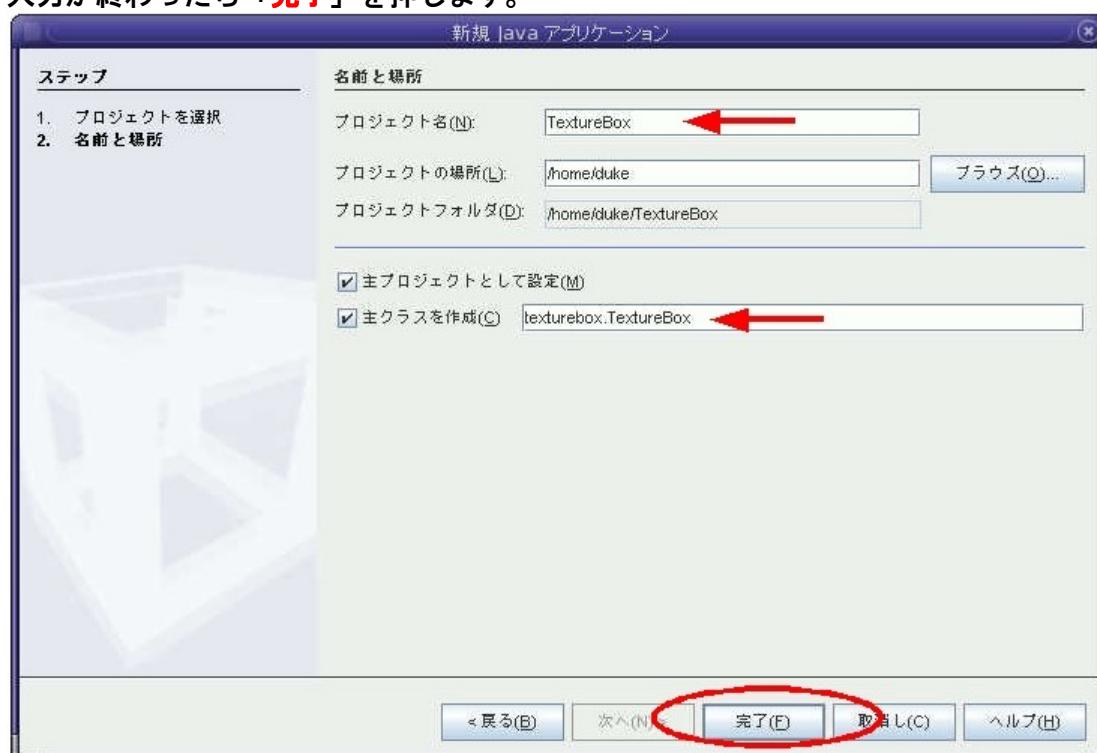
を選択します。確認後、「次へ」ボタンを押します。



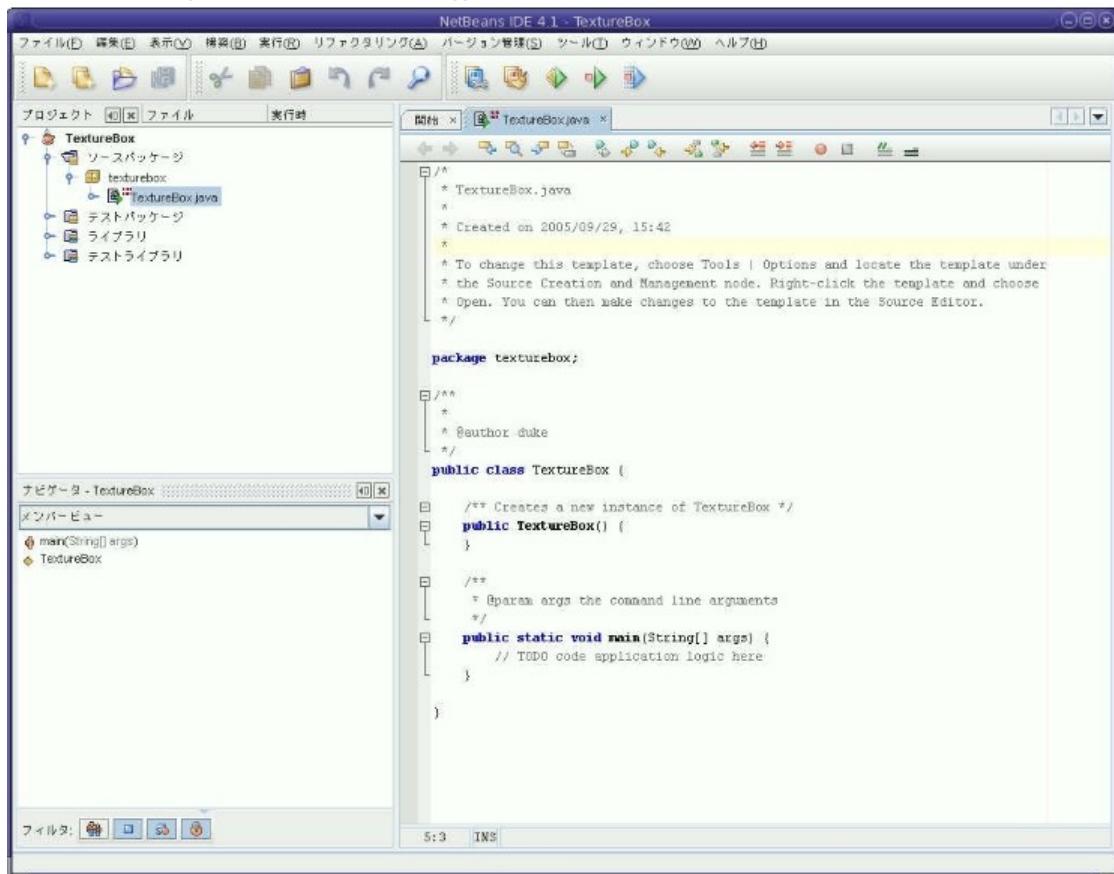
3. プロジェクトの情報を入力求められますので、

- ・ プロジェクト名 : **TextureBox**
- ・ 主クラスを作成 : **texturebox.TextureBox**

に変更します。パッケージ名はデフォルトの `texturebox` をそのまま利用します。
入力が終わったら「完了」を押します。

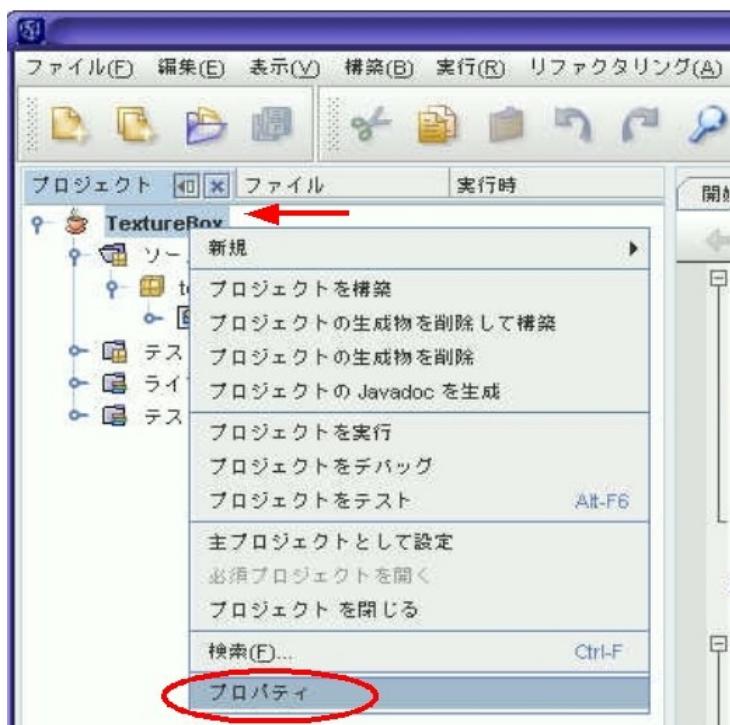


4. クラスなどを含むプロジェクトが作成され、下図のようになります。

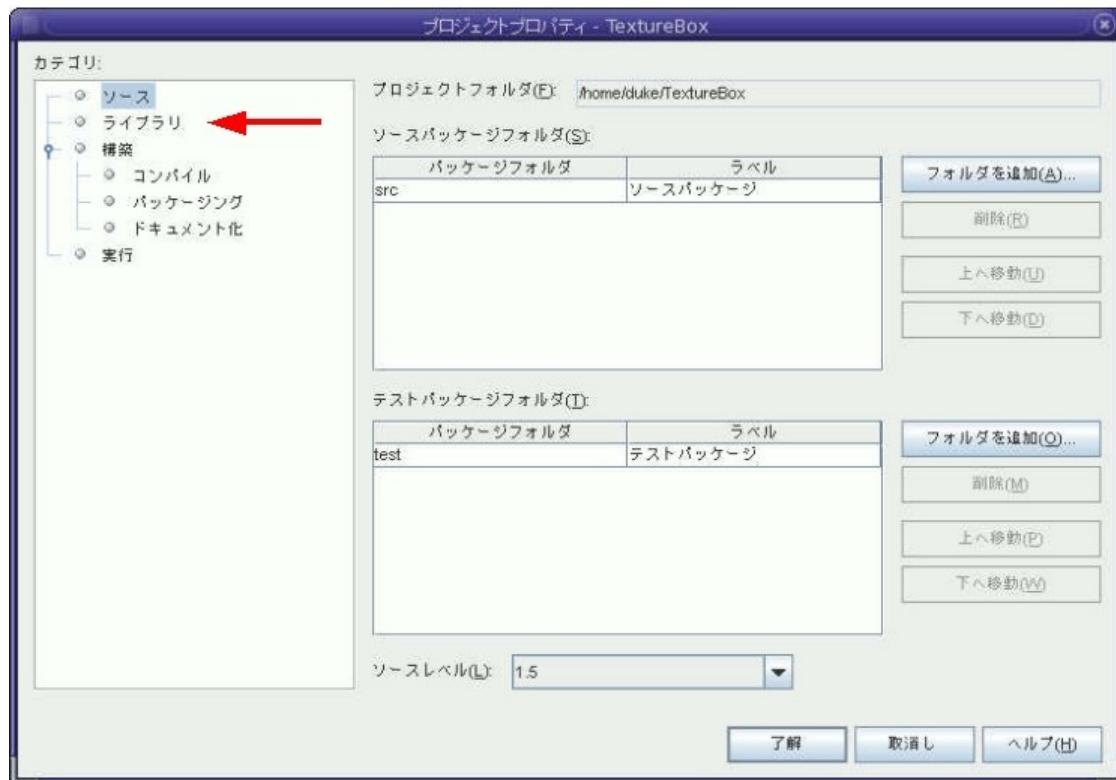


プログラミング環境、実行環境の設定

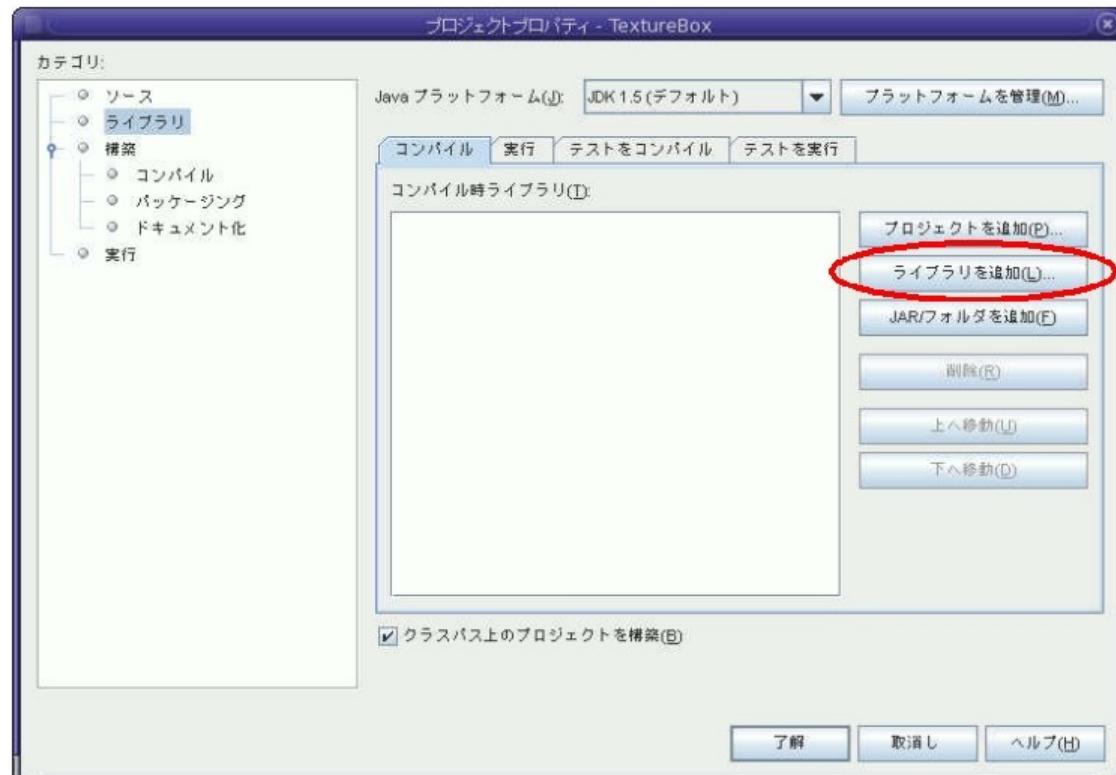
1. IDE の左上部にプロジェクトの情報が表示されていることを確認し 「TextureBox」を選択します。選択後、右クリックを押すことでメニューが表示されますので 「プロパティ」を選択します。



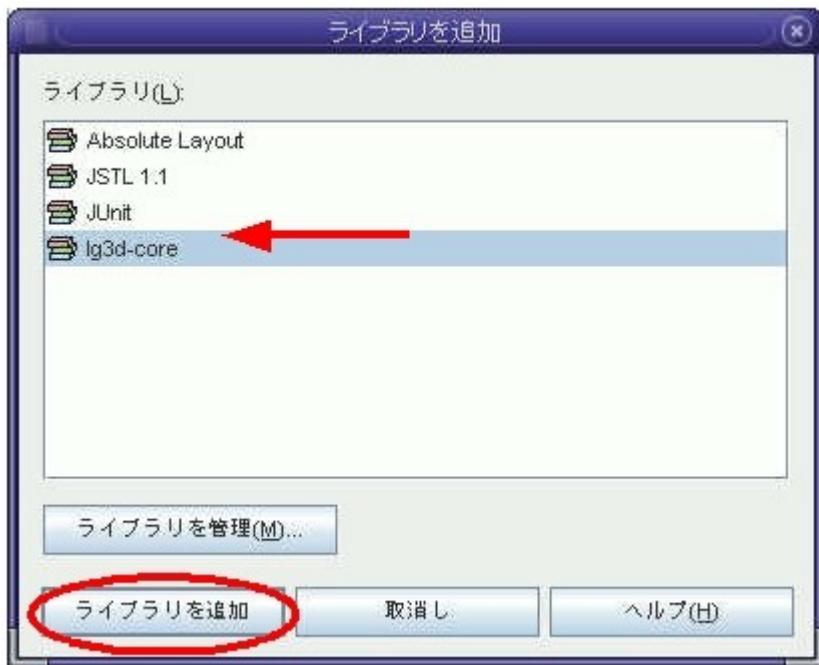
2. プロジェクトプロパティの設定用ウィンドウが表示されますので、
 　・ カテゴリ : **ライブラリ**
 　を選択します。



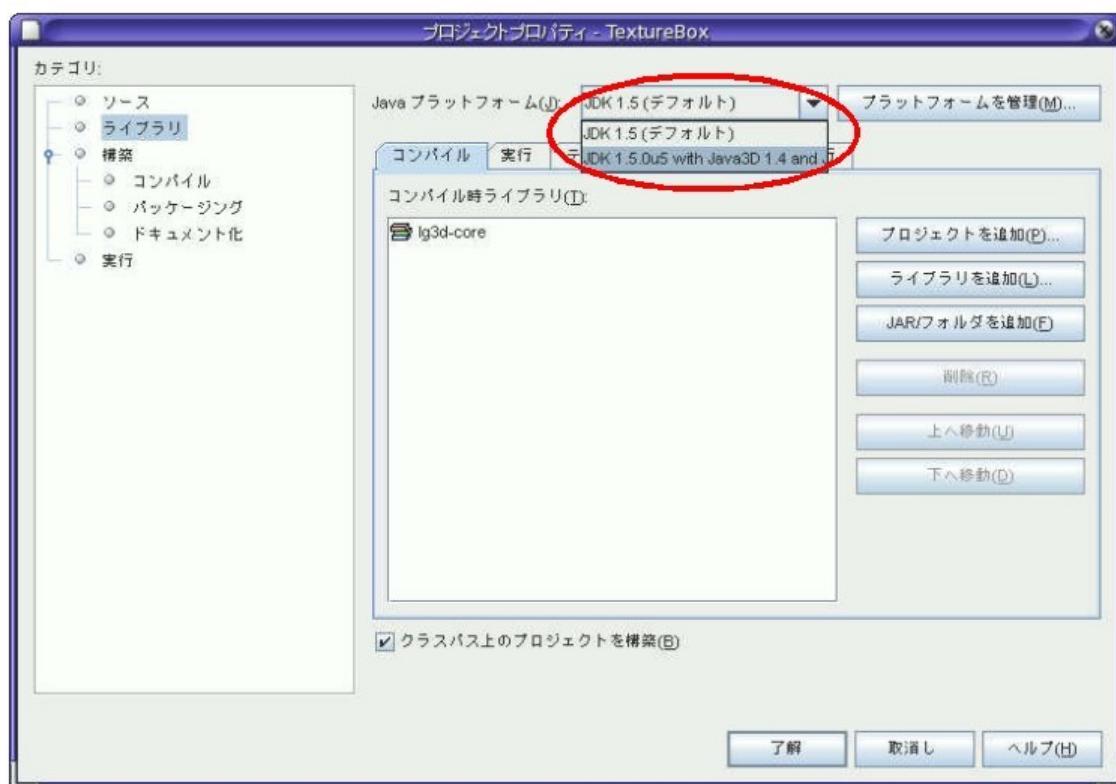
3. ウィンドウの右側の「コンパイル」が選択されていることを確認し、「ライブラリの追加」をクリックします。



