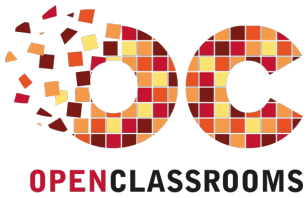
Présentation de bibliothèque SDL en Java

Par demonixis



www.openclassrooms.com

Dernière mise à jour le 24/07/2012

Sommaire

Présentation de bibliothèque SDL en Java

Par  demonixis

Mise à jour : 24/07/2012
Difficulté : Facile

Bonjour à tous,

Aujourd'hui nous allons découvrir et utiliser la librairie *sdljava*. Comme vous devez le savoir, la librairie SDL (« lib SDL » pour les intimes) est principalement utilisée pour créer des jeux vidéos en 2D voire en 3D si on l'utilise avec OpenGL. Pour être capable de suivre ce tutoriel, il est recommandé de connaître suffisamment Java, c'est-à-dire :

- gestion d'un projet ;
- gestion des exceptions ;
- transtypage (*cast*) ;
- approche objet.

La lecture du tutoriel « Programmation en Java » est donc vivement conseillée. Je ne vous en dis pas plus, c'est parti !
La lecture du tutoriel « Apprenez à programmer en C », partie III **peut** être un plus.

Partie 1 : Les bases de sdljava

Dans cette partie, nous verrons les principes fondamentaux pour utiliser la SDL avec Java. J'utiliserai l'EDI Eclipse pour l'installation et la configuration du projet, ce n'est pas très différent avec les autres EDI, mais je n'en parlerai pas. L'installation sera vraiment pas à pas, ainsi, les personnes n'ayant jamais travaillé avec des librairies externes ne seront pas perdues .

Utilisateurs de systèmes 64 bits : vous devez avoir une machine virtuelle Java 32 bits, ainsi que les différents fichiers (.dll ou .a) en version 32 bits, sinon ça ne fonctionnera pas correctement. En effet *sdljava* est compilée pour les système 32 bits.

Installation et préparation du projet

Voici la première étape de l'utilisation de **sdljava** : l'installation et la configuration. Mais avant cela, j'ai des choses à vous dire concernant cette bibliothèque car comme toutes les bibliothèques elle a des avantages, mais aussi des inconvénients. Je ne suis pas là pour vous les donner, mais je vous dirai ce que vous pourrez faire et ce que vous ne pourrez pas faire.

Pour installer cette bibliothèque, vous aurez besoin :

- d'un JDK 1.4 (minimum) ;
- de l'EDI Eclipse ;
- ou de l'EDI Netbeans ;
- de la bibliothèque SDL version C (Linux ou Windows).

Si vous utilisez une JVM 64 bit, vous pourrez rencontrer des problèmes ; je vous encourage donc à utiliser une JVM 32 bit. Que vous utilisiez un système 64 bits Linux ou Windows, l'installation d'une JVM en 32 bits reste possible à moins que vous soyez sur un système entièrement en 64 bits qui ne prend pas en charge l'exécution de code 32 bits.

C'est tout. Allez, on y va !

Le binding sdljava

Binding ?

Nous allons bientôt commencer l'installation, mais avant je voudrais vous parler de cette bibliothèque.
Comme vous le savez la *librairy* SDL est, à l'origine, programmée en langage C puis est compilée pour fonctionner avec ce dernier (et avec le C++). C'est là qu'intervient le mot **binding**. Un *binding*, c'est, en gros, un portage de bibliothèque d'un langage A vers un langage B. Dans *binding*, vous avez *bind* qui signifie en anglais "lier" ; c'est donc une liaison. C'est d'ailleurs pour cela qu'on aura besoin des fichiers libSDLmain.a ou SDL.dll de la bibliothèque standard C pour faire fonctionner notre application.

