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.

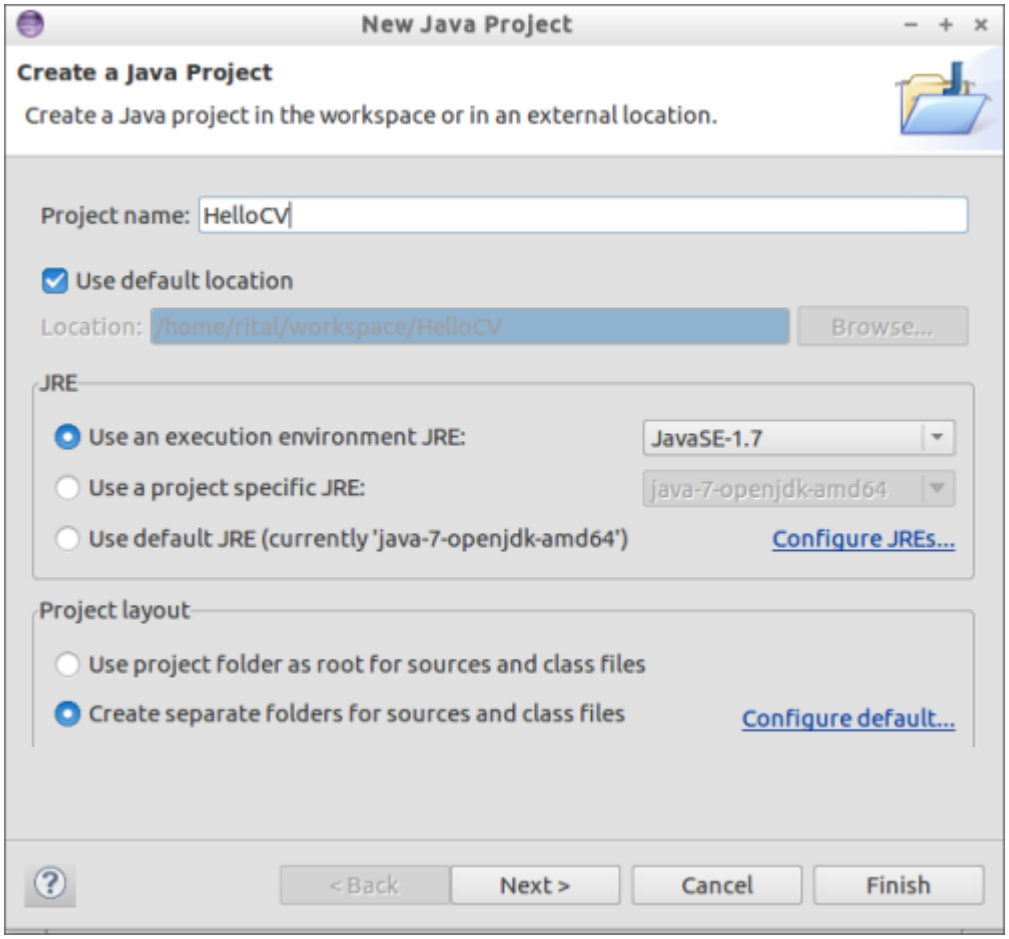
```
--
--   OpenCV modules:
--     To be built:                 core imgproc highgui bioinspired flann features2d calib3d ml objdetect video contrib cudaarithm cudawa
rping cuda cudafilters cudaimgproc legacy cudabgsegm cudacodec cudafeatures2d cudaoptflow cudastereo nonfree photo java optim softcascade
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 :

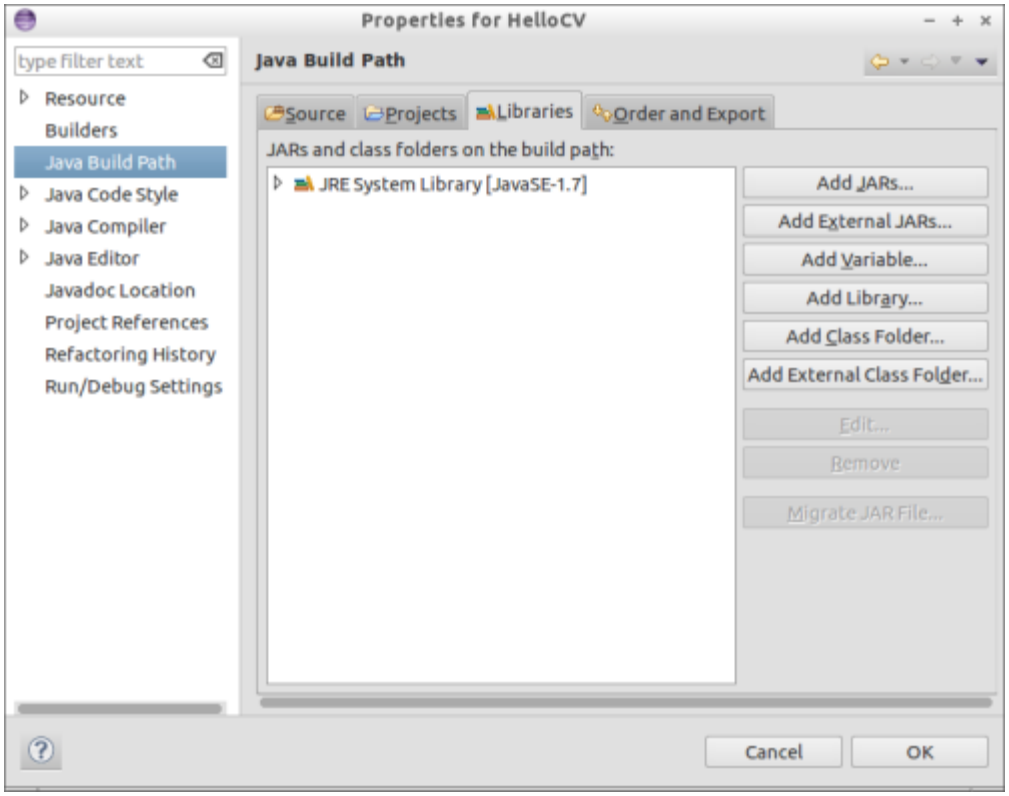
```
$>make -j2
```