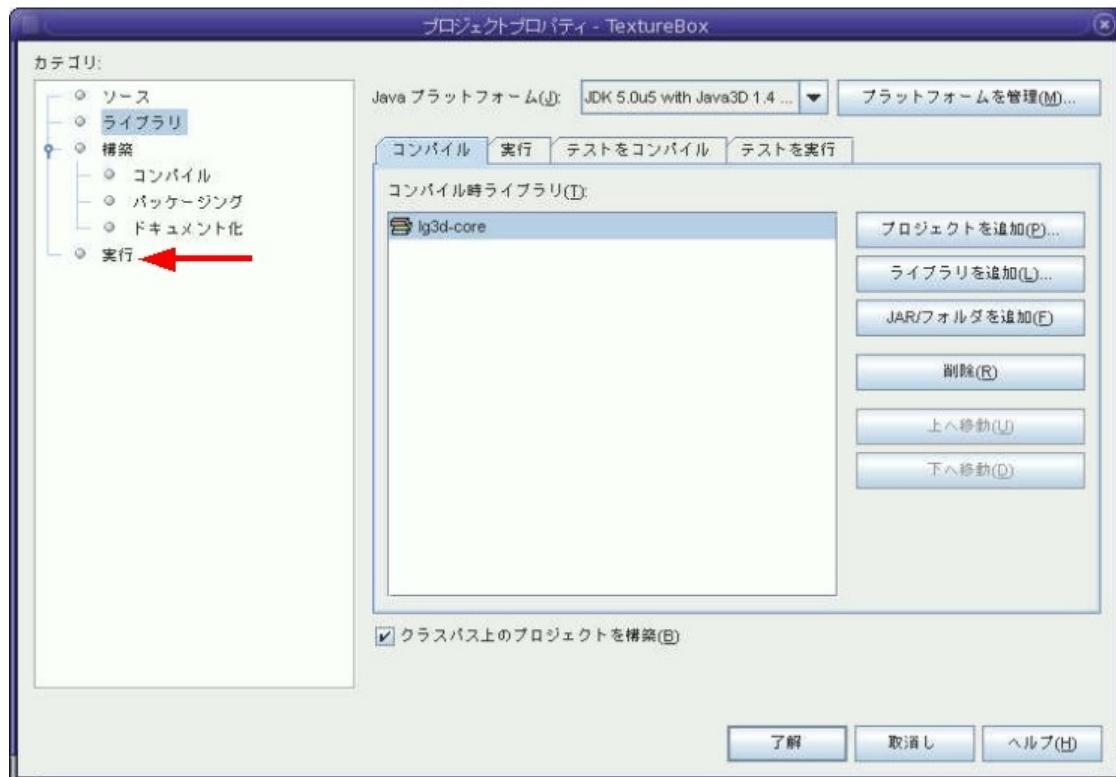
4. 「ライブラリの追加」という新しいウィンドウが表示されますので、「**Ig3d-core**」というライブラリを選択し、「**ライブラリの追加**」をクリックします。



5. プロジェクトプロパティの設定用ウィンドウの「コンパイル時ライブラリ」のリストに「**Ig3d-core**」が追加されていることを確認します。
確認後、上部の「Java プラットフォーム」の項目を
・ Java プラットフォーム : **JDK 5.0u5 with Java3D 1.4 and JAI 1.1.2_01**
に変更します。



6. 左側のパネルの「カテゴリ」のリストから「**実行**」を選択します。



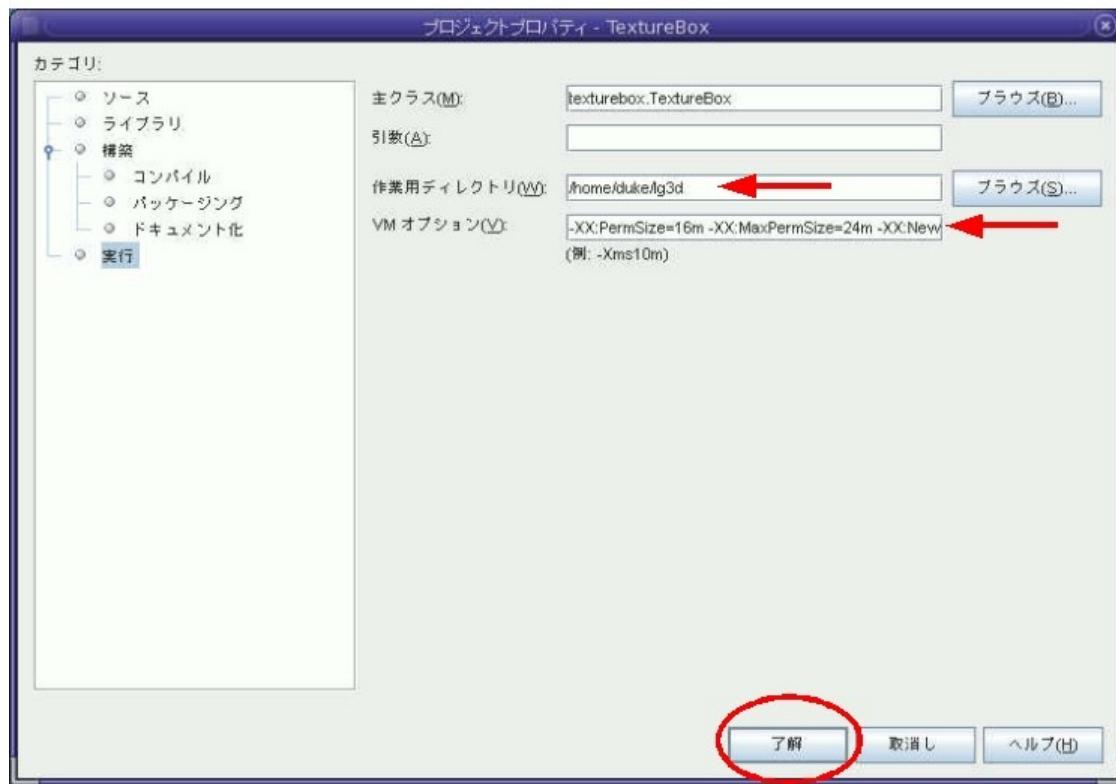
7. 右側のパネルの項目を次のようにします。

- ・ 作業用ディレクトリ : **/home/duke/lg3d** (LG3D 本体がインストールされているディレクトリ)
- ・ VM オプション :
-XX:PermSize=16m -XX:MaxPermSize=24m -XX:NewSize=128m -XX:MaxNewSize=128m -Xms256m -Xmx256m -XX:+UseConcMarkSweepGC -XX:+DisableExplicitGC -Dlg.etcdir=/home/duke/lg3d/etc/ -Dlg.configurl=file:///home/duke/lg3d/etc/lg3d/lgconfig_1p_nox.xml -Dlg.displayconfigurl=file:///home/duke/lg3d/etc/lg3d/displayconfig/j3d1x1

入力後、下側にある「**了解**」をクリックします。

これで、設定は終了です。

補足： 「VM オプション」の項目には、 LG3D を実行する際に利用するヒープメモリに関する設定と、 LG3D の実行に必要なコンフィギュレーションファイルの情報を入力しています。



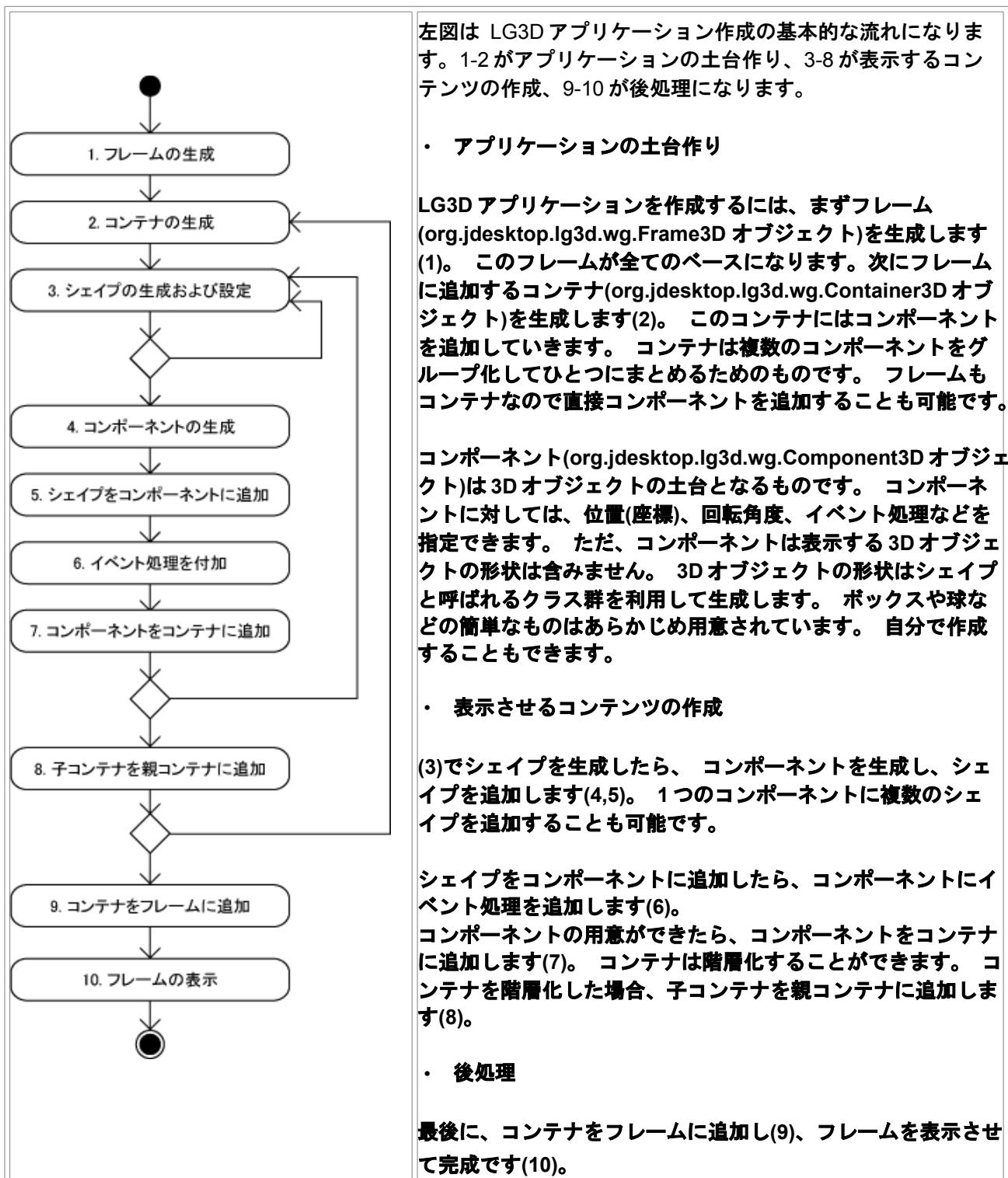
4.2 シンプルな 3D オブジェクトの作成

この章で行うこと：

NetBeans を用いて以下のことを行います。

- ・ 簡単な LG3D アプリケーションの作成
- ・ コンパイルおよび実行

LG3D アプリケーションの作成の流れ



シンプルな3Dオブジェクトの作成

この章では LG3D アプリケーション作成の手始めとしてボックスを作成します。

LG3D にあらかじめ用意してあるシェイプである org.jdesktop.lg3d.utils.shape.Box クラスを 利用します。

このボックスを作成する時、形状を示すために x, y ,z 座標 (すべて float) と アピアランスと呼ばれるものが必要になります。

作成されるボックスは (0,0,0)を中心とし、各辺の長さが 2x, 2y, 2z となります。 また、アピアランスは作成する3D アプリケーション(ここではボックス)の 表面の色や光の反射率を表すものです。

ここでは簡単なアピアランスを作成することができます

org.jdesktop.lg3d.utils.shape.SimpleAppearance クラスを用いて、 不透明な薄い青色を示すアピアランスを作成しています。

ボックスのシェイプはコンポーネントに追加し、コンポーネントをフレームに追加しています。
(今回はコンテナは作成せず、フレームに直接コンポーネントを追加しています。)

- 右側のパネルの **TextureBox.java** 下のリストのように書き換えます。

赤(太字)の部分が元のプログラムから変更・追加したところです。

```
/*
 * TextureBox.java
 *
 * Created on 2005/09/29, 15:42
 *
 * To change this template, choose Tools | Options and locate the template under
 * the Source Creation and Management node. Right-click the template and choose
 * Open. You can then make changes to the template in the Source Editor.
 */
package texturebox;

/**
 *
 * @author duke
 */

import javax.vecmath.Vector3f;
import org.jdesktop.lg3d.sg.Appearance;
import org.jdesktop.lg3d.utils.shape.Box;
import org.jdesktop.lg3d.utils.shape.SimpleAppearance;
import org.jdesktop.lg3d.wg.Component3D;
import org.jdesktop.lg3d.wg.Frame3D;

public class TextureBox {

    /** Creates a new instance of TextureBox */
    public TextureBox() {

        // フレームの生成
        Frame3D frame = new Frame3D();
```

```
// シェイプの生成および設定

// 色を指定した Appearance の設定
// 引数は R,G,B,アルファ(透過度) で、0-1 までの float 値
Appearance appearance
= new SimpleAppearance(0.6f, 0.6f, 1.0f, 1.0f);

// Box の生成
// 引数は、x、y、z 座標、アピアランス
// 座標の単位はメートル(float 値)
// 作成されるのは 横 0.20f, 縦 0.16f, 奥行き 0.12f のボックス
Box box = new Box(0.10f, 0.08f, 0.06f, appearance);

// コンポーネントの生成
Component3D component = new Component3D();

// シェイプをコンポーネントに追加
component.addChild(box);

// コンポーネントをコンテナに追加
// ここではコンテナを使用していないので、
// 直接フレームに追加
frame.addChild(component);

// フレームの表示
// フレームの大きさを設定
// 引数は、Java3D のクラス
// javax.vecmath.Vector3f(float x,float y,float z)
// x = 幅、y = 高さ、z = 奥行き (単位はメートル)
frame.setPreferredSize(new Vector3f(0.1f, 0.08f, 0.06f));

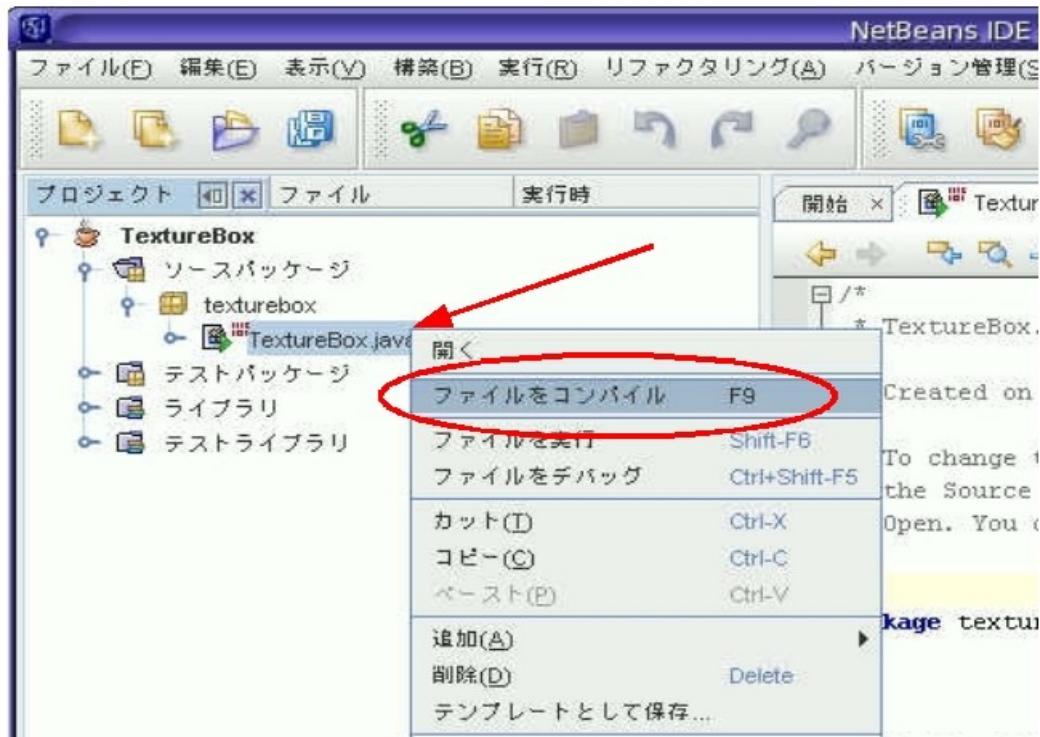
// フレームの表示
frame.changeEnabled(true);
frame.changeVisible(true);

}

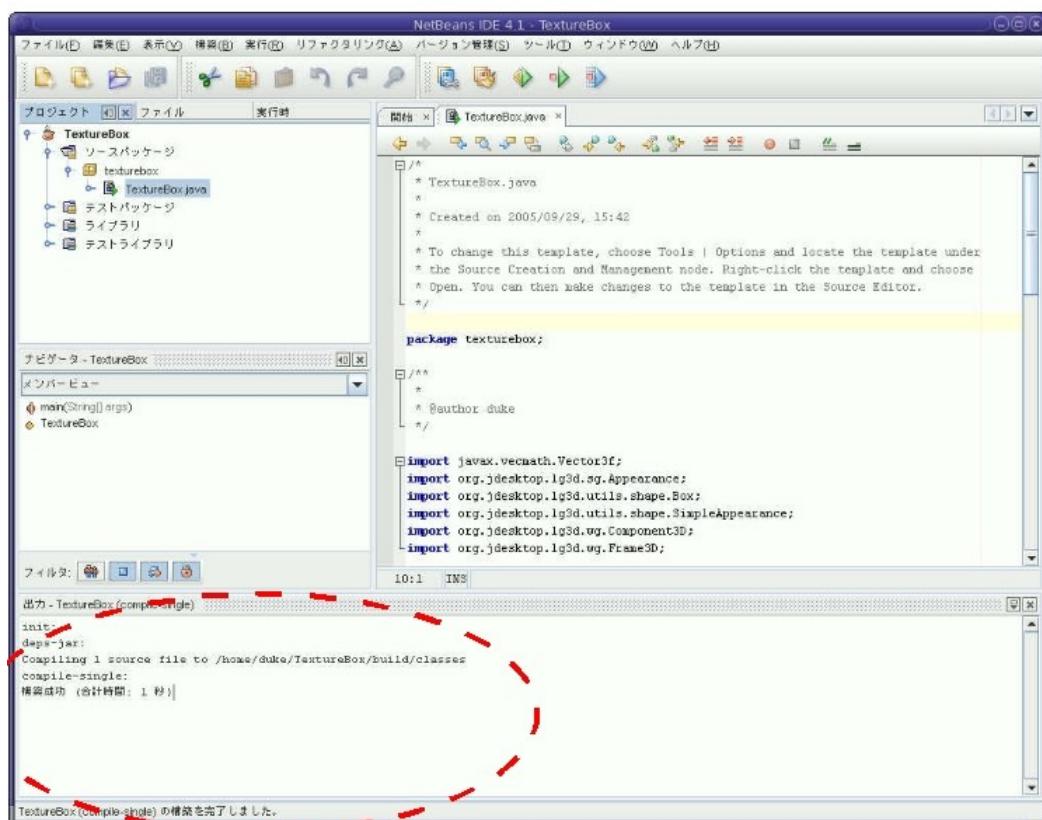
/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    // TODO code application logic here
    new TextureBox();
}

}
```