Citation : Wikipedia

> Un binding (qui est un terme anglais désignant l'action de lier des éléments entre eux) signifie en informatique le fait de permettre l'utilisation d'une bibliothèque logicielle dans un autre langage de programmation que celui avec lequel elle a été écrite.
>
> Nombre de bibliothèques sont écrites dans des langages proches de la machine comme le C ou le C++. Pour utiliser ces bibliothèques dans un langage de plus haut niveau, il est donc nécessaire de réaliser un binding.
>
> La conception d'un binding peut être motivée par le fait de profiter des performances offertes par l'utilisation d'un langage bas niveau que l'on ne peut obtenir avec un langage de plus haut niveau. La réutilisation de code éprouvé peut également être une autre source de motivation.

Contenu de sdljava

La *librairy* sdljava contient tous les éléments de la bibliothèque originale SDL (c'est déjà pas mal), mais contient aussi des extensions. En voici quelques-unes :

- SDL_image : permettra de manipuler plusieurs formats d'image ;
- SDL_mixer : permettra d'ajouter du son et de la musique dans nos applications ;
- SDL_ttf : permettra d'écrire dans notre fenêtre ;
- OpenGL : permettra d'utiliser l'accélération matérielle pour l'affichage.

Ce que vous devez savoir avant de commencer

Cette bibliothèque :

- est un *binding*, c'est-à-dire que vous devrez distribuer les fichiers .a ou/et .dll avec votre application ;
- n'existe pas en binaire pour Mac OS X, et la compilation est assez complexe (je n'y suis pas arrivé) ;
- dont la version "stable" 0.9.1 date de 2005, et dont la version *cvs* n'a pas été modifiée depuis fin 2006.

Préparation de l'installation
Téléchargement de sdljava

Nous allons maintenant télécharger la bibliothèque pour l'utiliser avec Eclipse ou Netbeans. Vous avez deux solutions : soit vous téléchargez les binaires (donc déjà compilée), soit les sources (c'est à vous de compiler). Nous allons privilégier la première méthode si ça ne vous dérange pas. Rendez-vous sur la page de téléchargement de sourceforge pour récupérer la version que vous voulez (Linux ou Windows).

- Version Linux ;
- Version Windows ;
- Pour les suicidaires : les sources.

Bien sûr, nous utilisons un binding, donc comme je vous l'ai expliqué nous devons aussi avoir les fichiers libSDLmain.a ou SDL.dll de SDL version C pour utiliser sdljava.
En avant pour le site officiel de SDL !

Nous n'avons pas besoin de prendre les versions de développement, contrairement au tutoriel de M@teo21 sur le SDL en langage C ; les *runtime library* feront l'affaire. Je tiens à préciser pour les utilisateurs de Linux que le site ne propose que des paquets RPM : suivant votre distribution, il faudra donc rechercher le fichier SDL correspondant. L'idéal serait une compilation de la bibliothèque et c'est ce que nous allons voir tout de suite. Les utilisateurs de Windows peuvent passer cette étape.

Récupération du fichier libSDLmain.a sous Linux

La récupération de libSDLmain.a peut se faire de différentes manières. Par exemple, si vous avez déjà SDL d'installé sur votre machine, vous devriez pouvoir retrouver ce fichier. Mais ici, pour que tout le monde soit sur un même pied d'égalité, nous compilerons les sources de SDL puis nous récupérerons notre fichier libSDLmain.a (tout ça pour un fichier, mais ça en vaut la peine).

Assurez-vous d'avoir installé les outils de développement.

Normalement, si vous avez déjà libsdl installé sur votre machine, vous devriez trouver libSDLmain.a dans `/usr/lib` .

Si vous avez déjà SDL installé mais que vous ne trouvez pas le fichier, vous pouvez suivre ce qui suit, jusqu'à la récupération du fichier. L'installation ne sera pas obligatoire pour vous.

- Téléchargez les sources de SDL ici.
- Décompressez l'archive dans votre répertoire de travail par exemple.
- Ouvrez un terminal, entrez dans le répertoire nouvellement décompressé et tapez :

Code : Console

```
./configure
make
```

Vous avez votre bibliothèque compilée, mais pas installée. Vous pouvez déjà aller dans le répertoire build du répertoire SDL et vous y trouverez... un fichier nommé libSDLmain.a. 

