Enfin, OpenCV est disponible pour les codeurs java !

OpenCV est une bibliothèque de traitement d'image ou plus généralement une bibliothèque graphique développée principalement avec le langage C++ par la société Intel dont Willow Garage assure le support depuis 2008. D'autres grands acteurs du domaine s'intéressent à OpenCV , tel que Nvidia. Cette dernière développe depuis quelques années des briques utilisant CUDA pour OpenCV . Ceci permet d'exploiter les avancées en termes de puissance de calcul graphique GPU.

Début 2007 et début octobre 2008, sont deux dates connues par l'apparition des smartphones équipés par un iOS ou par un androïd. Ces deux merveilles ont révolutionné le marché des Smartphones et des tablettes. En plus des fonctionnalités d'un simple téléphone, un Smartphone peut contenir des centaines d'applications allant d'une simple application agenda à une application plus complexe utilisant plus de ressources, réseaux, capteurs (caméra), etc. Les applications pour iPhone sont développées avec le langage de programmation Objective-C, tandis que les applications pour Androïd sont développées avec le langage de programmation Java. Ce dernier est connu pour sa portabilité. Un Smartphone comporte le plus souvent une caméra produisant des images de très bonne qualité. Outre la reprise des photos, une caméra peut être utilisée dans plusieurs applications, webcam, réalité virtuelle, reconstruction 3D, reconnaissances d'objets, etc. Ces dernières applications nécessitent des algorithmes de traitement et d'analyse d'images plus complexes. Heureusement, OpenCV est leader dans le domaine et surtout est très en avance sur ces sujets. Alors pourquoi réinventer la roue ? Il est donc impératif pour la librairie OpenCV de développer des interfaces qui permettent de s'interfacer avec d'autres langages et plus particulièrement les deux langages les plus populaires à savoir Java et Python. Dans ce tutoriel, nous nous intéressons plus particulièrement au langage Java.
Pour un développeur java, il existe plusieurs solutions qui permettent d'intégrer du code C++ dans un programme Java. Nous pouvons citer comme exemple : JNI, SWIG, etc. Ces solutions restent très difficiles à mettre en œuvre et surtout à maintenir. Pour chaque version C++, le développeur java réadapte son code pour intégrer les changements. JavaCV est un bon exemple.
Souvent, les développeurs Java abandonnent les avantages d'OpenCV et cherchent plutôt d'autres bibliothèques purement java ou d'autres bibliothèques qui embarquent une interface java. Je donne par exemple imageJ et Orefo ToolBox (otb). Cette dernière est très puissante mais malheureusement elle est limitée aux problématiques de télédétection. Dans cet article, je ne compare pas les librairies en détails, mais au niveau des fonctionnalités, je pense qu'OpenCVdépasse largement ses concurrents, en tout cas dans des applications de traitement d'images plus particulièrement "génériques".
Dans ce guide, vous trouverez une aide pour créer votre premier projet Java en utilisant OpenCV. Pour mener à bien notre objectif, nous allons utiliser Eclipse.
Le guide se déroulera en trois étapes :
Installer OpenCV avec une prise en charge Java.
Créer un projet Eclipse.
Écrire un projet simple OpenCV avec Java.
Installation OpenCV avec support Java
A partir de la version 2.4.4 OpenCV intègre une interface avec le langage Java. Pour bénéficier de cette fonctionnalité, nous allons télécharger et installer la version OpenCV 2.4.4 ou une autre version plus récente depuis le dépôt : http://sourceforge.net/projects/opencvlibrary/. Pour le bon déroulement du tutoriel, nous supposons que cmake, java, ainsi que python sont bien installés. Sinon, pour les utilisateurs de Debian / ubuntu lancer la commande :

```
$>sudo aptitude install cmake default-jdk openjdk-7-jdk python python-dev
```

Ensuite, nous allons cloner OpenCV depuis un serveur git avec la commande

```
$>git clone git://github.com/Itseez/opencv.git
$>cd opencv
$>git checkout 2.4.6
$>mkdir build
$>cd build
```

Avec cmake, nous générons le fichier MakeFile en lançant la commande suivante:

```
$>cmake -DBUILD_SHARED_LIBS=OFF ..
```

L’option BUILD_SHARED_LIBS est désactivée, ceci permet de générer la librairie d’interface java contenant seule toutes les dépendances d’OpenCV.
Ensuite, il faut examiner la sortie de cmake afin de vérifier que le module java sera bien généré. Ci-dessous, vous avez un aperçu de cette sortie. J’ai souligné le module java.