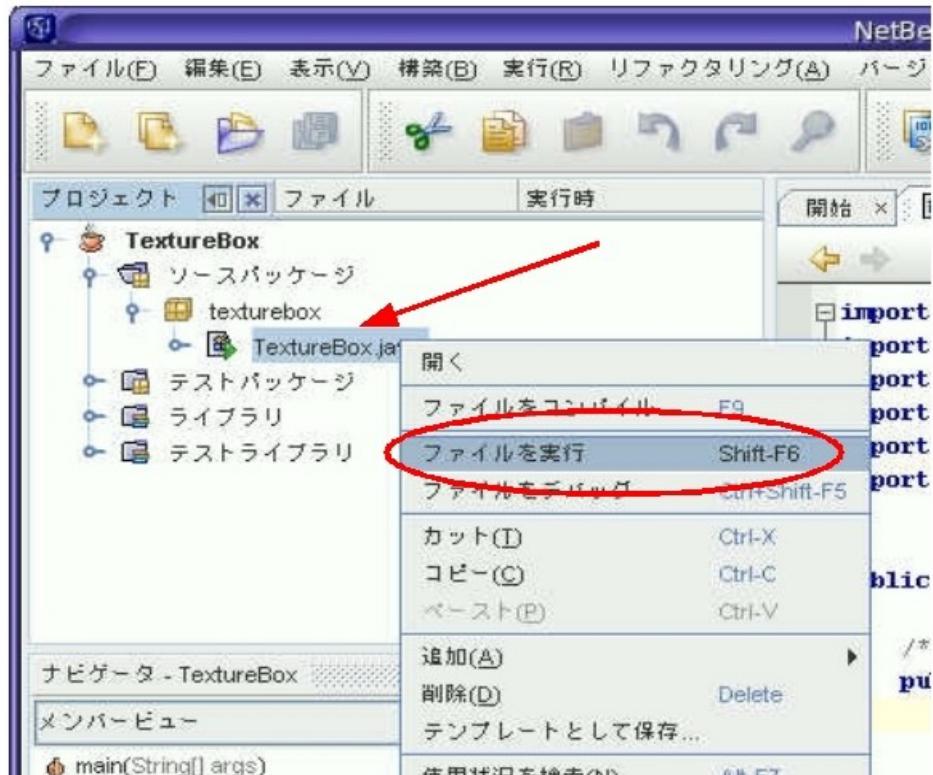
2. プログラムの記述が終わったら、左側のパネルの **TextBox.java** を選択し、マウスの右クリックをします。
下図のようなメニューが表示されるので 「**ファイルをコンパイル**」 を選択します。



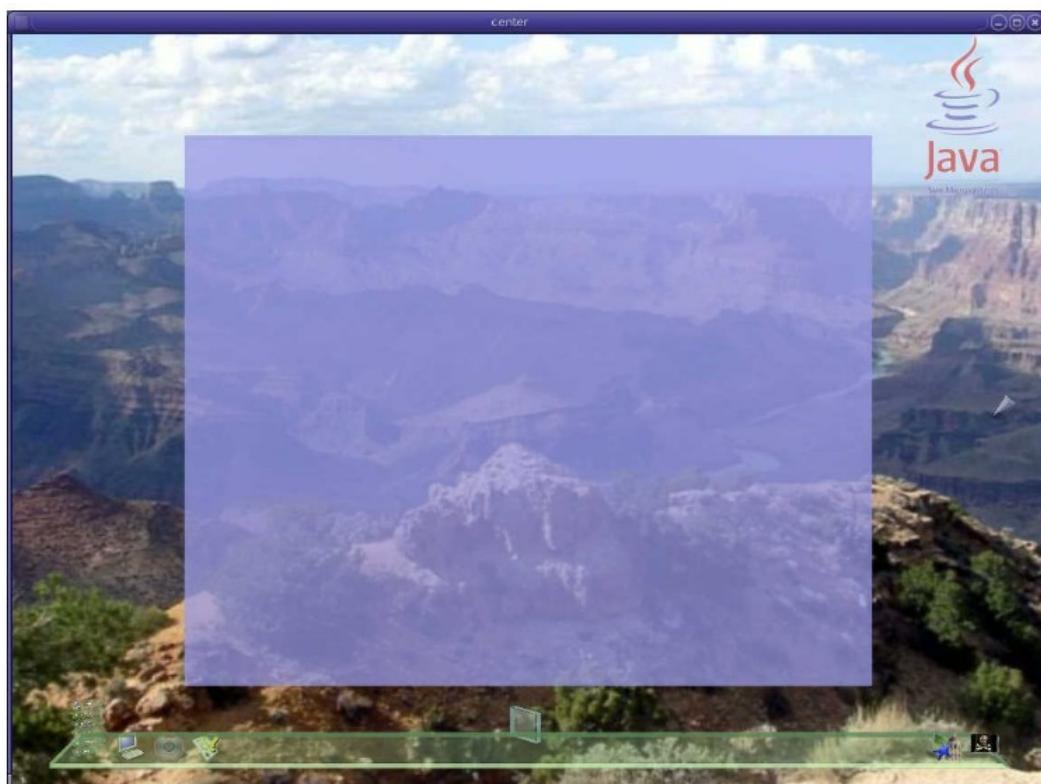
3. コンパイルが成功すると「構築成功」というメッセージが表示されます（下図の最下部を参照）。コンパイルに失敗した場合、この部分に エラーとその内容が表示されます。その場合は、プログラムを修正し、再度コンパイル作業を行ってください。



4. コンパイルが成功したら、カーソルを `TextBox.java` 上に移動し、マウスを右クリックします。
メニューが表示されるので「ファイルを実行」を選択します。



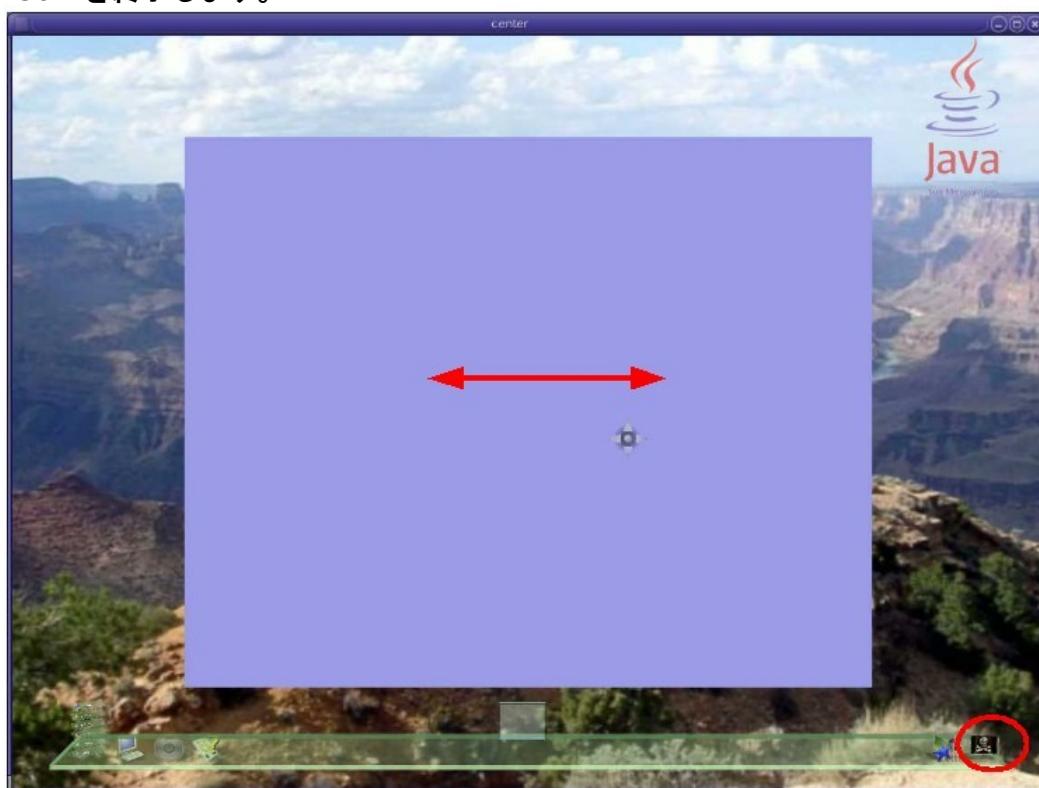
5. 下図のように別ウィンドウで LG3D が起動し、先ほど作成したボックスが表示されます。表示されない場合は、NetBeans の最下部のパネルにエラーが表示されていると思いますので、それを参考に設定の確認などを再度行ってください。



6. LG3D にボックスが表示されたら、ウィンドウの右上にある Java ロゴをドラッグ (Java ロゴをクリックしたままマウスを動かす) してみてください。下図のようにボックスを動かすことができます。ドラッグをやめるとボックスは元に戻ります。



7. マウスカーソルがボックス内に無いときはボックスは半透明ですが、マウスカーソルをボックス内に移動させると半透明ではなくなります。これは、LG3D の標準機能です。
このままボックスをドラッグして上下左右に動かしてみましょう。
LG3D 上で作成したボックスの動作を確認したら、右下の「**ドクロマーク**」をクリックし
LG3D を終了します。



4.3 テクスチャを使う

この章で行うこと：

ここまでで作成したボックスを拡張します。

以下のようなボックスを作成してみます。

- ・ 同じ画像をすべての面に貼り付けたボックス
- ・ すべての面に異なる画像を貼り付けたボックス

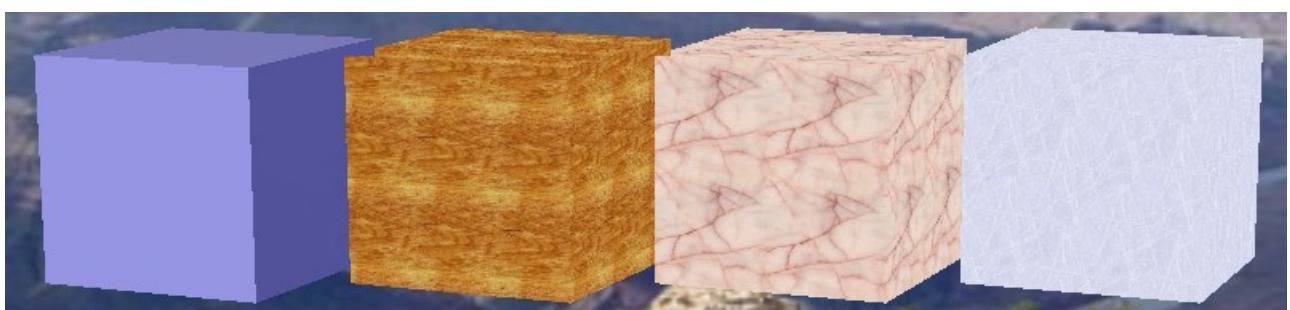
テクスチャとは？

3Dコンピュータグラフィックスにおいて、3Dオブジェクトの表面に貼り付ける画像のことをテクスチャと呼びます。

テクスチャを利用することにより3Dオブジェクトの質感などを簡単に表現することができます。

下図は前章で作成したボックスにテクスチャを貼ったした例です。

左から順に、テクスチャ無し、木のテクスチャ、ピンクの大理石のテクスチャ、氷のテクスチャです。



テクスチャを使う

では、実際にテクスチャを使ってみましょう。

テクスチャを利用するためには、まず画像を用意します。

ここではあらかじめ用意された画像(photo01.jpg - photo06.jpg)を利用します。

画像は作成するJARアプリケーションとは別に用意しても構いませんが、簡単化のために作成するJARアプリケーション内部に取り込むことにします。

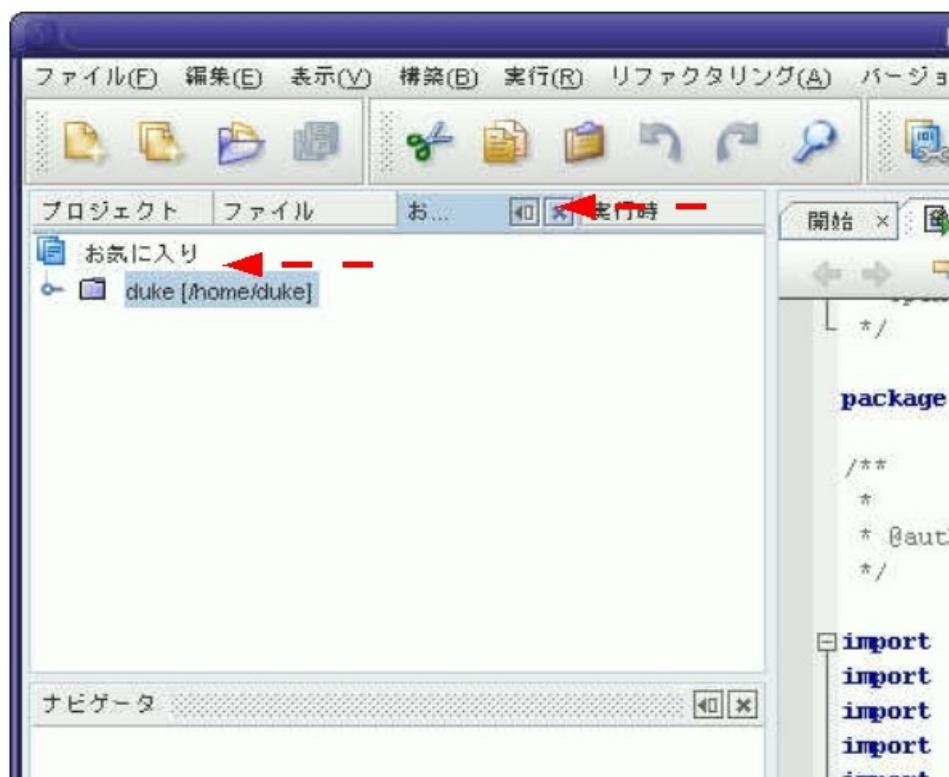
画像ファイルをプロジェクトに取り込む

画像ファイルはプロジェクトには含まれていないため、外部から取り込む必要があります。以下の手順で外部ディレクトリを読み込めるように設定し、画像ファイルをプロジェクトに取り込みます。