```
 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
```

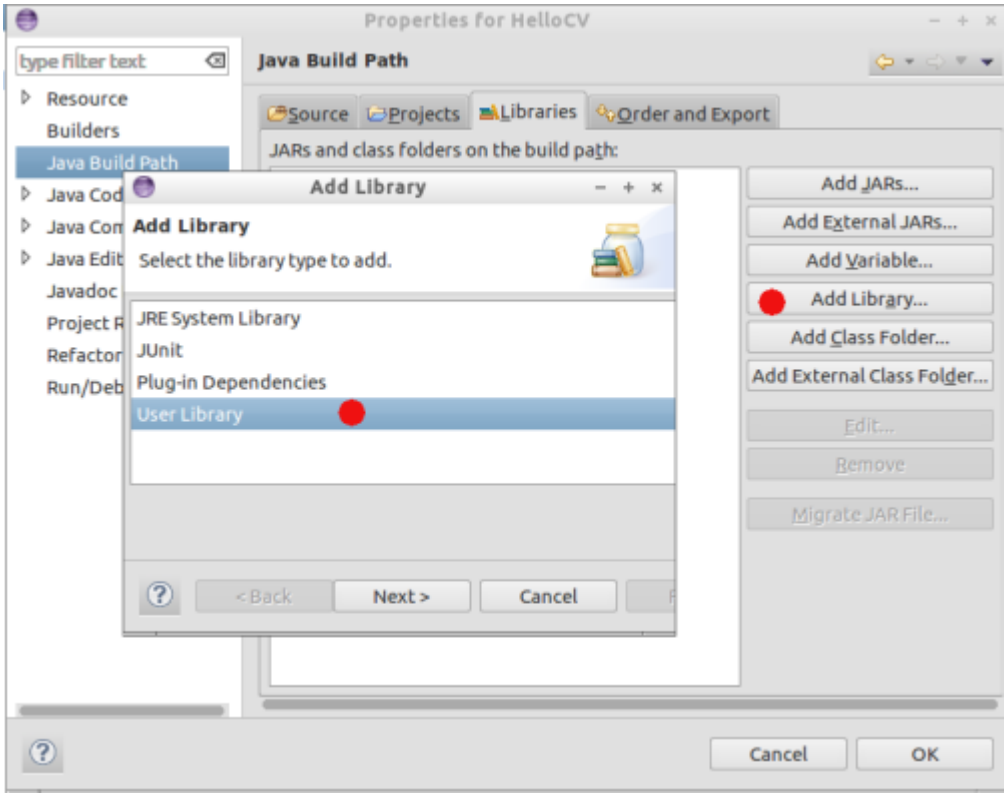
A la fin de ce processus, vous aurez dans le répertoire bin votre interface opencv_2.4.6.jar ainsi que la librairie libopencv_java_246.so. Ces deux fichiers seront utiles et nécessaires pour la prochaine étape.
Création d’un projet Java sous Eclipse
Maintenant nous allons exploiter OpenCV dans un programme 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



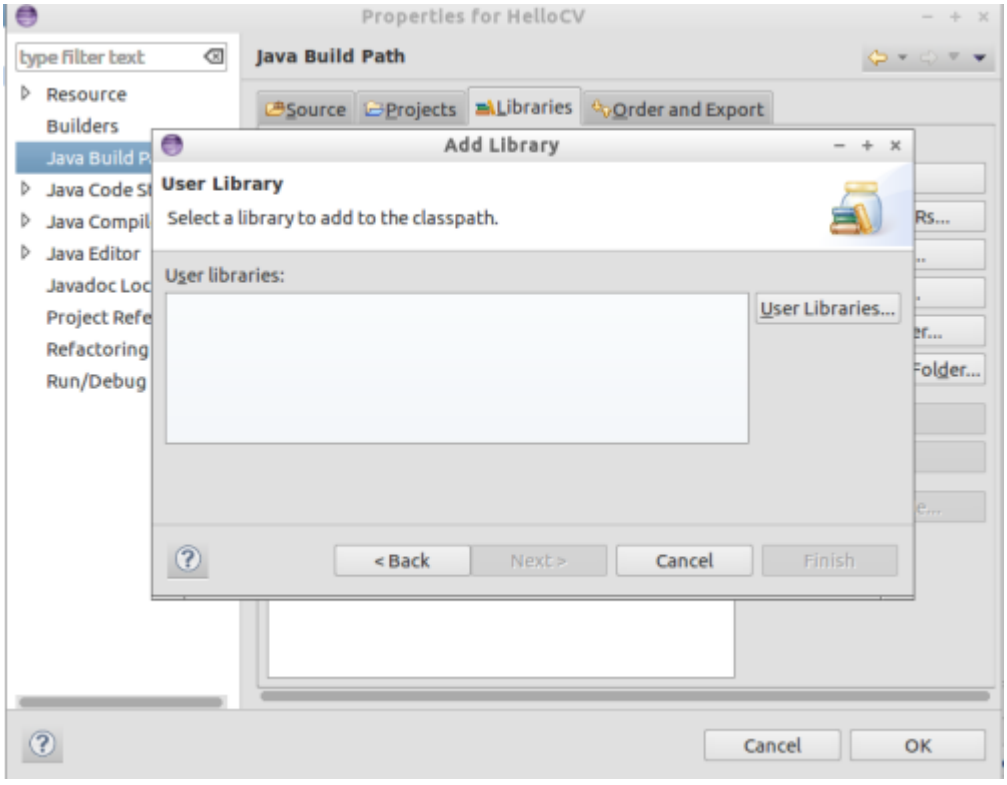
Ouvrir “Java Build Path” à partir de “Project Properties” afin de configurer les librairies :



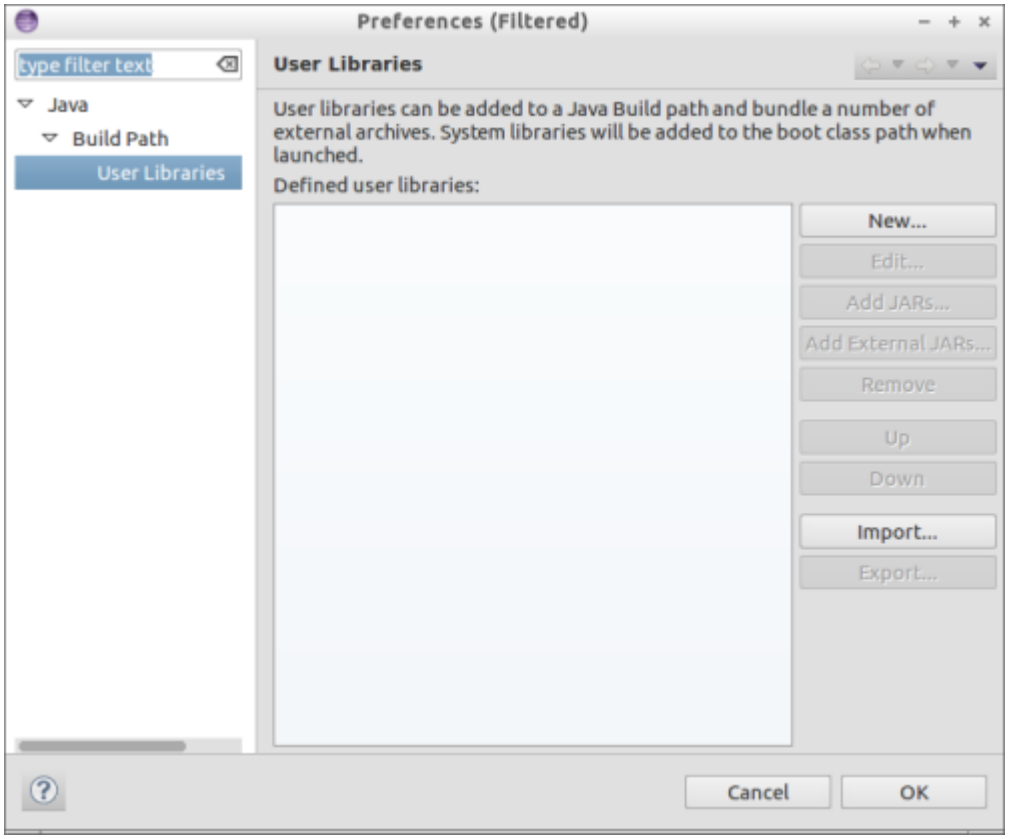
Cliquer sur “add library” , “User Library” puis “Next”



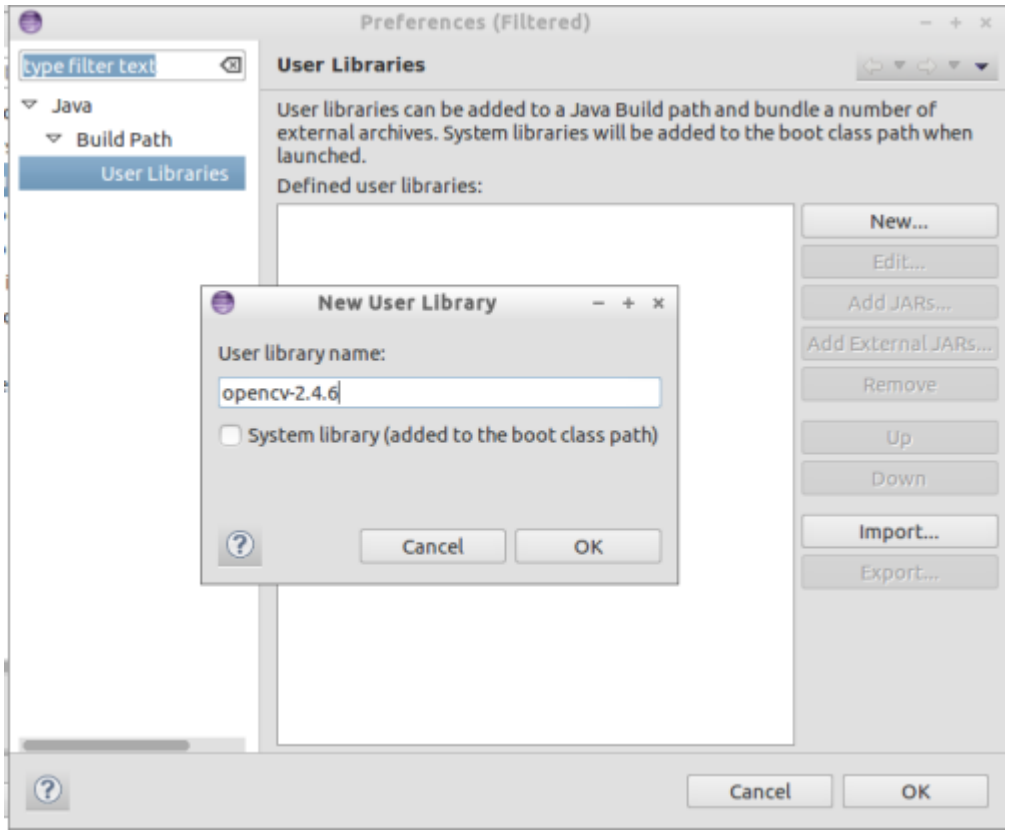