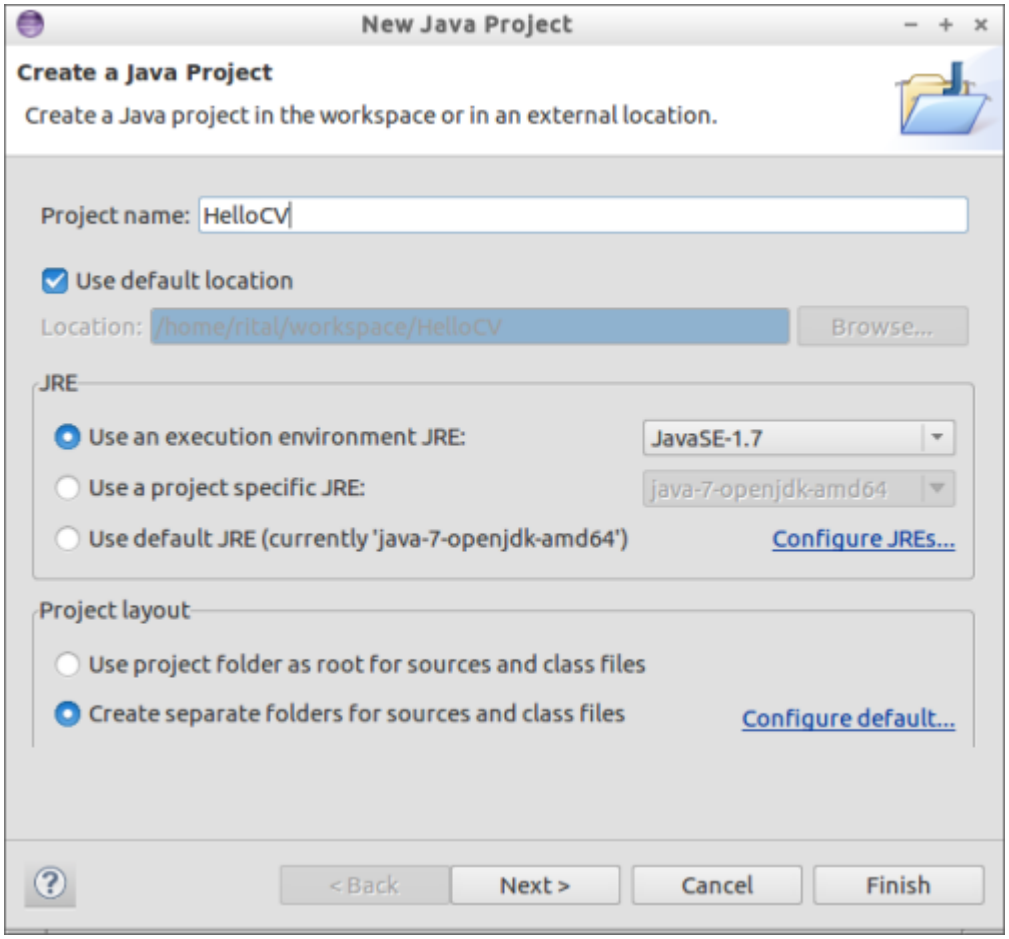
```
--
--   OpenCV modules:
--     To be built:                 core imgproc highgui bioinspired flann features2d calib3d ml objdetect video contrib cudaarithm cudawa
rping cuda cudafilters cudaimgproc legacy cudabgsegm cudacodec cudafeatures2d cudaoptflow cudastereo nonfree photo java optim softcascade
python shape stitching superres ts videostab
--     Disabled:                    viz world
--     Disabled by dependency:      -
--     Unavailable:                 androidcamera cudalegacy cudev matlab ocl
--
--   GUI:
--     QT:                          NO
--     GTK+ 2.x:                    YES (ver 2.24.17)
--     GThread :                    YES (ver 2.36.0)
--     GtkGlExt:                    NO
--     OpenGL support:              NO
--
--   Media I/O:
--     ZLib:                        /usr/lib/x86_64-linux-gnu/libz.so (ver 1.2.7)
--     JPEG:                        /usr/lib/x86_64-linux-gnu/libjpeg.so (ver )
--     WEBP:                        build (ver 0.3.1)
--     PNG:                         /usr/lib/x86_64-linux-gnu/libpng.so (ver 1.2.49)
--     TIFF:                        /usr/lib/x86_64-linux-gnu/libtiff.so (ver 42 - 4.0.2)
--     JPEG 2000:                   build (ver 1.900.1)
--     OpenEXR:                     build (ver 1.7.1)
--
```

J’ai ajouté dans la commande précédente de “aptitude” des dépendances Python afin de générer aussi l’interface Python pour OpenCV . Je ne donnerai pas un exemple de projet Python utilisant OpenCV , mais l’installation est un bon début. Je vous laisse faire la suite.
Une fois la génération de MakeFile est terminée, lancer la compilation :