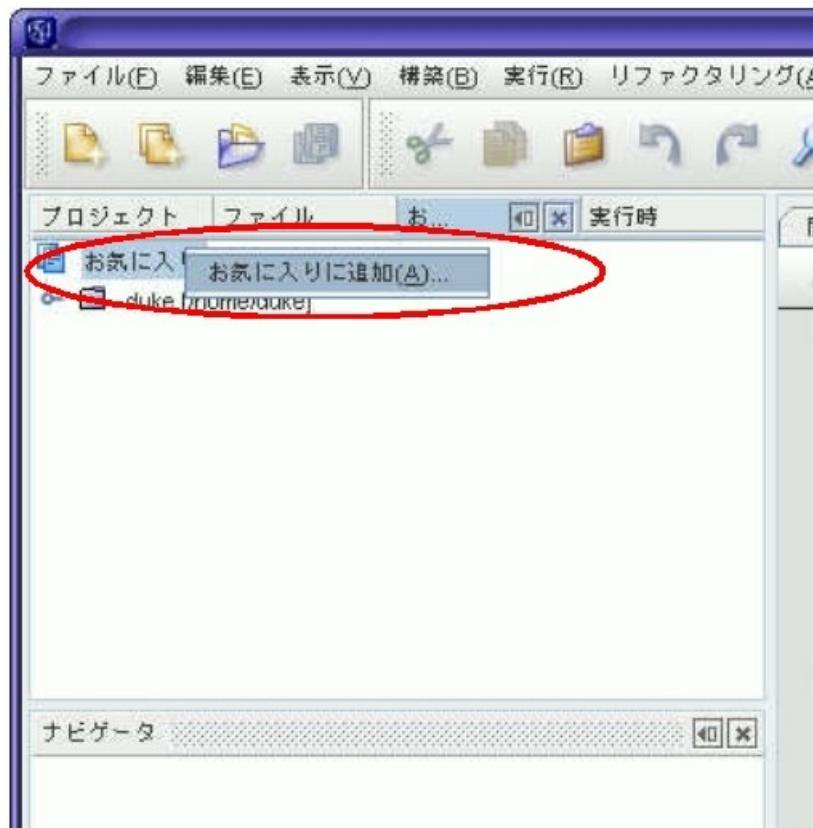
- メニューから「ウィンドウ」 → 「お気に入り」を選択します。



- IDE の左側のパネルに「お気に入り」が追加されていることを確認します。



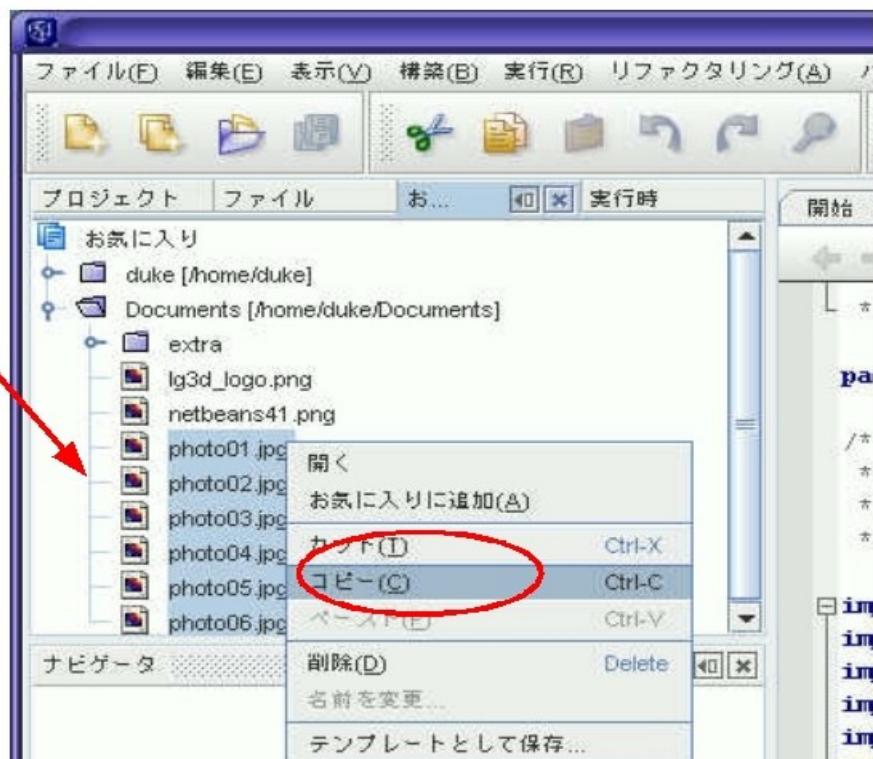
3. 「お気に入り」を右クリックすると メニューが表示されるで「お気に入りを追加」を選択します。



4. ファイル選択ダイアログが表示されるので 「Documents」を選択し、「追加」をクリックします。



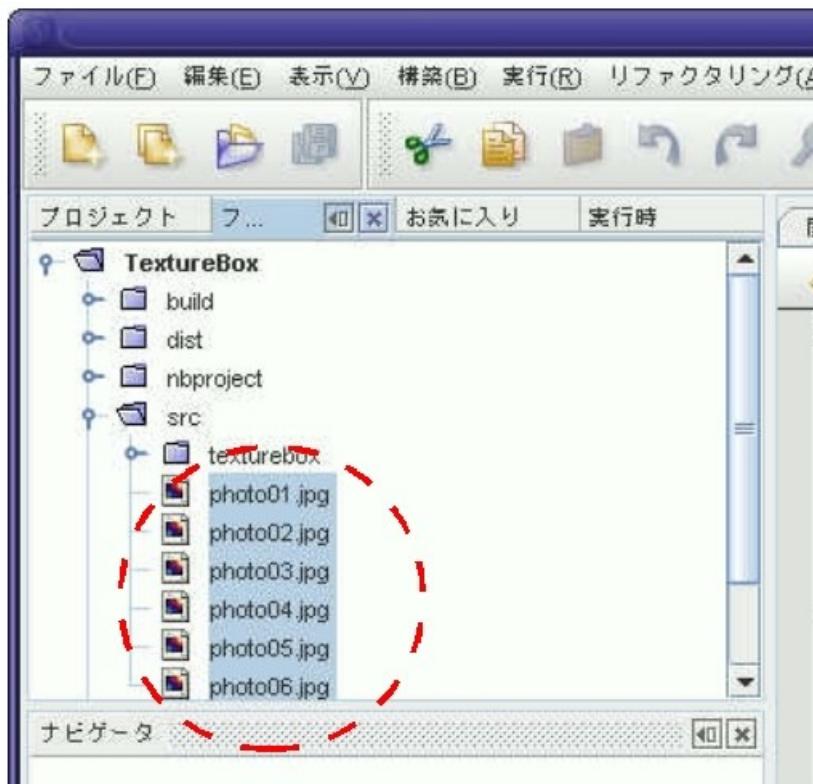
5. 左側のパネルに「Documents」 というフォルダ(ディレクトリ)が追加されます。
その中を開くと photo01.jpg - photo06.jpg という 6 つファイルが表示されますので、 これらを選択します(シフトキーを押しながらマウスで選択すると、複数選択が行えます)。選択後、右クリックするとメニューが表示されますので「コピー」を選択します。



6. 左上のタブから「ファイル」を選択します。
プロジェクト作成時に生成されたファイル、フォルダ(ディレクトリ)の一覧が表示されます。この中から「src」(ソースファイルのフォルダ(ディレクトリ))を選択し、 右クリックします。メニューが表示されますので「ペースト」を選択します。



7. 下図のように画像ファイルが追加されたことを確認します。これでプロジェクトへのファイルの追加は完成です。プロジェクトを構築する際に、これらのファイルが自動的に JAR アプリケーションに追加されます。



テクスチャを使う(その1)

前章で作成したボックスにテクスチャを貼ってみます。まずは 1つの画像をすべての面に表示させてみます。テクスチャを用いるためにプログラムに以下の変更を行います。

- アピアランスの生成部分

前章では 3D オブジェクトの表面の情報を指定するために アピアランスを使用することを学びました。テクスチャは 3D オブジェクトの表面に貼り付ける画像ですので、その指定にも同じアピアランスを用います。

具体的には、`org.jdesktop.lg3d.utils.shape.SimpleAppearance` クラスの引数の「R,G,B,アルファ(透過度)」の部分を 「**テクスチャとして利用する画像ファイル**」 に変更します。
(JPEG, GIF, PNG の 3つ画像ファイル形式に対応しています。)

- 例外処理

アピアランス生成時にテクスチャ用の画像ファイルがない場合は `FileNotFoundException`、読み込みに失敗した場合に `IOException` を発生します。そのため、例外処理を加える必要があります。

- org.jdesktop.lg3d.utils.shape.Box オブジェクトの生成部分

ボックスにテクスチャを貼るために `Box.GENERATE_TEXTURE_COORDS` というフラグを立てる必要があります。このフラグは `Box` オブジェクトがテクスチャを貼り付けるときに 必要となるテクスチャ座標の生成を行うというものです。

画像ファイルをプロジェクトに取り込んだため、 **ビルド(コンパイル)方法と実行方法が変わります** ので注意が必要です。

1. TextureBox.java を以下のように変更します。

変更箇所は **赤(太字)** で示します。

```
/*
 * TextureBox.java
 *
 * Created on 2005/09/29, 15:42
 *
 * To change this template, choose Tools | Options and locate the template under
 * the Source Creation and Management node. Right-click the template and choose
 * Open. You can then make changes to the template in the Source Editor.
 */
package texturebox;

/**
 *
 * @author duke
 */

import java.io.FileNotFoundException;
import java.io.IOException;
import javax.vecmath.Vector3f;
import org.jdesktop.j3d.lg3d.sg.Appearance;
import org.jdesktop.j3d.lg3d.utils.shape.Box;
import org.jdesktop.j3d.lg3d.utils.shape.SimpleAppearance;
import org.jdesktop.j3d.wg.Component3D;
import org.jdesktop.j3d.wg.Frame3D;

public class TextureBox {

    /** Creates a new instance of TextureBox */
    // SimpleAppearance オブジェクトを作成する際に、
    // 画像ファイルが見つからない場合に FileNotFoundException
    // 読み込みに失敗した場合 IOException
    // を発生させますので、それに対応するために Exception 处理を追加
    public TextureBox() throws FileNotFoundException, IOException {

        // フレームの生成
        Frame3D frame = new Frame3D();

        // シェイプの生成および設定
        // 画像を指定した Appearance の設定
        Appearance appearance
            = new SimpleAppearance("photo01.jpg");

        // Box の生成
        // 引数は、幅(x)、高さ(y)、奥行き(z)、フラグ、アピアランス
        // Box.GENERATE_TEXTURE_COORDS は
        // Box オブジェクトでテクスチャを利用するため必要なフラグです
        Box box = new Box(0.10f, 0.08f, 0.06f,
            Box.GENERATE_TEXTURE_COORDS, appearance);
}
```

```
// コンポーネントの生成
Component3D component = new Component3D();

// シエイプをコンポーネントに追加
component.addChild(box);

// コンポーネントをコンテナに追加
// ここではコンテナを使用していないので、
// 直接フレームに追加
frame.addChild(component);

// フレームの表示
// フレームの大きさを設定
frame.setPreferredSize(new Vector3f(0.1f, 0.08f, 0.06f));

// フレームの表示
frame.changeEnabled(true);
frame.changeVisible(true);

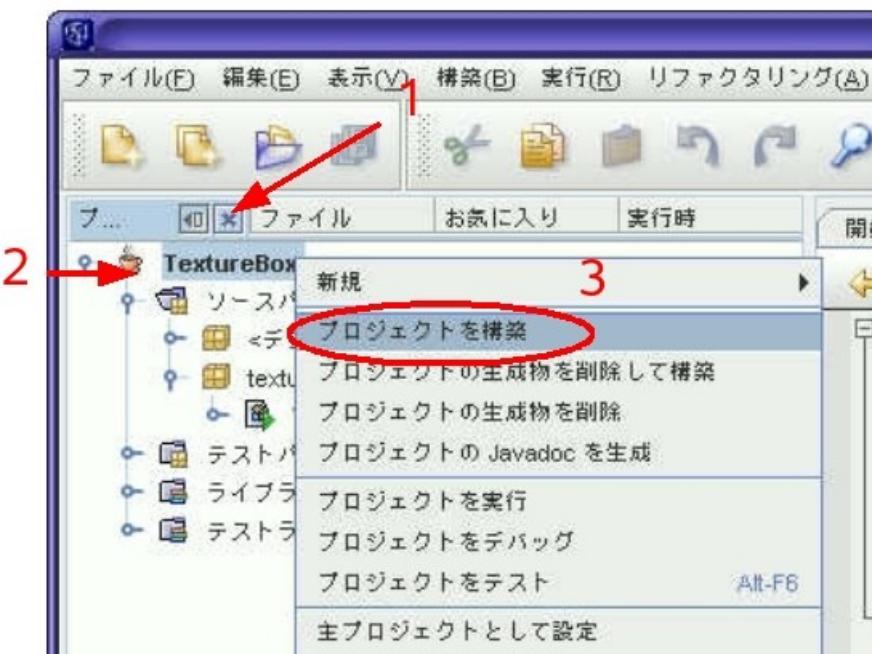
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    // TODO code application logic here

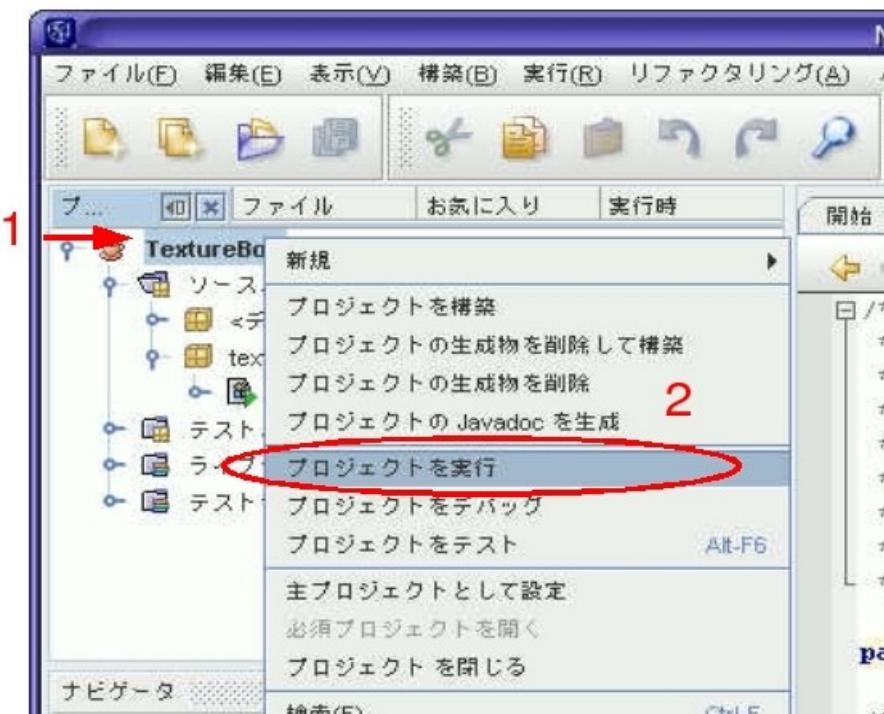
    // 例外処理を追加
    // - 画像ファイルが見つからなかった時
    // - 画像ファイルの読み込みに失敗した時
    try {
        new TextBox();
    } catch (FileNotFoundException ex) {
        System.err.println("画像ファイルが読み込めません。");
        ex.printStackTrace();
    } catch (IOException ex) {
        System.err.println("画像ファイルの読み込みに失敗しました。");
        ex.printStackTrace();
    }
}

}
```