Cliquer sur “User libraries”



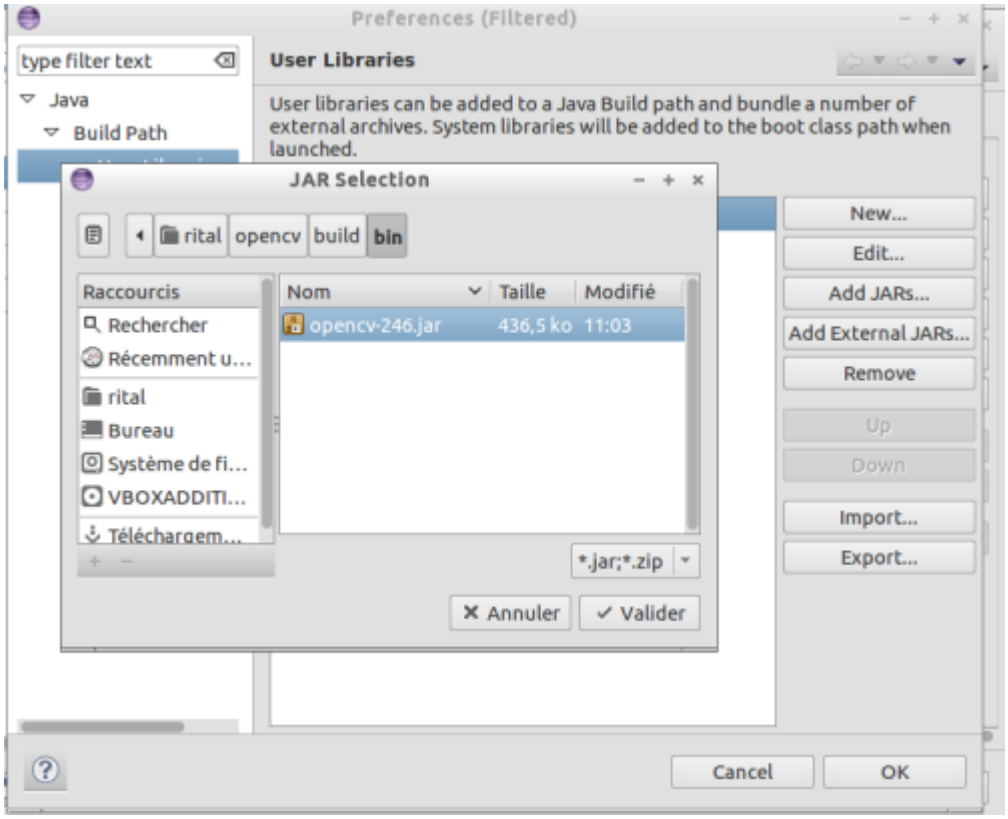
Ensuite cliquer sur “New”



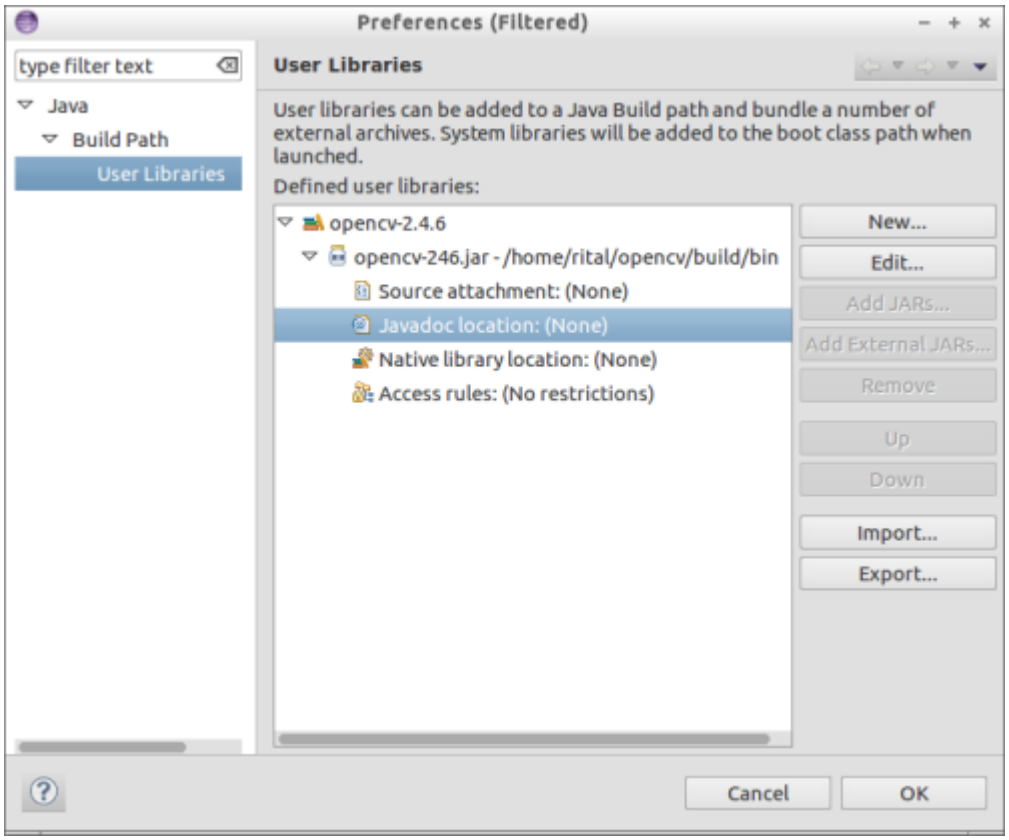
Donner un nom pour votre librairie: par exemple “opencv_2.4.6″ puis “ok”.



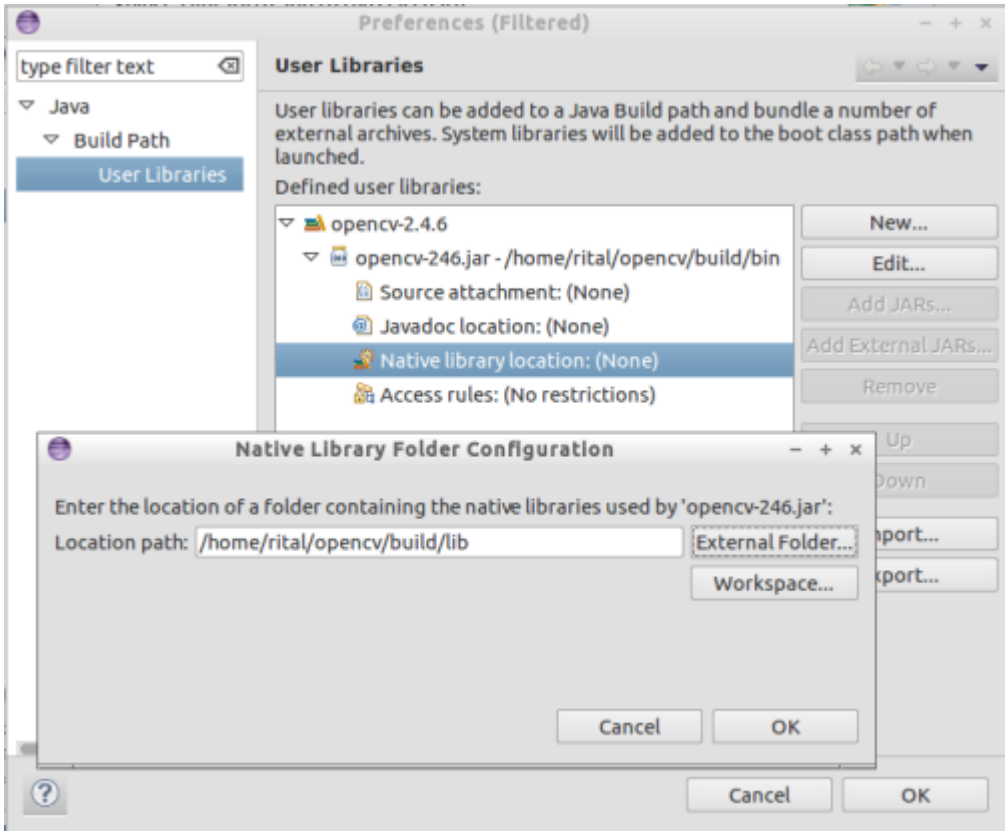
Sélectionner opencv-2.4.6 puis “add external JARs”.



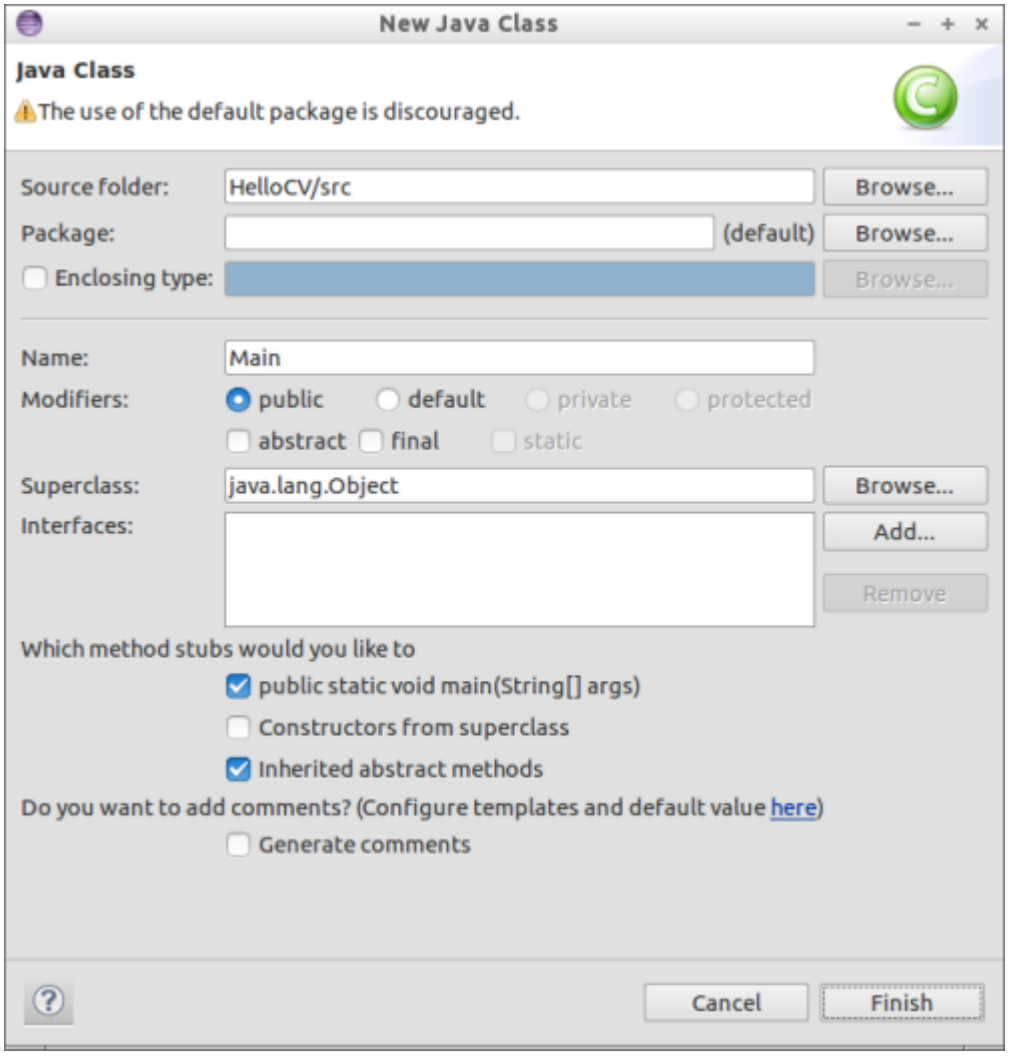
Sélectionner “Native library location” puis “Edit”



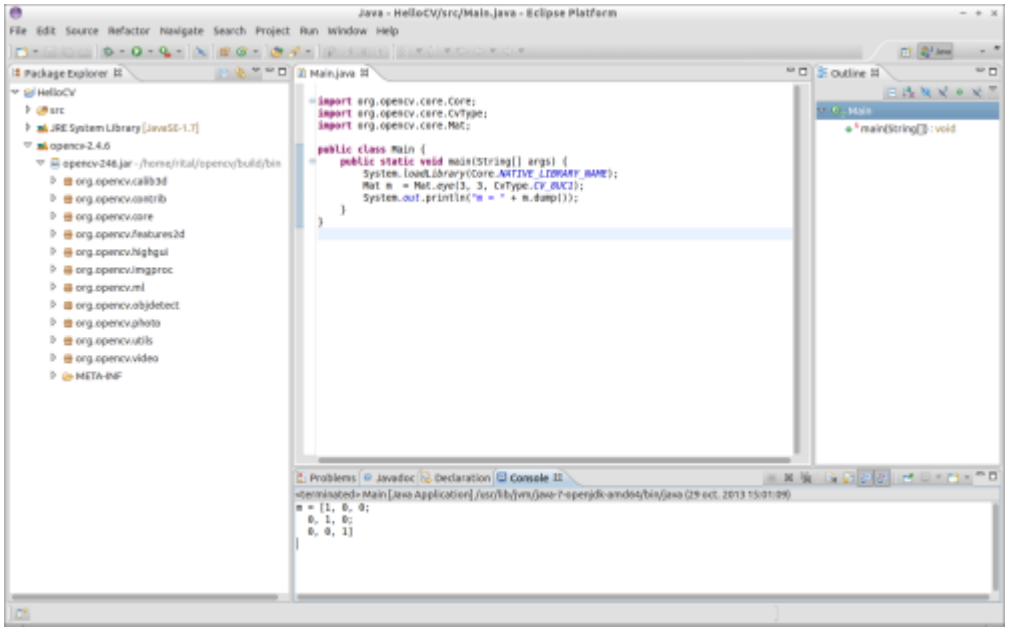
Entrer le chemin vers “libopencv_java246.so” puis ok.



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.

Redis, un moteur de base de données en mémoire

Redis (REmote Dictionary Server) est un moteur de base de données en mémoire développé à partir de 2009 par un développeur italien nommé Salvatore