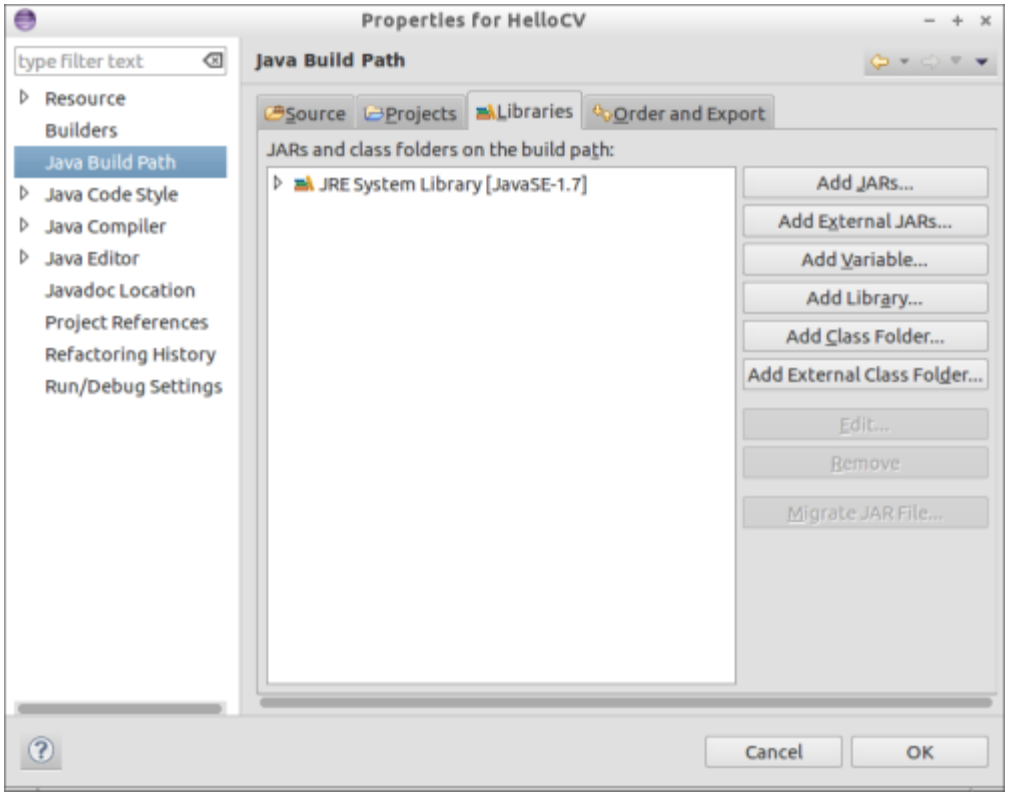
```
$>make -j2
```

```
 10%] Built target opencv_calib3d_pch_dephelp
 10%] Generating opencv_perf_calib3d_pch_dephelp.cxx
canning dependencies of target opencv_perf_calib3d_pch_dephelp
 10%] Building CXX object modules/calib3d/CMakeFiles/opencv_perf_calib3d_pch_dephelp.dir/opencv_perf_calib3d_
elp.cxx.o
 10%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfHuf.cpp.o
inking CXX static library ../../lib/libopencv_perf_calib3d_pch_dephelp.a
 10%] Built target opencv_perf_calib3d_pch_dephelp
 10%] Generating opencv_test_calib3d_pch_dephelp.cxx
canning dependencies of target opencv_test_calib3d_pch_dephelp
 10%] Building CXX object modules/calib3d/CMakeFiles/opencv_test_calib3d_pch_dephelp.dir/opencv_test_calib3d_
elp.cxx.o
 10%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfTileDescriptionAttribute.cpp.o
 10%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfEnvmap.cpp.o
 10%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfOpaqueAttribute.cpp.o
inking CXX static library ../../lib/libopencv_test_calib3d_pch_dephelp.a
 10%] [ 10%] Built target opencv_test_calib3d_pch_dephelp
uilding CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfKeyCodeAttribute.cpp.o
 11%] Generating opencv_ml_pch_dephelp.cxx
canning dependencies of target opencv_ml_pch_dephelp
 11%] Building CXX object modules/ml/CMakeFiles/opencv_ml_pch_dephelp.dir/opencv_ml_pch_dephelp.cxx.o
 11%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfLineOrderAttribute.cpp.o
 11%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/b44ExpLogTable.cpp.o
inking CXX static library ../../lib/libopencv_ml_pch_dephelp.a
 11%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfFloatAttribute.cpp.o
 11%] Built target opencv_ml_pch_dephelp
 11%] Generating opencv_test_ml_pch_dephelp.cxx
canning dependencies of target opencv_test_ml_pch_dephelp
 11%] [ 11%] Building CXX object modules/ml/CMakeFiles/opencv_test_ml_pch_dephelp.dir/opencv_test_ml_pch_deph
o
uilding CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfBoxAttribute.cpp.o
 12%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfPizCompressor.cpp.o
 12%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfRgbaFile.cpp.o
inking CXX static library ../../lib/libopencv_test_ml_pch_dephelp.a
 12%] Built target opencv_test_ml_pch_dephelp
 12%] Generating opencv_video_pch_dephelp.cxx
canning dependencies of target opencv_video_pch_dephelp
 12%] Building CXX object modules/video/CMakeFiles/opencv_video_pch_dephelp.dir/opencv_video_pch_dephelp.cxx.
inking CXX static library ../../lib/libopencv_video_pch_dephelp.a
 12%] Built target opencv_video_pch_dephelp
 12%] [ 12%] Building CXX object 3rdparty/openexr/CMakeFiles/IlmImf.dir/IlmImf/ImfKeyCode.cpp.o
enerating opencv_perf_video_pch_dephelp.cxx
canning dependencies of target opencv_perf_video_pch_dephelp
 12%] Building CXX object modules/video/CMakeFiles/opencv_perf_video_pch_dephelp.dir/opencv_perf_video_pch_de
```