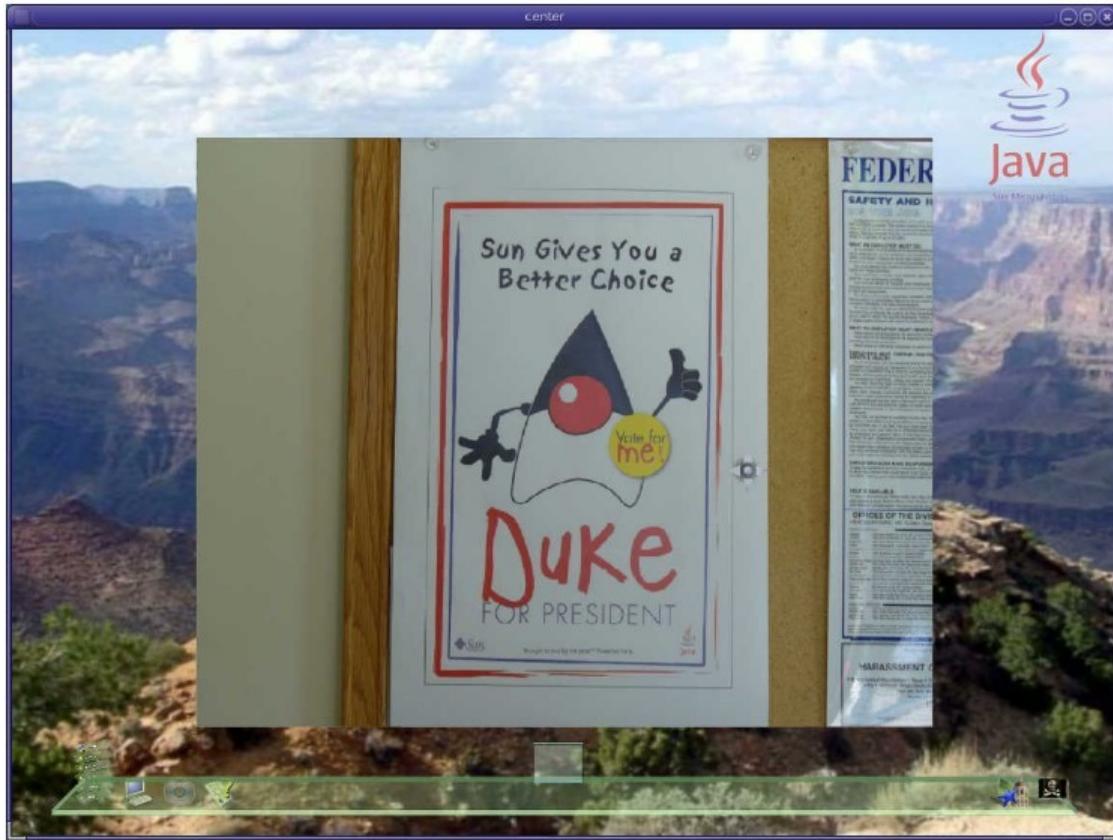
2. 左側のパネルから「**TextBox**」(プロジェクト名)を選択し、右クリックします。
メニューが表示されますので「**プロジェクトの構築**」を選択します。
エラー等が発生した場合は下部のフレームにエラーが表示されますので、エラーを修正し、再度プロジェクトの構築を行って下さい。



3. 「**TextBox**」(プロジェクト名)を選択し、右クリックします。
メニューが表示されますので「**プロジェクトを実行**」を選択します。



4. 実行画面は下図のようになります。



5. Java ロゴをドラッグしてボックスを回転させてみて下さい。全ての面に同じ画像が表示されていることがわかります。確認後、右下のドクロマークで LG3D を終了します。



テクスチャを使う(その2)

上記では1つの画像をすべての面に表示させる処理を行いましたが、次は6面すべてに違う画像を表示するようにプログラムを変更します。変更点は次の通りです。

- org.jdesktop.lg3d.utils.shape.Box オブジェクトの生成部分
 - アピアランスを null にする
各面に異なるテクスチャを貼るために、ここではアピアランスは指定しない
 - ボックスの各面に異なる画像を貼り付ける
 - Box オブジェクトから各面のシェイプ(org.jdesktop.lg3d.sg.Shape3D オブジェクト)を取得
 - Shape3D.setAppearance() メソッドを利用し、ボックスの各面にアピアランスを設定する
各面のアピアランスは SimpleAppearance クラスを利用し、別々のテクスチャを使ったアピアランスを生成する。

1. TextureBox.java を次のように書き換えます。

変更部分は赤(太字)の部分です。

```
/*
 * TextureBox.java
 *
 * Created on 2005/09/29, 15:42
 *
 * To change this template, choose Tools | Options and locate the template under
 * the Source Creation and Management node. Right-click the template and choose
 * Open. You can then make changes to the template in the Source Editor.
 */

package texturebox;

/**
 *
 * @author duke
 */

import java.io.FileNotFoundException;
import java.io.IOException;
import javax.vecmath.Vector3f;
import org.jdesktop.lg3d.sg.Appearance;
import org.jdesktop.lg3d.sg.Shape3D;
import org.jdesktop.lg3d.utils.shape.Box;
import org.jdesktop.lg3d.utils.shape.SimpleAppearance;
import org.jdesktop.lg3d.wg.Component3D;
import org.jdesktop.lg3d.wg.Frame3D;

public class TextureBox {

    /** Creates a new instance of TextureBox */
    public TextureBox() throws FileNotFoundException, IOException { /**

        // フレームの生成
        Frame3D frame = new Frame3D();
    }
}
```

```
// Box の生成
// 引数は、幅(x)、高さ(y)、奥行き(z)、フラグ、アピアランス
// Box.GENERATE_TEXTURE_COORDS は
// Box オブジェクトで テクスチャを利用するため必要なフラグです
Box box = new Box(0.10f, 0.08f, 0.06f,
                  Box.GENERATE_TEXTURE_COORDS, null);

// 各面にテクスチャを貼る
// Box オブジェクトから面の情報を取得するには
// org.jdesktop.lg3d.utils.shape.Box の
// static 変数(FRONT, RIGHT, LEFT, TOP, BOTTOM, BACK) を利用します。

// 正面
Shape3D shape = box.getShape(Box.FRONT);
shape.setAppearance(new SimpleAppearance("photo01.jpg"));

// 右側
shape = box.getShape(Box.RIGHT);
shape.setAppearance(new SimpleAppearance("photo02.jpg"));

// 左側
shape = box.getShape(Box.LEFT);
shape.setAppearance(new SimpleAppearance("photo03.jpg"));

// 上
shape = box.getShape(Box.TOP);
shape.setAppearance(new SimpleAppearance("photo04.jpg"));

// 下(底面)
shape = box.getShape(Box.BOTTOM);
shape.setAppearance(new SimpleAppearance("photo05.jpg"));

// 後
shape = box.getShape(Box.BACK);
shape.setAppearance(new SimpleAppearance("photo06.jpg"));

// コンポーネントの生成
Component3D component = new Component3D();

// シェイプをコンポーネントに追加
component.addChild(box);

// コンポーネントをコンテナに追加
// ここではコンテナを使用していないので、
// 直接フレームに追加
frame.addChild(component);

// フレームの表示
// フレームの大きさを設定
frame.setPreferredSize(new Vector3f(0.1f, 0.08f, 0.06f));

// フレームの表示
frame.changeEnabled(true);
```

```

        frame.setVisible(true);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        // 例外処理を追加
        // - 画像ファイルが見つからなかった時
        // - 画像ファイルの読み込みに失敗した時
        try {
            new TextureBox();
        } catch (FileNotFoundException ex) {
            System.err.println("画像ファイルが読み込めません。");
            ex.printStackTrace();
        } catch (IOException ex) {
            System.err.println("画像ファイルの読み込みに失敗しました。");
            ex.printStackTrace();
        }
    }
}

```

2. 「**プロジェクトの構築**」および「**プロジェクトの実行**」を行います。
LG3D の実行ウィンドウが表示されますので、Java ロゴをドラッグし、各面に違う画像が表示されているか確認します。



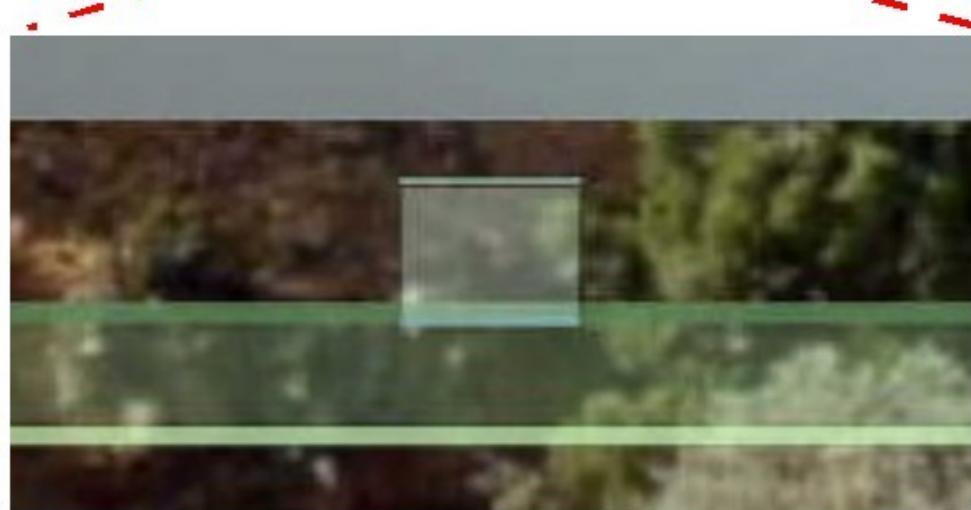
4.4 サムネイルの作成

この章で行うこと：

この章では、前章までで作成したボックスと同じ形をしたサムネイルの作成を行います。

LG3D を起動すると下部にタスクバーが表示されます。このタスクバーには「アプリケーションを起動するためのアイコン」と「現在起動しているアプリケーションの情報を示すサムネイル」が表示されます。

デフォルトのサムネイルは下図のような単純なものですが、LG3D ではこのサムネイルも自由に作成することができます。ここでは、作成したボックスと同じ形のサムネイルを作成します。



サムネイルの作成

org.jdesktop.lg3d.wg.Thumbnail クラスを利用してサムネイルを作成します
Thumbnail クラスは Component3D の派生クラスで、コンポーネントと同様に

- Thumbnail オブジェクトにコンポーネントを追加する

ことによりサムネイルを作成します。

ここでは、前章で作成したテクスチャを貼ったボックスと同じ形状のサムネイルを作成します。
なお、フレームにコンポーネントを追加する場合 Frame3D.addChild() を使用しましたが、サム
ネイルを追加する場合は Frame3D.setThumbnail() になります。

- TextureBox.java を変更し、テクスチャを貼る部分を別のメソッドにします。

変更箇所は赤(太字)で示します。

サムネイルの作成方法はコンポーネントと同様なので、テクスチャの生成および貼り付け
作業を コンポーネントとサムネイルの 2 度行うことになります。ここでは簡単化のため
に **setTextures(Box box)** というテクスチャ貼り付け用のメソッドを作成します。

補足:

LG3D では 1 つの 3D オブジェクトを複数のコンポーネント/コンテナに追加す
ることが出来ません。そのため、見栄えが同じオブジェクトが 2 つ必要な場
合でも、それぞれ別々に作成する必要があります。

```
/*
 * TextureBox.java
 *
 * Created on 2005/09/29, 15:42
 *
 * To change this template, choose Tools | Options and locate the template under
 * the Source Creation and Management node. Right-click the template and choose
 * Open. You can then make changes to the template in the Source Editor.
 */
package texturebox;

/**
 *
 * @author duke
 */

import java.io.FileNotFoundException;
import java.io.IOException;
import javax.vecmath.Vector3f;
import org.jdesktop.lg3d.sg.Appearance;
import org.jdesktop.lg3d.sg.Shape3D;
import org.jdesktop.lg3d.utils.shape.Box;
import org.jdesktop.lg3d.utils.shape.SimpleAppearance;
import org.jdesktop.lg3d.wg.Component3D;
import org.jdesktop.lg3d.wg.Frame3D;

public class TextureBox {

    /** Creates a new instance of TextureBox */
    public TextureBox() throws FileNotFoundException, IOException { /**

```

```

// フレームの生成
Frame3D frame = new Frame3D();

// Box の生成
Box box = new Box(0.10f, 0.08f, 0.06f,
    Box.GENERATE_TEXTURE_COORDS, null);

// 各面にテクスチャを貼る
setTextures(box);

// コンポーネントの生成
Component3D component = new Component3D();

// シェイプをコンポーネントに追加
component.addChild(box);

// コンポーネントをコンテナに追加
// ここではコンテナを使用していないので、
// 直接フレームに追加
frame.addChild(component);

// フレームの表示
// フレームの大きさを設定
frame.setPreferredSize(new Vector3f(0.1f, 0.08f, 0.06f));

// フレームの表示
frame.changeEnabled(true);
frame.changeVisible(true);

}

private void setTextures(Box box)
    throws FileNotFoundException, IOException {
// 各面にテクスチャを貼る
Shape3D shape = box.getShape(Box.FRONT);
shape.setAppearance(new SimpleAppearance("photo01.jpg"));

shape = box.getShape(Box.RIGHT);
shape.setAppearance(new SimpleAppearance("photo02.jpg"));

shape = box.getShape(Box.LEFT);
shape.setAppearance(new SimpleAppearance("photo03.jpg"));

shape = box.getShape(Box.TOP);
shape.setAppearance(new SimpleAppearance("photo04.jpg"));

shape = box.getShape(Box.BOTTOM);
shape.setAppearance(new SimpleAppearance("photo05.jpg"));

shape = box.getShape(Box.BACK);
shape.setAppearance(new SimpleAppearance("photo06.jpg"));
}

```

```

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    // TODO code application logic here
    try {
        new TextureBox();
    } catch (FileNotFoundException ex) {
        System.err.println("画像ファイルが読み込めません。");
        ex.printStackTrace();
    } catch (IOException ex) {
        System.err.println("画像ファイルの読み込みに失敗しました。");
        ex.printStackTrace();
    }
}
}

```

2. Texture.java を変更し、サムネイルを表示するようします。

サムネイルの作成方法は以下のようになります

- ・ シェイプを作成する
- ・ シェイプにテクスチャを貼る
- ・ シェイプを含むコンポーネントを作成する
- ・ コンポーネントをサムネイルに格納する
- ・ サムネイルをフレームに格納する

また、サムネイルを作成する際、サムネイルとシェイプのサイズはコンポーネント作成時と同じサイズにします。 サムネイルは自動的に縮小表示されます。

```

/*
 * TextureBox.java
 *
 * Created on 2005/09/29, 15:42
 *
 * To change this template, choose Tools | Options and locate the template under
 * the Source Creation and Management node. Right-click the template and choose
 * Open. You can then make changes to the template in the Source Editor.
 */

package texturebox;

/**
 *
 * @author duke
 */

import java.io.FileNotFoundException;
import java.io.IOException;
import javax.vecmath.Vector3f;
import org.jdesktop.j3d_sg.Appearance;
import org.jdesktop.j3d_sg.Shape3D;
import org.jdesktop.j3d_utils.shape.Box;

```

```
import org.jdesktop.lwjgl.util.shape.SimpleAppearance;
import org.jdesktop.lwjgl.Component3D;
import org.jdesktop.lwjgl.Frame3D;
import org.jdesktop.lwjgl.Thumbnail;

public class TextureBox {

    /** Creates a new instance of TextureBox */
    public TextureBox() throws FileNotFoundException, IOException { /**

        // フレームの生成
        Frame3D frame = new Frame3D();

        // Box の生成
        Box box = new Box(0.10f, 0.08f, 0.06f,
                           Box.GENERATE_TEXTURE_COORDS, null);

        // 各面にテクスチャを貼る
        setTextures(box);

        // コンポーネントの生成
        Component3D component = new Component3D();

        // シェイプをコンポーネントに追加
        component.addChild(box);

        // サムネイル用の Box(シェイプ) の生成
        // シェイプの内の引数はコンポーネント作成時と同じものにします。
        Box thumbBox = new Box(0.10f, 0.08f, 0.06f,
                               Box.GENERATE_TEXTURE_COORDS, null);

        // サムネイル用の Box(シェイプ)にテクスチャを貼る
        setTextures(thumbBox);

        // コンポーネントを生成し、Box(シェイプ)を格納する
        Component3D thumbComponent = new Component3D();
        thumbComponent.addChild(thumbBox);

        // サムネイルの生成
        Thumbnail thumbnail = new Thumbnail();

        // サムネイルにコンポーネントを格納
        thumbnail.addChild(thumbComponent);

        // サムネイルのサイズを設定
        // このサイズもコンポーネントと同じサイズにします
        thumbnail.setPreferredSize(new Vector3f(0.10f, 0.08f, 0.06f));

        // サムネイルの追加
        // サムネイルの追加は Frame3D.setThumbnail() を使用します
        frame.setThumbnail(thumbnail);
```

```

// コンポーネントをコンテナに追加
// ここではコンテナを使用していないので、
// 直接フレームに追加
frame.addChild(component);

// フレームの表示
// フレームの大きさを設定
frame.setPreferredSize(new Vector3f(0.1f, 0.08f, 0.06f));

// フレームの表示
frame.changeEnabled(true);
frame.changeVisible(true);

}

private void setTextures(Box box)
    throws FileNotFoundException, IOException {
    // 各面にテクスチャを貼る
    Shape3D shape = box.getShape(Box.FRONT);
    shape.setAppearance(new SimpleAppearance("photo01.jpg"));

    shape = box.getShape(Box.RIGHT);
    shape.setAppearance(new SimpleAppearance("photo02.jpg"));

    shape = box.getShape(Box.LEFT);
    shape.setAppearance(new SimpleAppearance("photo03.jpg"));

    shape = box.getShape(Box.TOP);
    shape.setAppearance(new SimpleAppearance("photo04.jpg"));

    shape = box.getShape(Box.BOTTOM);
    shape.setAppearance(new SimpleAppearance("photo05.jpg"));

    shape = box.getShape(Box.BACK);
    shape.setAppearance(new SimpleAppearance("photo06.jpg"));
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    // TODO code application logic here
    try {
        new TextureBox();
    } catch (FileNotFoundException ex) {
        System.err.println("画像ファイルが読み込めません。");
        ex.printStackTrace();
    } catch (IOException ex) {
        System.err.println("画像ファイルの読み込みに失敗しました。");
        ex.printStackTrace();
    }
}
}

```

3. プログラムの変更が終了したら、「**プロジェクトの構築**」および、「**プロジェクトの実行**」を行います。実行画面をキャプチャしたものが下図になります。