Sanfilippo. Ce dernier ainsi qu'un contributeur important de Redis (Pieter Noordhuis) sont recrutés par VMWare en 2010 pour se consacrer à plein temps sur le développement de Redis tout en laissant en licence libre BSD. Redis est écrit avec le langage de programmation C ANSI. Redis fait partie des solutions NoSQL et il se distingue par sa rapidité, son efficacité et sa légèreté. Par exemple, en terme de rapidité il est très difficile à battre, tant en lecture qu'en écriture. Il peut traiter plus de 100 000 opérations par seconde.

PRESENTATION

Redis est un moteur non relationnel qui opère principalement en mémoire. Il manipule des structures de données clé-valeur. Il est à la fois gestionnaire de ces types de données et un cache de données à la manière de memcached (un système d'usage général servant à gérer la mémoire cache distribuée. Il est souvent utilisé pour augmenter la vitesse de réponse des sites web (Facebook, Twitter, Wikipedia) créés à partir de bases de données. Il gère les données et les objets en RAM de façon à réduire le nombre de fois qu'une même donnée stockée dans un périphérique externe est lue.

TYPES DE DONNEES

Contrairement autres solutions NoSQL qui opèrent sur des documents (json, xml, etc), Redis manipule des paires clé-valeur, les valeurs peuvent être de cinq types de données : chaines, listes, ensembles, hachages et ensembles triés. Ci-dessous, nous détaillons ces cinq types de données.

Chaînes