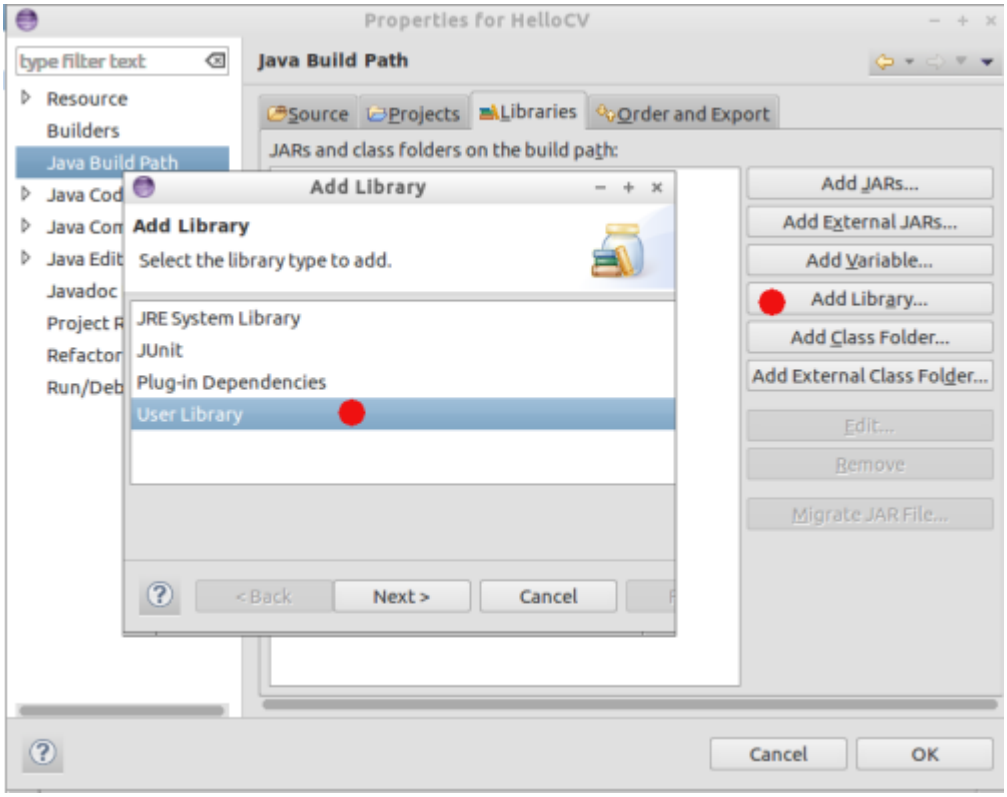
A la fin de ce processus, vous aurez dans le répertoire bin votre interface opencv_2.4.6.jar ainsi que la librairie libopencv_java_246.so. Ces deux fichiers seront utiles et nécessaires pour la prochaine étape.
Création d’un projet Java sous Eclipse
Maintenant nous allons exploiter OpenCV dans un programme Java en utilisant l’IDE Eclipse.
Créer un nouvel espace de travail.
Activer une perspective Java.
Créer un nouveau projet Java : Fichier -> Nouveau –> Projet Java



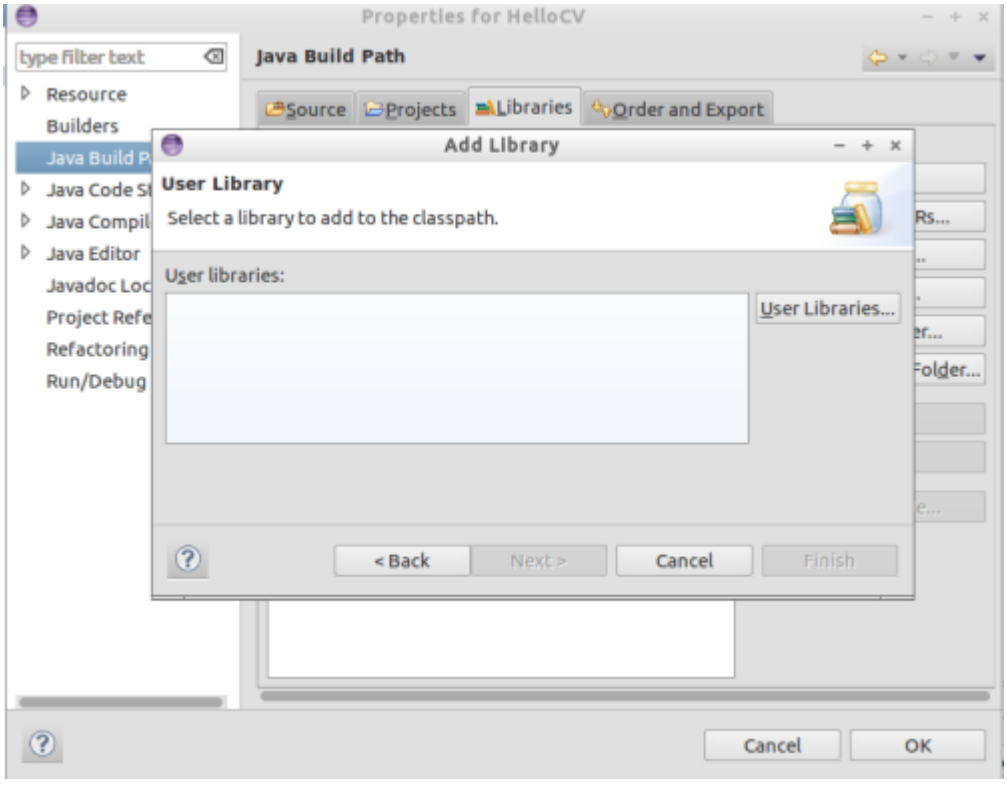
Ouvrir “Java Build Path” à partir de “Project Properties” afin de configurer les librairies :