(マウスカーソルがアプリケーションの外にある場合)



(マウスカーソルがアプリケーション内にある場合)



4.5 マウスイベント・アクションの作成

この章で行うこと：

作成したボックスを拡張してマウスイベントおよびアクションを作成します。

以下の2種類を実装してみます。

1. マウスボタンを押している間、ボックスが回転する
 - LG3D 標準のアクションを利用します
2. マウスの左クリックでボックスを 90 度回転させ、右クリックでボックスを逆方向に 90 度回転させる
 - アクションを新規に作成します

マウスイベント・アクションの概要

ここまでを行ってきた内容は LG3D アプリケーションの表示の部分です。

この章ではイベント処理の実装を行います。

LG3D でイベント処理を行うためには イベントアダプタ と アクション を用意します。

イベントアダプタ	イベントアダプタはイベントが発生したときに呼び出されるクラスです。イベントアダプタはイベントから情報を取得し、イベントアダプタに登録されているアクションの performAction() メソッドを呼び出します。イベントアダプタは org.jdesktop.g3d.utils.eventadapter パッケージで定義されており EventAdapter インタフェースを実装しています。
アクション	アクションはイベントが発生した際に使う処理を定義するためのクラスです。 org.jdesktop.lg3d.utils.action.Action インタフェースを親とした派生インターフェース群が提供されているので、これらのインターフェースのいずれかを実装したクラスでイベント処理を定義します。

イベントアダプタとアクションを利用したイベント処理の作成方法は次のようにになります。

- シエイプを含んだコンポーネントを作成する
- イベント発生時に使う動作(アクション)を定義する
- アクションを含むイベントアダプタを作成する
- コンポーネントにイベントアダプタを登録する

LG3D では、基本的に簡単なアクションをいくつか提供していますが、自分で作成することもできます。

ここでは、まず標準で提供されているアクションを利用してマウスイベントの作成を行います。その後、オリジナルのアクションを作成してみます。

既存のアクションを利用したアニメーション

ここでは、作成したコンポーネント上でマウスのボタンを押している続けている間にゆっくりと1回転し、マウスのボタンをはなすと元に戻るというイベント処理を作成してみます。このイベント処理は次のイベントアダプタおよびアクションを用いて実装します。

イベントアダプタ	<p>org.jdesktop.lg3d.utils.eventadapter.MousePressedEventAdapter クラス</p> <p>マウスのボタンを押している間のイベント処理を行うためのイベントアダプタ このイベントアダプタは次に示す3種類のインターフェースのいずれかを実装した アクションを設定することができます。</p> <ul style="list-style-type: none">• org.jdesktop.lg3d.utils.action.ActionBoolean• org.jdesktop.lg3d.utils.action.ActionBooleanFloat2• org.jdesktop.lg3d.utils.action.ActionBooleanFloat3
アクション	<p>org.jdesktop.lg3d.utils.action.RotateActionBoolean クラス</p> <p>イベント発生時にコンポーネントを回転させるためのアクションです。 コンストラクタの引数として、回転させたいオブジェクト、回転角度(ラジアン)、回転にかかる時間(ミリ秒)を取ります。</p>

1. TextureBox.java を以下のように変更します。

変更箇所は赤(太字)で示します。

adapter と thumbAdapter という2つのイベントアダプタ(MousePressedEventAdapter オブジェクト)を作成します。 作成したイベントアダプタは両方とも component (ボックスのコンポーネントのオブジェクト)に登録します。

これにより、ボックス上でマウスのボタンを押し続けることにより、ボックスと同時にサムネイルも回転します。

ボックスとサムネイルは 10 秒間で 1 回転するようにしています。

イベント処理を追加する前に、ボックスのコンポーネントとサムネイルの setAnimation() メソッドを呼び出しています

(component.setAnimation(), thumbComponent.setAnimation())。 これはアニメーション時にどのような動きをするかを定義したものです。 プログラム中に出てくる設定は滑らかなアニメーションをするための設定です。 アニメーションをさせたいオブジェクトに対するお約束の処理と考えてください。

```
/*
 * TextureBox.java
 *
 * Created on 2005/09/29, 15:42
 *
 * To change this template, choose Tools | Options and locate the template under
 * the Source Creation and Management node. Right-click the template and choose
 * Open. You can then make changes to the template in the Source Editor.
```

```

*/
package texturebox;

/**
 *
 * @author duke
 */

import java.io.FileNotFoundException;
import java.io.IOException;
import javax.vecmath.Vector3f;
import org.jdesktop.lg3d.sg.Appearance;
import org.jdesktop.lg3d.sg.Shape3D;
import org.jdesktop.lg3d.utils.action.RotateActionBoolean;
import org.jdesktop.lg3d.utils.actionadapter.ToggleAdapter;
import org.jdesktop.lg3d.utils.c3danimation.NaturalMotionAnimation;
import org.jdesktop.lg3d.utils.eventadapter.MousePressedEventAdapter;
import org.jdesktop.lg3d.utils.shape.Box;
import org.jdesktop.lg3d.utils.shape.SimpleAppearance;
import org.jdesktop.lg3d.wg.Component3D;
import org.jdesktop.lg3d.wg.Frame3D;
import org.jdesktop.lg3d.wg.Thumbnail;

public class TextureBox {

    /** Creates a new instance of TextureBox */
    public TextureBox() throws FileNotFoundException, IOException { /**

        // フレームの生成
        Frame3D frame = new Frame3D();

        // Box の生成
        Box box = new Box(0.10f, 0.08f, 0.06f,
                           Box.GENERATE_TEXTURE_COORDS, null);

        // 各面にテクスチャを貼る
        setTextures(box);

        // コンポーネントの生成
        Component3D component = new Component3D();

        // シェイプをコンポーネントに追加
        component.addChild(box);

        // サムネイル用の Box の生成
        Box thumbBox = new Box(0.10f, 0.08f, 0.06f,
                               Box.GENERATE_TEXTURE_COORDS, null);
        setTextures(thumbBox);
        Component3D thumbComponent = new Component3D();
        thumbComponent.addChild(thumbBox);

        // サムネイルの生成
        Thumbnail thumbnail = new Thumbnail();
        thumbnail.addChild(thumbComponent);
        thumbnail.setPreferredSize(new Vector3f(0.10f, 0.08f, 0.06f));
    }
}

```

```
// 滑らかに動かすためのお約束
component.setAnimation(new NaturalMotionAnimation(1000));

// マウスボタンを押しているときのイベント処理を行うためのイベントアダプタ
MousePressedEventAdapter adapter =
    new MousePressedEventAdapter(
        new RotateActionBoolean(component,
            (float)(Math.PI * 2.0), 10000));

// コンポーネントにイベントアダプタを登録
component.addListener(adapter);

// コンポーネントの上でマウスボタンを押した場合に
// サムネイルも同じようにアニメーションをさせるためのイベントアダプタ
thumbComponent.setAnimation(
    new NaturalMotionAnimation(1000));
MousePressedEventAdapter thumbAdapter =
    new MousePressedEventAdapter(
        new RotateActionBoolean(thumbComponent,
            (float)(Math.PI * 2.0), 10000));

// イベントをコンポーネントに登録する
// コンポーネント(ボックス)上でマウスボタンを押している時に
// サムネイルも動くようにするためにコンポーネントに登録しています
// サムネイル(thumbComponent)にイベントアダプタを登録しないでください
component.addListener(thumbAdapter);

// コンポーネントにイベントリスナを追加した際に必要なお約束
component.setMouseEventPropagatable(true);

// サムネイルの追加
frame.setThumbnail(thumbnail);

// コンポーネントをコンテナに追加
// ここではコンテナを使用していないので、
// 直接フレームに追加
frame.addChild(component);

// フレームの表示
// フレームの大きさを設定
frame.setPreferredSize(new Vector3f(0.1f, 0.08f, 0.06f));

// フレームの表示
frame.changeEnabled(true);
frame.changeVisible(true);

}

private void setTextures(Box box)
    throws FileNotFoundException, IOException {
```

```
// 各面にテクスチャを貼る
Shape3D shape = box.getShape(Box.FRONT);
shape.setAppearance(new SimpleAppearance("photo01.jpg"));

shape = box.getShape(Box.RIGHT);
shape.setAppearance(new SimpleAppearance("photo02.jpg"));

shape = box.getShape(Box.LEFT);
shape.setAppearance(new SimpleAppearance("photo03.jpg"));

shape = box.getShape(Box.TOP);
shape.setAppearance(new SimpleAppearance("photo04.jpg"));

shape = box.getShape(Box.BOTTOM);
shape.setAppearance(new SimpleAppearance("photo05.jpg"));

shape = box.getShape(Box.BACK);
shape.setAppearance(new SimpleAppearance("photo06.jpg"));
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    // TODO code application logic here
    try {
        new TextureBox();
    } catch (FileNotFoundException ex) {
        System.err.println("画像ファイルが読み込めません。");
        ex.printStackTrace();
    } catch (IOException ex) {
        System.err.println("画像ファイルの読み込みに失敗しました。");
        ex.printStackTrace();
    }
}
}
```

2. プログラムの記述が終わったら、「**プロジェクトの構築**」と「**プロジェクトの実行**」を行います。実行後、ボックス上で**マウスのボタンを押し続ける**とゆっくりとボックスとサムネイルが回転します。また、マウスのボタンを離すと元に戻ります。



アクションの作成

次はマウスをクリックしたときに 90 度づつ回転するようにプログラムを変更します。

マウスをクリックした場合のイベントアダプタは

org.jdesktop.desktop.lg3d.utils.eventadapter.MouseClickedEventAdapter クラスです。

MouseClickedEventAdapter クラスはコンストラクタの引数として、次のインターフェースのいずれかを実装したアダプタが必要です。

- org.jdesktop.desktop.lg3d.utils.action.ActionNoArg
- org.jdesktop.desktop.lg3d.utils.action.ActionFloat2
- org.jdesktop.desktop.lg3d.utils.action.ActionFloat3

ここでは ActionNoArg インタフェースを実装したクラスを作成します。

補足:

アクションを作成するために利用するインターフェースの名前は イベント発生時に呼び出される performAction() メソッドの引数に対応しています。 (引数のうち、先頭の org.jdesktop.desktop.lg3d.event.LgEventSource を除いたもの)

上記のインターフェースの performAction メソッドは次のようにになっています。

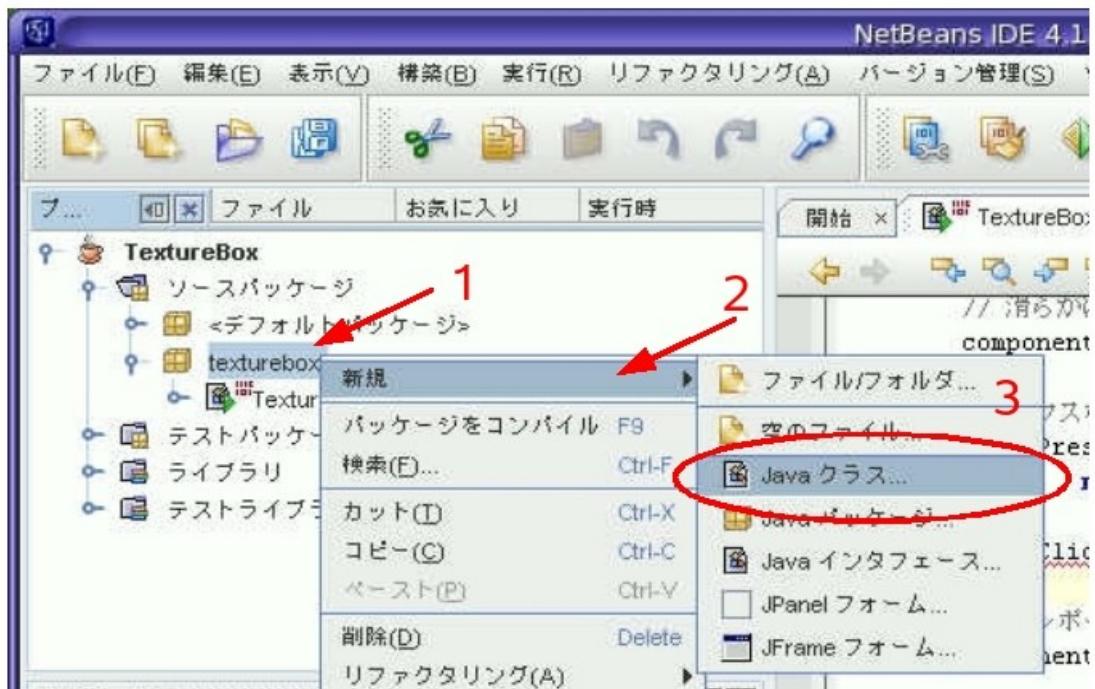
- ActionNoArg.performAction(LgEventSource source)
- ActionNoArg.performAction(LgEventSource source, float x, float y)
- ActionNoArg.performAction(LgEventSource source, float x, float y, float z)

performAction()メソッド実行時に渡される LgEventSource オブジェクトはイベントの発生元です。

1. アクションを作成するために新しいクラスを作成します。

左側のパネルが「プロジェクト」となっていることを確認し、 TextBox プロジェクト内の「ソースパッケージ」→「texturebox」を選択します。

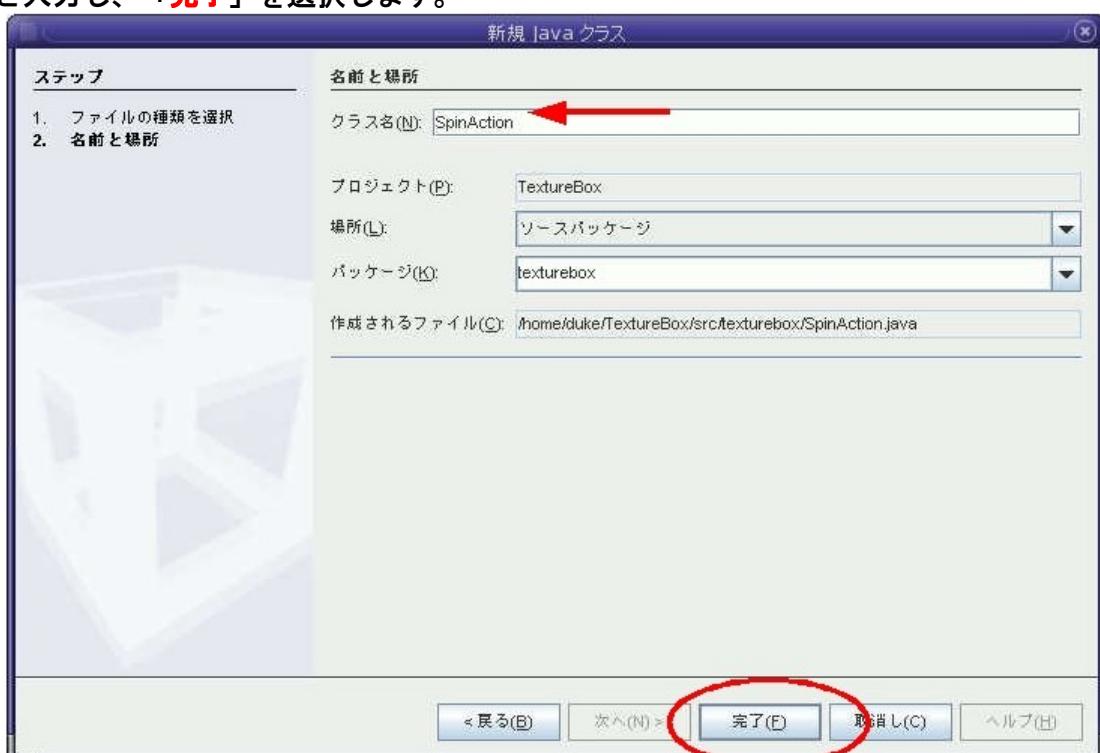
選択後、右クリックでメニューが表示されるので「新規」→「Java クラス」を選択します。



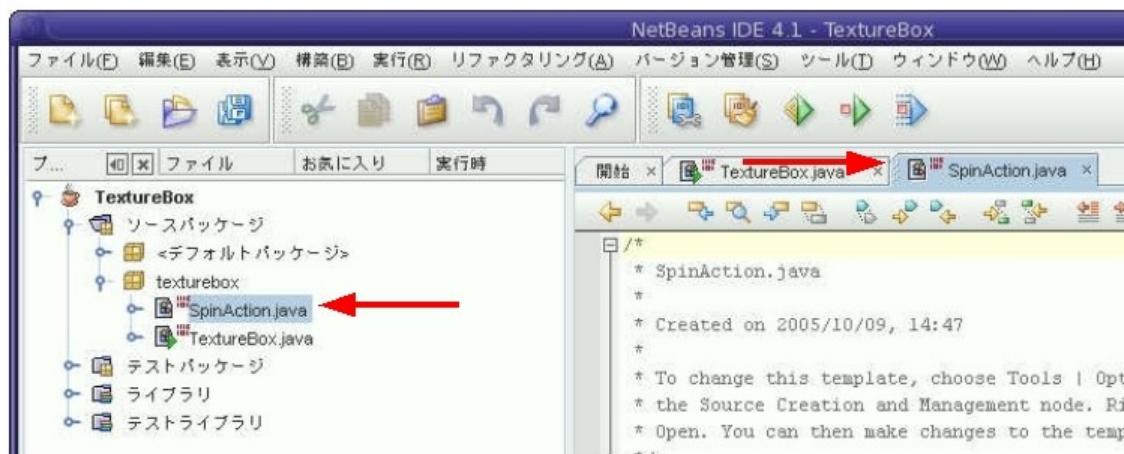
2. クラス情報を入力するためのダイアログが表示されます。

- ・ クラス名 : SpinAction

と入力し、「完了」を選択します。



3. 左側のパネルに「**SpinAction**」が追加されます。
 また、右側のソースコードエディタに「**SpinAction**」が表示されます。



4. SpinAction.java を以下のように変更します。

org.jdesktop.lg3d.utils.action.Action の派生インターフェースは イベント発生時に呼び出される performAction メソッドが必要とする変数が応じたものが用意されています。
 ここでは引数と特に必要としない org.jdesktop.lg3d.utils.action.ActionNoArg インタフェースを実装しています。