Une chaîne en Redis est plus qu'une chaîne standard. Elle permet de stocker aussi de stocker des valeurs numériques et des valeurs binaires. Ces dernières permettent par exemple de stocker une image en mémoire. Ceci afin d'éviter le codage et décodage de format image. La taille maximale est de 512 Mo. D'autres commandes sont intégrées spécialement pour modifier des chaînes de type numérique INCR, INCRBY, DECR, DECRBY. Nous verrons toutes ces commandes dans les sections de travaux pratiques.

Listes

Les listes sont simplement des listes de chaînes.

Ensembles

Un ensemble est une collection non triée de chaînes. Quelques opérateurs ensemblistes sont disponibles au niveau de serveur Redis : union, intersection et différence.

Hachages

Il représente un dictionnaire, ce qui permet de stocker des objets par exemple (de la même façon qu'un document JSON). Le stockage est optimisé, ce qui fait que le hachage prend moins de place en mémoire.

Ensembles triés

L'ensemble trié se comporte comme un ensemble où chaque élément de l'ensemble est associé à un poids ou un score qui permet de trier l'ensemble.

PRATIQUES

Dans ces lignes, nous détaillons comment installer Redis, comment le configurer, donner quelques exemples de base pour manipuler les cinq types de données Redis, et enfin un projet complet Redis en utilisant CMake, Hiredis.

Installation

Redis est disponible sous forme de paquet dans plusieurs distributions : Debian, Ubuntu, etc. Sous Ubuntu ou Debian, le paquet est nommé : redis-server. Pour l'installer, il faut tout simplement lancer la commande :