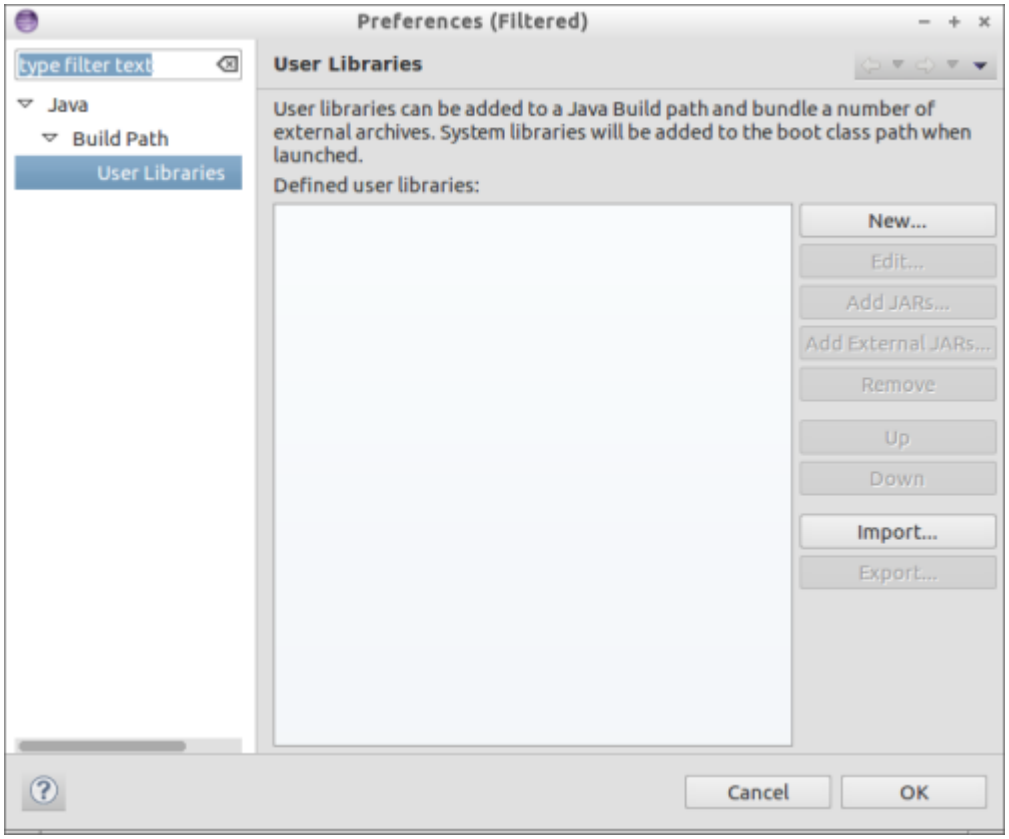
Cliquer sur “add library” , “User Library” puis “Next”



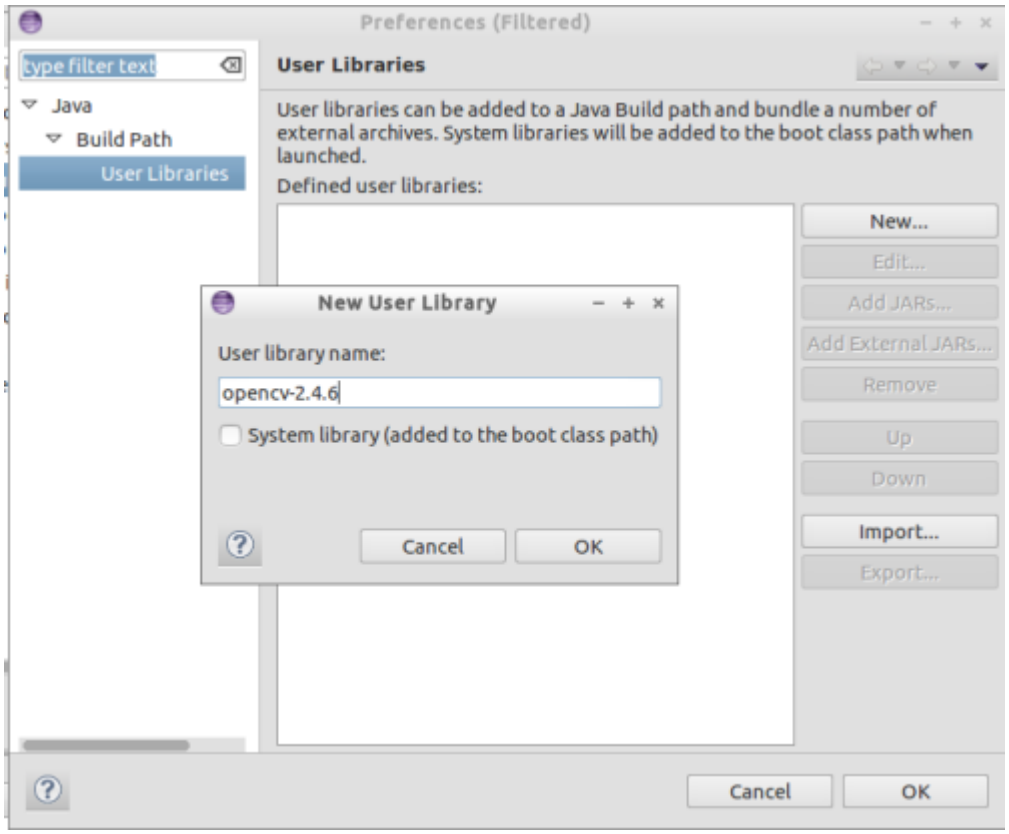
Cliquer sur “User libraries”