SpinAction クラスのメソッドはコンストラクタと performAction() の 2 つになります。

コンストラクタでは、

- 回転させたいコンポーネント
- 回転する角度
- 回転にかかる時間

を引数にしています。

performAction() はイベント発生時に実際にを行うアクションを実装するためのメソッドです。

ここではコンポーネントを 90 度回転させています。

引数の org.jdesktop.lg3d.event.LgEventSource オブジェクトはイベント発生元で、イベント発生時にイベントアダプタからアクションに渡されます。

```
/*
 * SpinAction.java
 *
 * Created on 2005/10/09, 14:47
 *
 * To change this template, choose Tools | Options and locate the template under
 * the Source Creation and Management node. Right-click the template and choose
 * Open. You can then make changes to the template in the Source Editor.
 */
package texturebox;

import org.jdesktop.lg3d.utils.action.ActionNoArg;
import org.jdesktop.lg3d.wg.Component3D;
import org.jdesktop.lg3d.event.LgEventSource;
```

```
/*
 *
 * @author duke
 */

public class SpinAction implements ActionNoArg{

    private Component3D target;
    private float diff;
    private int duration;
    private float angle;

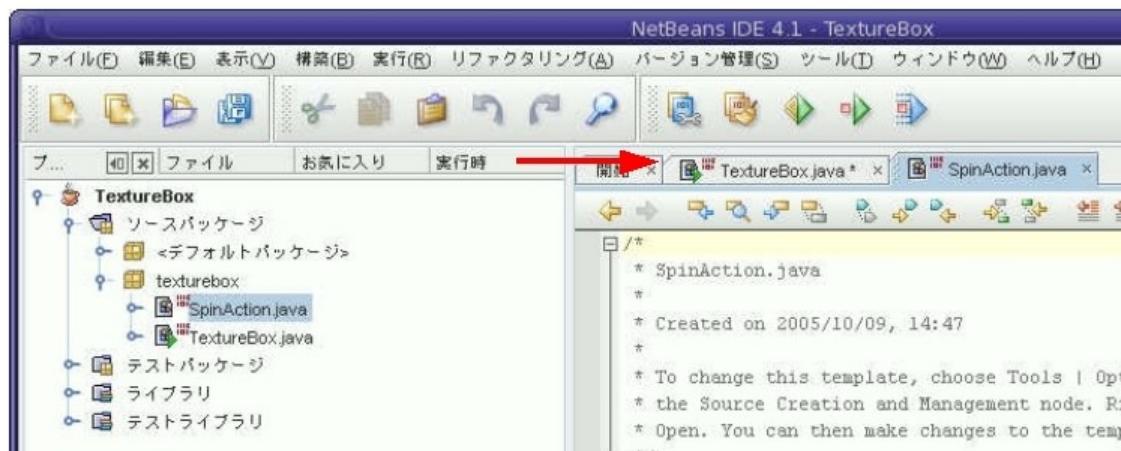
    // コンストラクタ
    // target 回転させたいコンポーネント
    // diff 回転させる角度 (ラジアン)
    // duration 回転に要する時間 (ミリ秒)
    public SpinAction(Component3D target, float diff, int duration) {
        if (target == null) {
            throw new IllegalArgumentException(
                "targetオブジェクトが指定されていません");
        }

        this.target = target;
        this.diff = diff;
        this.duration = duration;
    }

    // イベント処理をおこなうためのメソッド
    public void performAction(LgEventSource source) {
        // target.getFinalRotationAngle() は現在の回転角度(ラジアン)を
        // 取得するためのメソッドです。
        // このメソッドに diff(回転させる角度)を足して、
        // 新しい回転させる角度を定義します。
        angle = target.getFinalRotationAngle() + diff;
        // angle ... 最終的な回転角度
        // duration ... 回転に要する時間
        // です。実際に回転する角度は diff
        // (最終的な回転角度 angle - 現在の回転角度 getFinalRotationAngle() )
        // になります
        target.changeRotationAngle(angle, duration);
    }

}
```

5. 右側のソースコードエディタから「**TextureBox**」を選択し、 **TextureBox.java** の編集画面に切り替えます。



6. **TextureBox.java** を以下のように変更します。

イベントアダプタとして `org.jdesktop.lg3d.utils.eventadapter.MouseClickedEventAdapter` クラスを利用しています。

イベントアダプタ生成時のコンストラクタの第1引数はマウスのボタンを定義しています。

マウスのボタンは `org.jdesktop.lg3d.wg.event.MouseEvent3D` クラス内の `Enum` 変数 `ButtonId` に定義されており、

- `ButtonId.BUTTON1` = 左ボタン
- `ButtonId.BUTTON2` = 中ボタン
- `ButtonId.BUTTON3` = 右ボタン

を示しています。

ここでは、マウスの左クリックと右クリックの際のイベント処理の作成をしており、 左クリックで 90 度回転、右クリックで逆方向に 90 度回転します。

また、登録するアクションは先ほど作成した `SpinAction` クラスを利用しています。

```
/*
 * TextureBox.java
 *
 * * Created on 2005/09/29, 15:42
 *
 * * To change this template, choose Tools | Options and locate the template under
 * * the Source Creation and Management node. Right-click the template and choose
 * * Open. You can then make changes to the template in the Source Editor.
 */
package texturebox;

/**
 *
 * @author duke
 */

import java.io.FileNotFoundException;
```

```
import java.io.IOException;
import javax.vecmath.Vector3f;
import org.jdesktop.lwjgl.g3d.sg.Appearance;
import org.jdesktop.lwjgl.g3d.sg.Shape3D;
import org.jdesktop.lwjgl.g3d.utils.c3danimation.NaturalMotionAnimation;

import org.jdesktop.lwjgl.g3d.utils.eventadapter.MouseEventHandlerAdapter;
import org.jdesktop.lwjgl.g3d.wg.event.MouseEvent3D.ButtonId;

import org.jdesktop.lwjgl.g3d.utils.shape.Box;
import org.jdesktop.lwjgl.g3d.utils.shape.SimpleAppearance;
import org.jdesktop.lwjgl.g3d.wg.Component3D;
import org.jdesktop.lwjgl.g3d.wg.Frame3D;
import org.jdesktop.lwjgl.g3d.wg.Thumbnail;

public class TextureBox {

    /** Creates a new instance of TextureBox */
    public TextureBox() throws FileNotFoundException, IOException { /**

        // フレームの生成
        Frame3D frame = new Frame3D();

        // Box の生成
        Box box = new Box(0.10f, 0.08f, 0.06f,
                           Box.GENERATE_TEXTURE_COORDS, null);

        // 各面にテクスチャを貼る
        setTextures(box);

        // コンポーネントの生成
        Component3D component = new Component3D();

        // シェイプをコンポーネントに追加
        component.addChild(box);

        // サムネイル用の Box の生成
        Box thumbBox = new Box(0.10f, 0.08f, 0.06f,
                               Box.GENERATE_TEXTURE_COORDS, null);
        setTextures(thumbBox);
        Component3D thumbComponent = new Component3D();
        thumbComponent.addChild(thumbBox);

        // サムネイルの生成
        Thumbnail thumbnail = new Thumbnail();
        thumbnail.addChild(thumbComponent);
        thumbnail.setPreferredSize(new Vector3f(0.10f, 0.08f, 0.06f));

        // コンポーネントのイベント処理
        // 滑らかに動かすためのお約束
        component.setAnimation(new NaturalMotionAnimation(1000));

        // 左クリックで回転
        // 回転角度は 90 度
        MouseClickedEventAdapter leftClickedAdapter =
```

```
new MouseClickedEventAdapter(ButtonId.BUTTON1,
    new SpinAction(component, (float)(Math.PI / 2.0), 1000));

// 右クリックで逆回転
// 回転角度は -90 度
MouseClickedEventAdapter rightClickedAdapter =
    new MouseClickedEventAdapter(ButtonId.BUTTON3,
        new SpinAction(component, - (float)(Math.PI / 2.0), 1000));

// コンポーネントにイベントアダプタを登録
component.addListener(leftClickedAdapter);
component.addListener(rightClickedAdapter);

// イベント発生時にサムネイルも同じアクションを行うようにするため、
// サムネイルにも同じアクションを設定する
thumbComponent.setAnimation(
    new NaturalMotionAnimation(1000));
// 左クリックで回転
MouseClickedEventAdapter thumbLeftClickedAdapter =
    new MouseClickedEventAdapter(ButtonId.BUTTON1,
        new SpinAction(thumbComponent, (float)(Math.PI / 2.0),
        1000));

// 右クリックで逆回転
MouseClickedEventAdapter thumbRightClickedAdapter =
    new MouseClickedEventAdapter(ButtonId.BUTTON3,
        new SpinAction(thumbComponent, - (float)(Math.PI / 2.0),
        1000));

// コンポーネントがクリックされた時にサムネイルも回転させるために
// イベントアダプタをコンポーネントに登録します
component.addListener(thumbLeftClickedAdapter);
component.addListener(thumbRightClickedAdapter);

// コンポーネントにイベントリスナを追加した際に必要なお約束
component.setMouseEventPropagatable(true);

// サムネイルの追加
frame.setThumbnail(thumbnail);

// コンポーネントをコンテナに追加
// ここではコンテナを使用していないので、
// 直接フレームに追加
frame.addChild(component);

// フレームの表示
// フレームの大きさを設定
frame.setPreferredSize(new Vector3f(0.1f, 0.08f, 0.06f));

// フレームの表示
frame.changeEnabled(true);
frame.changeVisible(true);
```

```
}

private void setTextures(Box box)
    throws FileNotFoundException, IOException {
    // 各面にテクスチャを貼る
    Shape3D shape = box.getShape(Box.FRONT);
    shape.setAppearance(new SimpleAppearance("photo01.jpg"));

    shape = box.getShape(Box.RIGHT);
    shape.setAppearance(new SimpleAppearance("photo02.jpg"));

    shape = box.getShape(Box.LEFT);
    shape.setAppearance(new SimpleAppearance("photo03.jpg"));

    shape = box.getShape(Box.TOP);
    shape.setAppearance(new SimpleAppearance("photo04.jpg"));

    shape = box.getShape(Box.BOTTOM);
    shape.setAppearance(new SimpleAppearance("photo05.jpg"));

    shape = box.getShape(Box.BACK);
    shape.setAppearance(new SimpleAppearance("photo06.jpg"));
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    // TODO code application logic here
    try {
        new TextureBox();
    } catch (FileNotFoundException ex) {
        System.err.println("画像ファイルが読み込めません。");
        ex.printStackTrace();
    } catch (IOException ex) {
        System.err.println("画像ファイルの読み込みに失敗しました。");
        ex.printStackTrace();
    }
}

}
```

7. プログラムの変更が終了したら「**プロジェクトの構築**」および「**プロジェクトの実行**」を行います。

実行画面は下図のようになります。

ボックスをクリックまたは右クリックしてください。



4.6 JAR ベースアプリケーションの作成

前章までで LG3D の基本的な機能を利用したアプリケーションの作成を行いました。

ここでは、作成したアプリケーションを配布形式である JAR ファイルの作成を行います。

実は前章までで行ってきた「**プロジェクトの構築**」を実行すれば JAR ファイルを作成することができます。

ただ、今の状態では LG3D のタスクバーに表示するために必要な情報が定義されていないため、LG3D のメニューには表示されません。

メニューに表示させるようにするためには MANIFEST.MF ファイルを記述する必要があります。MANIFEST.MF ファイルは JAR ファイルの内容(バージョン、メインクラスなど)について記述するためのファイルです。

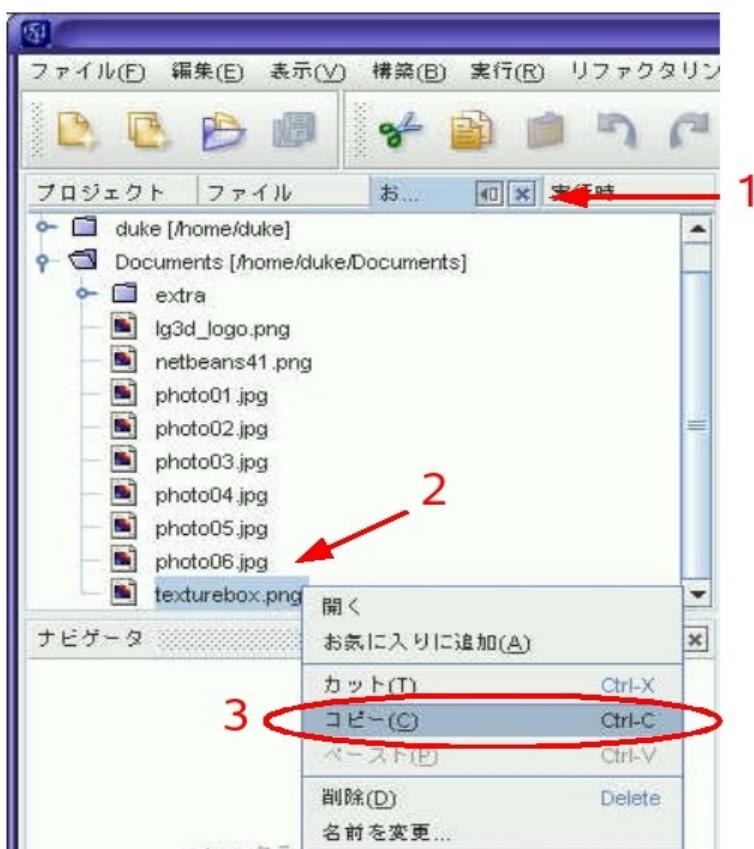
JAR ベースアプリケーションを作成する場合、MANIFEST.MF ファイルに

- Implementation-Name : アプリケーション名
- Main-Class : 実行用の main() メソッドが定義してあるクラス
- Icon-Filename : タスクバーに表示するアイコンファイル

を記述しておく必要があります。

JAR ベースアプリケーションの作成

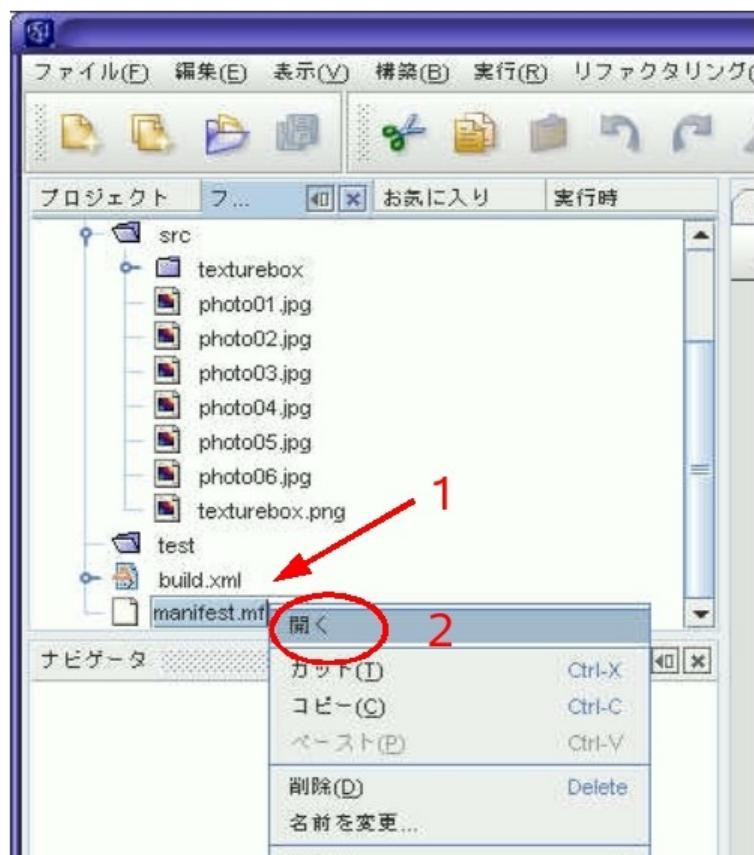
1. まず、プロジェクト内に LG3D のタスクバーに表示させるアイコンファイルを取り込みます。左側のパネルのタブから「**お気に入り**」を選択します。
「Documents」の下にある 「texturebox.jpg」 を選択します。
右クリックを押すとメニューが出るので 「**コピー**」 を選択します。