```
#$> sudo aptitude install redis-server
```

Ensuite, une fois l'installation terminée, nous pouvons démarrer le serveur en lançant la commande :

```
#$> nohup redis-server
```

Avec cette commande le serveur démarre avec les paramètres par défaut. Si nous souhaitons les personnaliser alors, nous avons deux façons :

```
#$> nohup redis-server /etc/redis/redis.conf &
```

ou

```
#$> nohup redis-server --port 5555 --slaveof 127.0.0.1 8888 &
```

Dans la première commande les valeurs des paramètres sont renseignées dans le fichier de configuration redis.conf. Cependant dans la deuxième commande, les paramètres sont passés dans la ligne de commande.

Enfin, pour s'assurer que le serveur Redis est bien démarré, vous pouvez lancer la

commande :

```
#$> Redis-cli ping
#$> PONG
```

Information serveur

Avant de personnaliser les paramètres de serveur Redis, il est utile de jeter un coup d'oeil sur la commande : INFO Pour lancer cette commande, il faut passer par redis-cli. Ce dernier est livré avec Redis-server. Redis-cli est une invite interactive qui permet d'envoyer des commandes au serveur.

```
#$> redis-cli
redis 127.0.0.1:6379> INFO
redis_version:2.4.14
redis_git_sha1:00000000
redis_git_dirty:0
arch_bits:64
multiplexing_api:epoll
gcc_version:4.6.2
process_id:5793
uptime_in_seconds:83
uptime_in_days:0
lru_clock:1410105
used_cpu_sys:0.00
used_cpu_user:0.04
used_cpu_sys_children:0.00
used_cpu_user_children:0.00
connected_clients:1
connected_slaves:0
client_longest_output_list:0
client_biggest_input_buf:0
blocked_clients:0
used_memory:726144
...
```

La commande INFO fournit un aperçu utile sur les informations et les paramètres de serveur Redis, y compris la version, ID de processus, la mémoire utilisée, la mémoire disponible, le numéro de port, l'adresse de serveur ; etc.

Configuration

Le serveur Redis possède plusieurs paramètres de configuration. Pour les personnaliser, nous pouvons lancer le serveur avec notre propre fichier de configuration. Afin de faciliter cette tâche, vous pouvez vous servir du fichier de configuration d'exemple redis.conf téléchargeable depuis cette adresse : https://github.com/antirez/redis. Voici un aperçu de fichier de configuration :

Utilisation de redis-cli

Dans cette section, nous utilisons redis-cli pour manipuler les types de données de base dans Redis.

```
redis 127.0.0.1:6379[1]> EXISTS mykey
(integer) 0
redis 127.0.0.1:6379[1]> APPEND mykey "hello"
(integer) 5
redis 127.0.0.1:6379[1]> APPEND mykey " world"
(integer) 11
redis 127.0.0.1:6379[1]> GET mykey
"hello world"
```

Je vérifie si mykey existe dans la base de données, puis je lui associée la valeur hello. Ensuite je concatène à cette valeur le mot word. En fin, avec la commande GET j'affiche le contenu de mykey. Vous remarquez que le serveur Redis répond toujours par le nombre de caractère du contenu de ma clé.

D'autres commandes peuvent être utiles si la valeur associée à une clé est de type numérique.

```
redis 127.0.0.1:6379> SET mykey "10"
OK
redis 127.0.0.1:6379> INCR mykey
(integer) 11
redis 127.0.0.1:6379> GET mykey
"11"
```

La liste des commandes concernant les chaînes se trouve à cette adresse : http://redis.io/commands#string

Avec Hache, nous pouvons stoker un objet. Supposons que nous avons besoin de stocker un article. Avec Redis la commande sera :

```
redis 127.0.0.1:6379> hmset article:1 prenom "Pierre" nom "Durand" email
"pierre.durand@conceptit.fr" titre "Redis, quand on l'essaie, on l'adopte"
OK
redis 127.0.0.1:6379> hvals article:1
1) "Pierre"
2) "Durand"
3) "pierre.durand@conceptit.fr"
4) "Redis, quand on l'essaie, on l'adopte"
redis 127.0.0.1:6379> hget article:1 email
"pierre.durand@conceptit.fr"
```

D’autres commandes concernant cette fois les dictionnaires sont détaillées depuis cette adresse : http://redis.io/commands#hash
Si vous souhaitez utiliser les transactions, Redis met a votre disposition deux commandes MULTI et EXEC. MULTI commance une transaction. Toutes les instructions qui suivent seront stockées dans une file d’attente et ne seront exécutées qu’après l’appel de la commande EXEC. DISCARD permet d’annuler la transaction.

```
redis 127.0.0.1:6379> append mykey "hello"
(integer) 5
redis 127.0.0.1:6379> Multi
OK
redis 127.0.0.1:6379> append mykey " world"
QUEUED
redis 127.0.0.1:6379> get mykey
QUEUED
redis 127.0.0.1:6379> renamenx mykey macle
QUEUED
redis 127.0.0.1:6379> get macle
QUEUED
redis 127.0.0.1:6379> EXEC
1) (integer) 11
2) "hello world"
3) (integer) 1
4) "hello world"
```

APPLICATION CLIENTE
Dans cet article, j’ai choisi de vous donner un exemple d’application totalement développée avec le langage C++. Pour faire le pont entre notre application et Redis, nous utilisons Hiredis. Ce dernier existe en paquet pour debian et ubuntu sous le nom libhiredis-dev pour l’installer en même temps que cmake, il faut taper la commande :
sudo aptitude install cmake libhiredis-dev
Pour avoir cmake en interface graphique, il faut ajouter le paquet : cmake-curses-gui
Ingrédients
CMake,
Redis-server,
libhiredis-dev
Préparation
Dans un répertoire nommé par exemple “projet”, créer deux sous répertoires : build, src.
Ensuite créer un fichier texte nommé CMakeLists.txt.
Dans le répertoire src, nous ajoutons notre programme main.cpp. Ce dernier contient les instructions nécessaires pour interroger notre serveur redis via bien sûr hiredis.
1
Dans le fichier CMakeLists.txt, ajouter ce contenu.

```
cmake_minimum_required(VERSION 2.8)
project(prjredis)
FIND_PACKAGE(HiRedis REQUIRED)
include_directories(${HiRedis_INCLUDE_DIR})
link_directories(${HiRedis_LIBRARY_DIR})
# Sources project
FILE(GLOB MY_SOURCES
src/main.cpp
)
SET (LINKS
${LIBHIREDIS_LIBRARIES}
)
```

```
add_executable(prjredis ${MY_SOURCES})
target_link_libraries(
prjredis
${LINKS}
)
```

Pour compiler notre projet lancer la commande depuis le repertoire build:

```
cmake ..
```

puis lancer l’exécutable prj.

```
./prjredis
```