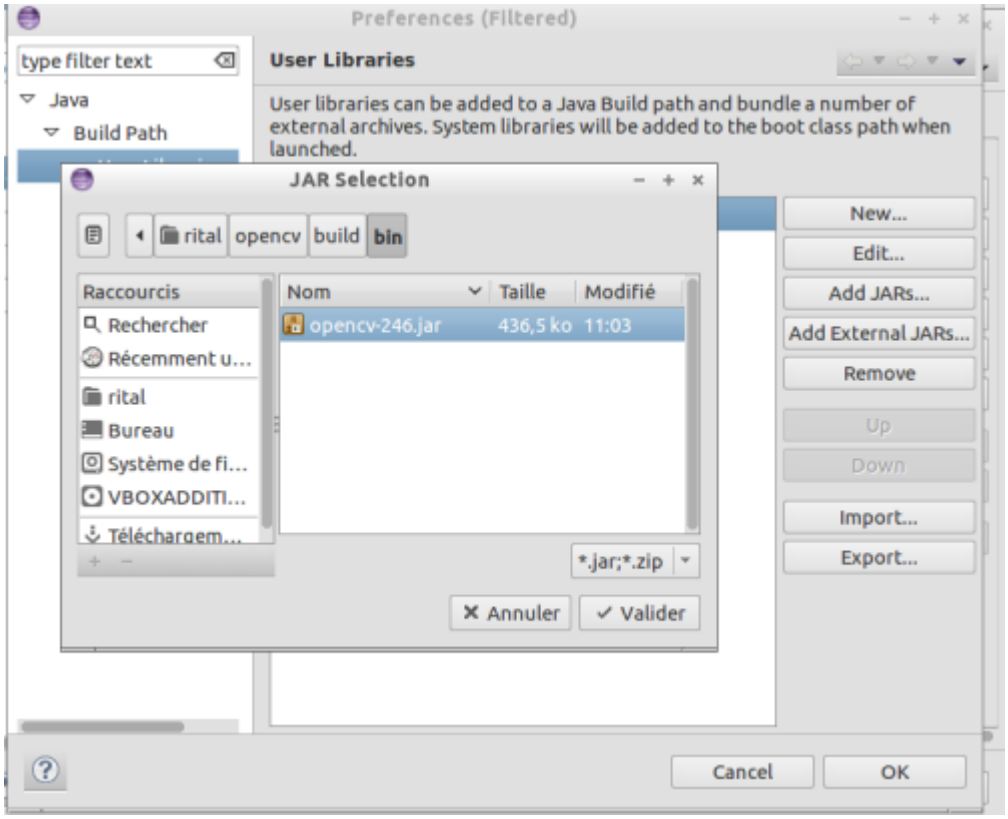
Ensuite cliquer sur “New”



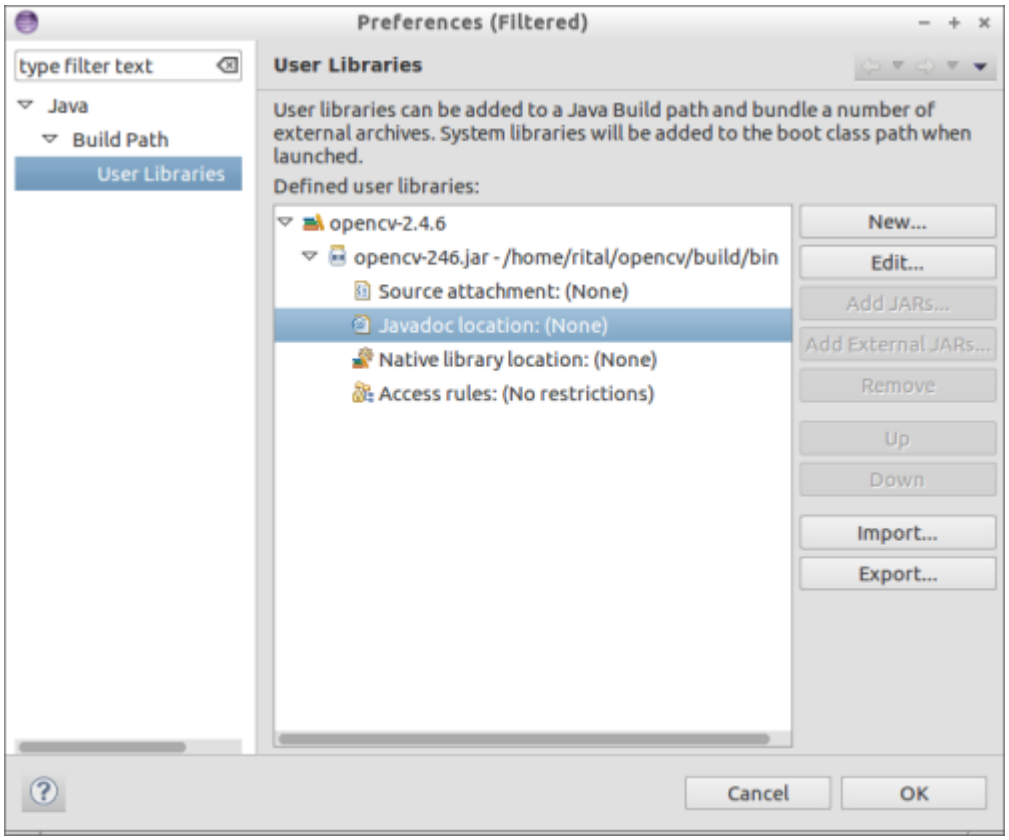
Donner un nom pour votre librairie: par exemple “opencv_2.4.6″ puis “ok”.