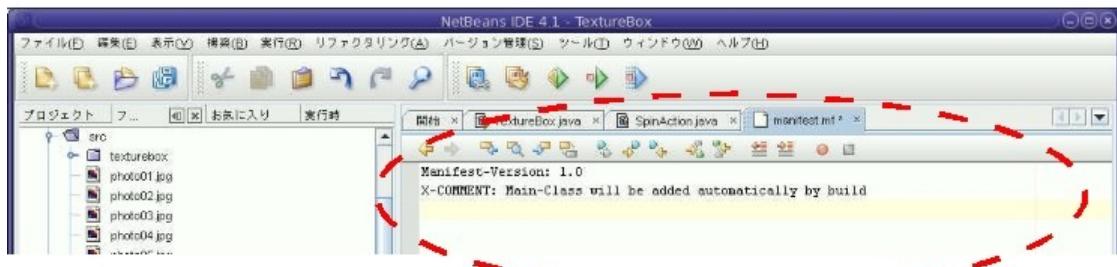
2. 左パネルのタブから「**ファイル**」を選択します。
パネル内の「src」を選択します。
右クリックするとメニューが出ますので「**ペースト**」を選択します。



3. 左側のパネルの最下部にある「manifest.mf」を選択します。
右クリックを押すとメニューが出ますので「**開く**」を選択します。



4. 右側のソースコードエディタに「**manifest.mf**」が表示され、編集可能な状態になります。



5. manifest.mf を編集し、

- ・ アプリケーション名
- ・ メインクラス
- ・ アイコンファイル

を追加します。

追加部分は**赤(太字)**で示します。

```
Manifest-Version: 1.0
X-COMMENT: Main-Class will be added automatically by build
Implementation-Title: Texture Box
Main-Class: texturebox.TextureBox
Icon-Filename: texturebox.png
```

6. プログラムの変更が終了したら、左側のパネルのタブから「**プロジェクト**」を選択し、「**プロジェクトの構築**」を行います。

JARベースアプリケーションの作成はこれで終わりです。

アプリケーションは プロジェクトのフォルダ(ディレクトリ)内の「**dist**」に置かれます。

今回の場合は **/home/duke/TextureBox/dist/texturebox.jar** が作成したアプリケーションです。

JARベースアプリケーションの実行

作成した JAR ベースアプリケーションを LG3D に登録します。

LG3D 本体のフォルダ(ディレクトリ)にある ext/app フォルダ(ディレクトリ)が 作成したアプリケーションの登録フォルダになります(今回の場合 **/home/duke/lg3d/ext/app**)。ここに作成した JAR ベースアプリケーションをコピーすれば登録完了です。

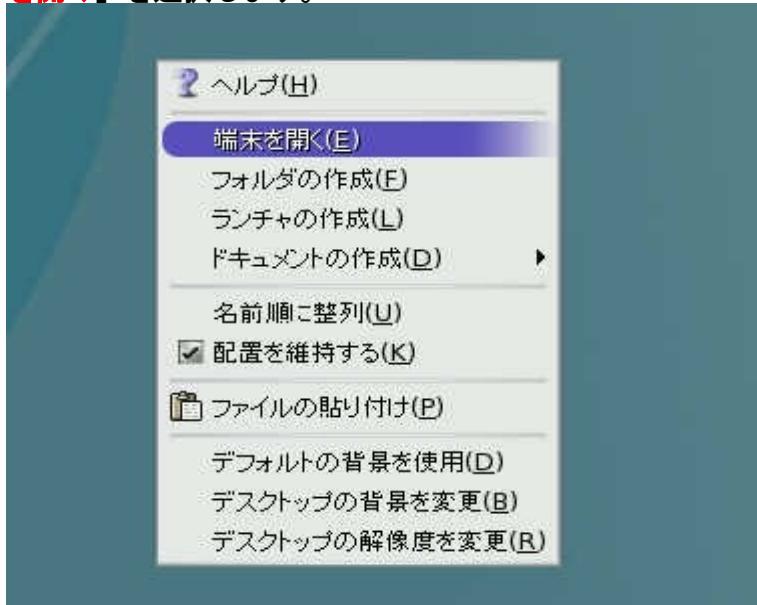
アプリケーションの登録がうまくいけば、LG3D のタスクバーにアプリケーションの起動アイコンが追加されます。

補足: LG3D 0.7.1 からスタートメニューが導入されました。

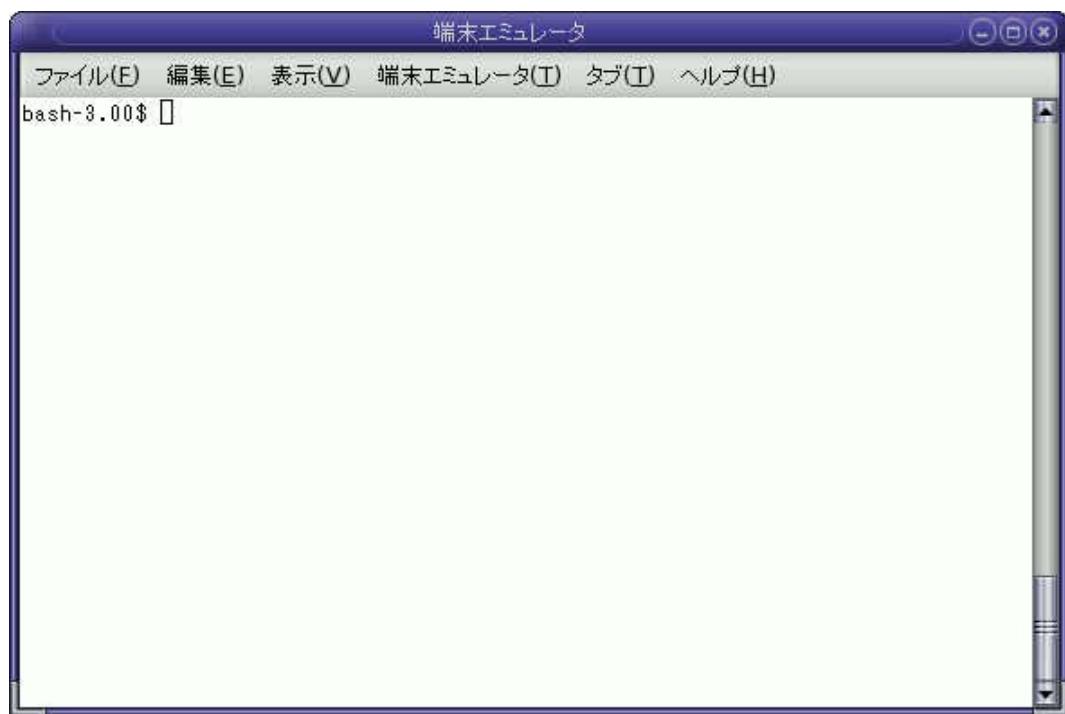
ですが、スタートメニューの登録方法は将来のバージョンで変更される予定のため、このテキストではスタートメニューへの登録方法の解説は省略しています。

以下に、作成したアプリケーションを登録して、実行する方法を示します。

1. Solaris 10 の JDS 環境では背景で右クリックするとメニューが表示されますので、「**端末を開く**」を選択します。



2. 次のような端末が表示されますので、その上で作業を行います。



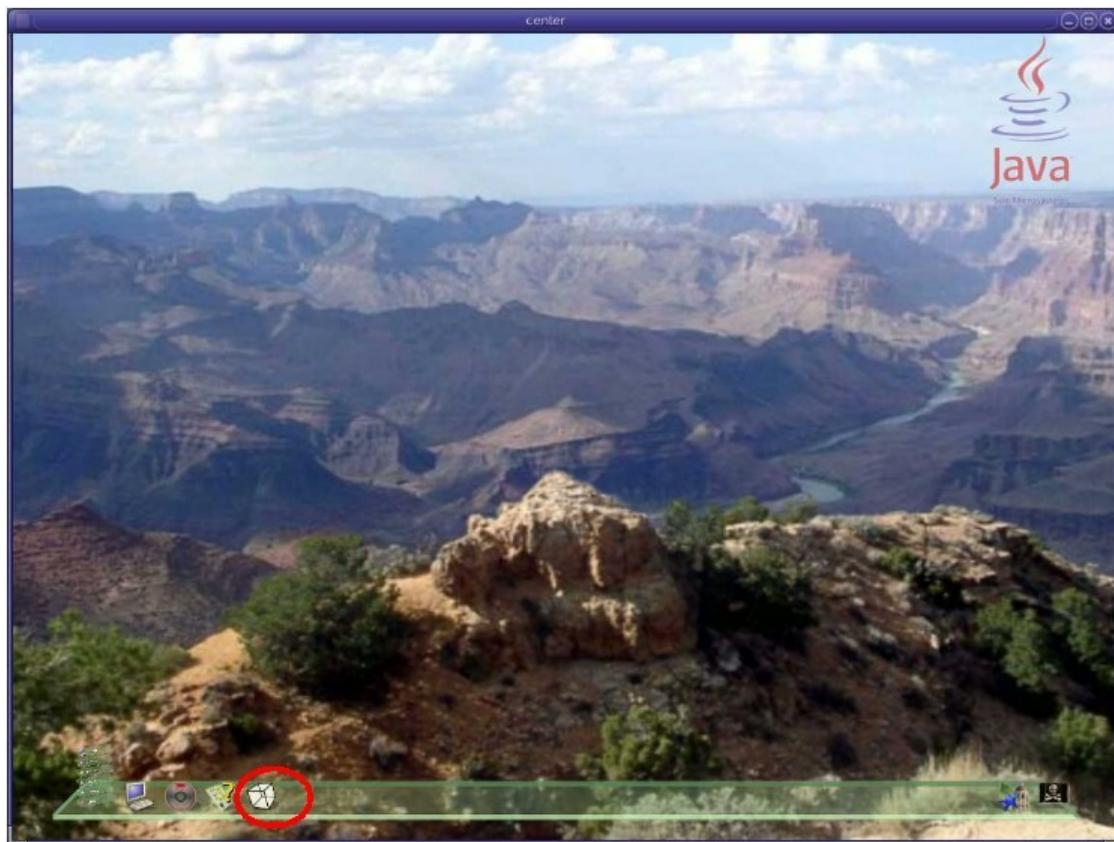
3. 次のコマンドを実行して、JARベースアプリケーションを LG3D のディレクトリにコピーします。(# を入力する必要はありません。)

```
# cp ~/TextureBox/dist/texturebox.jar ~/lg3d/ext/app/.
```

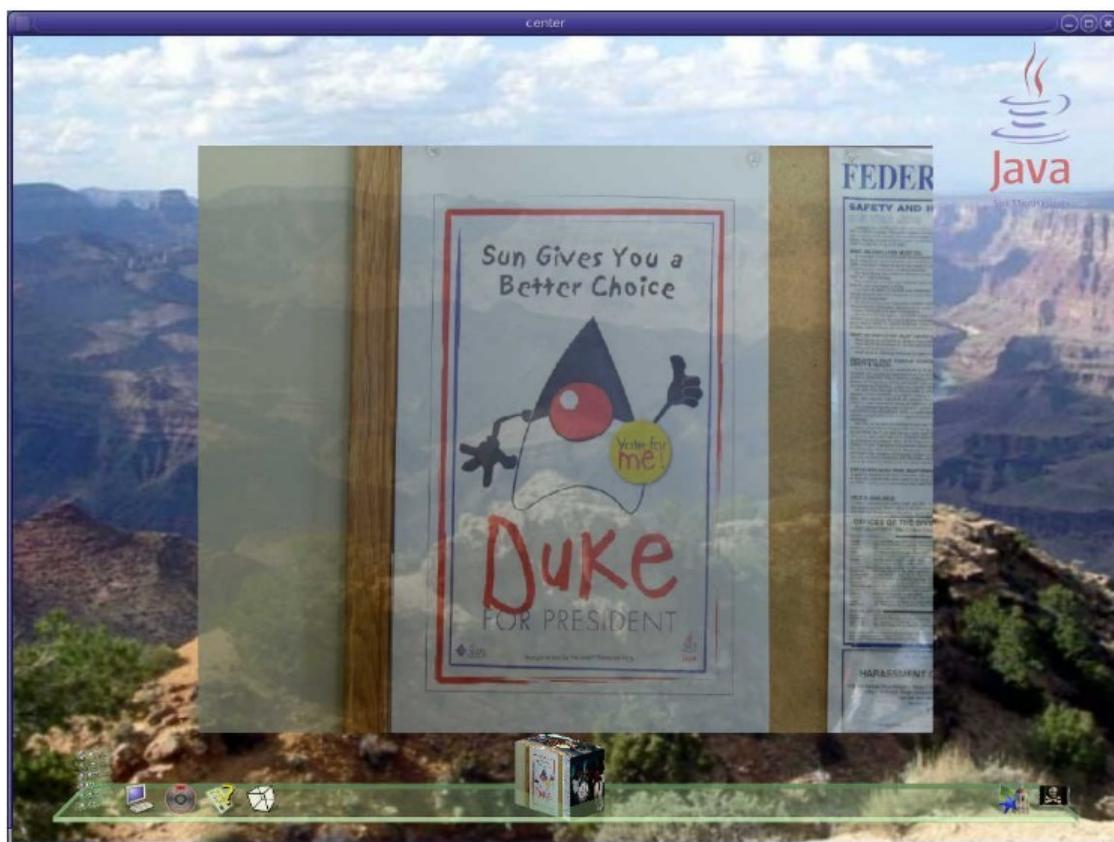
4. 次のコマンドを実行し、LG3D を起動します。

```
# cd lg3d  
# export JAVA_HOME=/home/duke/lg3d/jdk1.5.0_05  
# bin/lg3d-dev
```

5. 下図のように JAR ベースアプリケーションのアイコンが表示されます。クリックして実行します。



6. 下図はアイコンをクリックしてアプリケーションを起動した後の画面です。



応用問題

/home/duke/Documents/extra 以下には今回のスライド用アプリケーションで使用した画像があります。 それらを利用して、以下の改良に挑戦してください。

- ・ サムネイル上で中クリックで最小化

Thumbnail オブジェクトにイベント処理を追加します。

Frame3D.changeVisible(boolean) メソッドを利用することにより 最小化を行うことができます。

- ・ 最小化、終了ボタンの作成

Frame3D.changeEnabled(boolean) メソッドを利用することにより アプリケーションを終了することができます。

また、lg3d-core.jar には

- ・ resources/images/button/window-close.png
- ・ resources/images/button/window-minimize.png

が含まれていますので、最小化、終了ボタン作成にこれらを利用するのも良いでしょう。

- ・ 4枚以上の画像に対応する(表示画像を切り替えていく)

「4.3 テクスチャを使う」ではボックスの全ての面に違う画像を表示させましたが、ここではボックスの上下の面は画像の表示対象からはずします。

一番お手軽なのは、画像ごとに SimpleAppearance オブジェクトを生成し、Shape3D.setAppearance(Appearance) メソッドを利用する方法です。

- ・ ボックス以外のシェイプを使う

Box 以外のシェイプとして

- ・ org.jdesktop.lg3d.utils.shape.ImagePanel
- ・ org.jdesktop.lg3d.utils.shape.FuzzyEdgePanel

などがあります。 ImagePanel と FuzzyEdgePanel はテクスチャを貼るために用意されている シェイプなので、Box のように GENERATE_TEXTURE_COORDS を指定をする 必要はありません。

- ・ ページ番号の表示を行う

org.jdesktop.lg3d.utils.shape.Text2D クラスを使う方法が簡単です。 Text2D は簡単な文字列を扱うためのシェイプです。 Text2D.setString(String) で文字列の変更が行えます。 文字が大きすぎる場合は Component3D オブジェクトに格納後 Component3D.setScale(float) で拡大・縮小が行えます。

参照文献、URL 等

公式サイト

- Project Looking Glass の総本山
<http://lg3d.dev.java.net/>
 - Project Looking Glass Core
バイナリ、ドキュメント、FAQ など
<http://lg3d-core.dev.java.net/>(英語)
<http://lg3d-core.dev.java.net/ja/>(日本語)
-

ML/掲示板

- 日本語 ML : interest_ja@lg3d.dev.java.net
登録ページ : 登録には java.net のアカウントが必要です(無料)
<https://lg3d.dev.java.net/servlets/ProjectMailingListList>
 - 英語 ML : interest@lg3d.dev.java.net
登録ページ : 登録には java.net のアカウントが必要です(無料)
<https://lg3d.dev.java.net/servlets/ProjectMailingListList>
 - 日本語掲示板 (上記の日本語 ML と連動しています)
閲覧は誰でも可能ですが、投稿には javadesktop.org のアカウントが必要です (無料)
<http://www.javadesktop.org/forums/forum.jspa?forumID=57>
 - 英語掲示板 (上記の英語 ML と連動しています)
閲覧は誰でも可能ですが、投稿には javadesktop.org のアカウントが必要です (無料)
<http://www.javadesktop.org/forums/forum.jspa?forumID=56>
-

Web サイト

- Java in the Box 内 Project Looking Glass のページ (櫻庭祐一さん)
インストール、プログラミング、リンク、Eclipse プラグインの情報など
<http://www.javainthebox.net/lg3d/index.html>
-

雑誌

- Software Design 10月号
Project Looking Glass 徹底攻略 川原 英哉、藤楓 泰宏、かみや たま、櫻庭 祐一 他
<http://www.gihyo.co.jp/magazines/SD/archive/200510/>