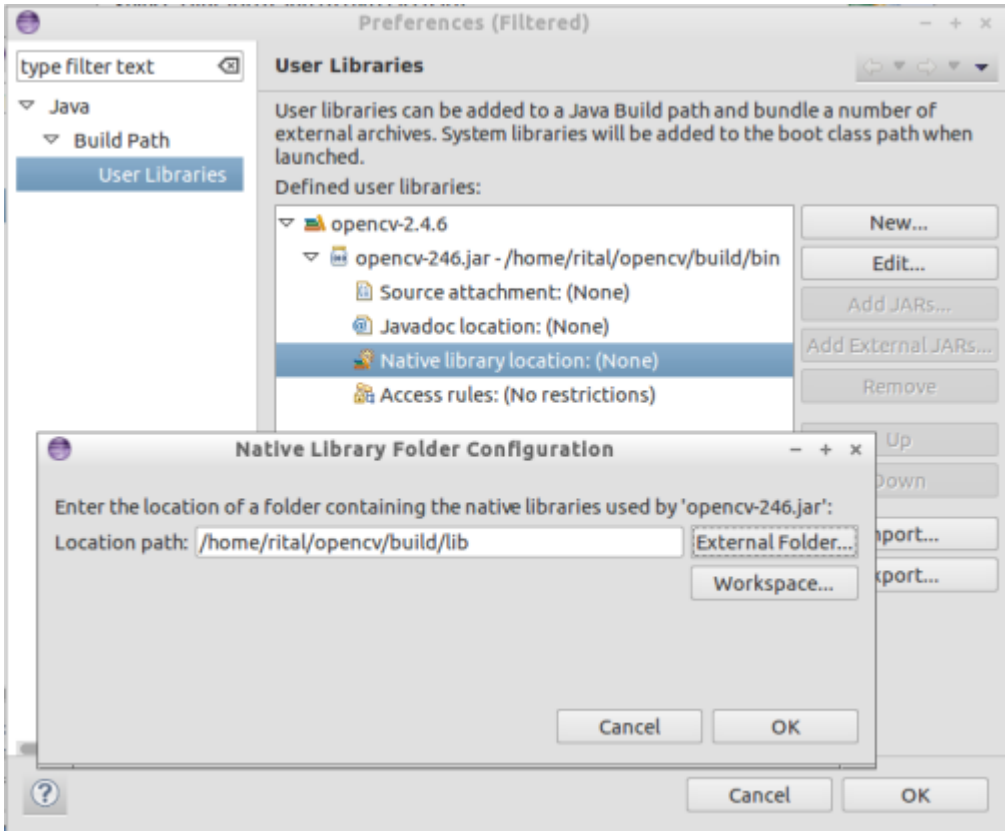
Sélectionner opencv-2.4.6 puis “add external JARs”.



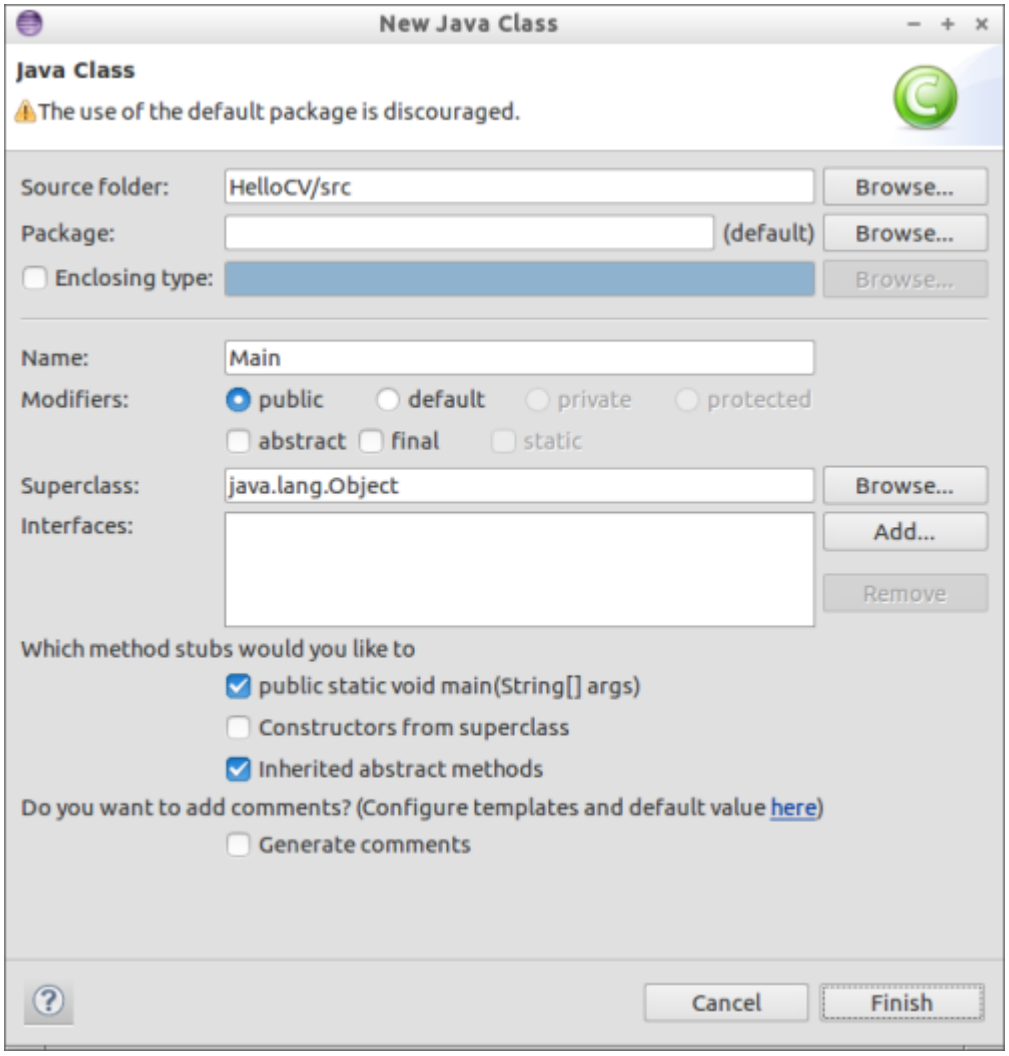
Sélectionner “Native library location” puis “Edit”



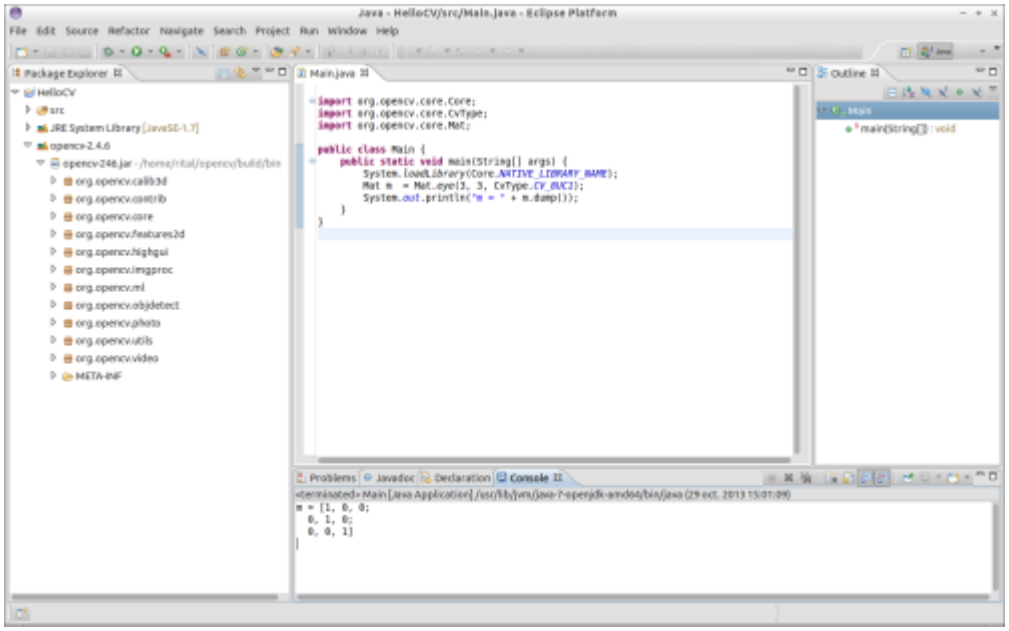
Entrer le chemin vers “libopencv_java246.so” puis ok.



Projet simple OpenCV avec Java
Maintenant, votre projet est configuré avec OpenCV (lib et jar) et il est prêt pour un test. Nous allons maintenant crée une classe Main.java avec le contenu ci-dessous :1



Ensuite, cliquer sur “Run” et trouver dans le console d’Eclipse : la matrice identité.



Conclusion
Avec ces quelques lignes, je pense que vous serez maintenant capable de débuter un projet Java simple utilisant la librairie OpenCV .
Bientôt sur notre site :
Projet Java-Opencv de détection de visage.
Projet de génération d’un dépôt maven incluant la librairie opencv.jar.
Projet Java-Opencv pour Android.