Mettez-le de côté pour l'instant. Si vous avez déjà SDL installée, vous pouvez passer la prochaine étape. À partir de maintenant, vous n'avez plus qu'à vous mettre en super-utilisateur dans le terminal (commande `su` suivie de votre mot de passe ou `sudo` si vous êtes sous debian-like) et à taper `make install`.

Récapitulatif des commandes

Pour distribution non debian-like (Slackware, Mandriva, Fedora, OpenSuse...) :

Code : Console

```
./configure
make
su # saisie du mot de passe
make install
```

Pour distribution debian-like avec sudo (Debian, Ubuntu et dérivés...) :

Code : Console

```
./configure
make
sudo make install
# saisie du mot de passe
```

Voilà ! SDL version C est installée sur votre système et nous avons notre fichier libSDLmain.a.  On peut donc passer à la suite !

Le fichier SDL.dll sous Windows

Rendez-vous ici et téléchargez l'archive, puis décompressez son contenu dans un dossier que vous mettrez de côté.

Je n'ai pas donné de lien pour la version Mac Os X car comme je vous l'ai dit, je n'ai pas réussi à la compiler (les binaires n'existant pas ), mais si j'arrive à faire quelque chose le tutoriel sera actualisé. Si vous êtes sous Mac Os X et que cela vous intéresse, vous pouvez me contacter.

Préparation de l'installation

Bien qu'on ne puisse pas vraiment parler d'installation, car nous ferons uniquement de la copie, il y a des choses à faire dans l'ordre !
Tout d'abord, vous allez décompresser l'archive de sdljava.

Vous pouvez constater qu'il y a plusieurs dossiers. Nous n'en utiliserons que deux, dont un obligatoire, et un vivement conseillé. Nous garderons donc les répertoires : lib et docs.

- lib : contient le jar de sdljava ainsi que les fichiers .so ou .dll qui lui sont propres ;
- docs : contient la javadoc de sdljava ; elle est réellement indispensable !

Vous pouvez donc les mettre de côté avec le fichier libSDLmain.a / SDL.dll et passer à la dernière étape.

Création du projet avec Eclipse

Nous allons maintenant créer notre projet sous Eclipse. L'avantage est qu'Eclipse est multi plates-formes donc en utilisant la même version sous Linux ou Windows, le résultat sera exactement le même.

Création du projet

Lancez Eclipse comme vous en avez pris l'habitude. Ensuite, créez un nouveau projet Java, puis un nouveau package et pour finir une nouvelle classe, qui contient une méthode `main` (n'oubliez pas de cocher la case public static void main(...). Pour notre exemple, j'ai créé un projet qui se nomme *TutorielJavaSDL*, puis un package que j'ai appelé *configuration* et enfin une classe *installation*.
Nous allons maintenant installer la lib' sdljava. Pour cela, rien de plus simple, mais je vais vous faire un petit rappel sur l'architecture d'un projet avec Eclipse.

Les projets sous Eclipse

Quand vous créez un nouveau projet, celui-ci est copié dans le *workspace* (espace de travail), ce dernier est placé par défaut dans `$HOME/workspace` sous Linux et `C:\Documents and Settings\(utilisateur)\workspace` sous Windows 2000/XP (`C:\Utilisateurs\(utilisateur)\workspace` sous Vista). Si vous consultez le *workspace*, vous constatez qu'il y a un répertoire par projet et que ces derniers se composent eux-même de répertoires (bin & src).

En résumé dans le *workspace*

Il y a un répertoire qui porte le nom de votre projet. Ce dernier en contient deux autres :

- bin : contient les fichiers compilés .class ;
- src : contient les fichiers sources .java ;

Si on s'intéresse à un de ces répertoires, on constate qu'il contient encore un répertoire ! Ce sont les packages (par défaut il n'y en a pas) et enfin dans ce répertoire (package), il y a vos sources !

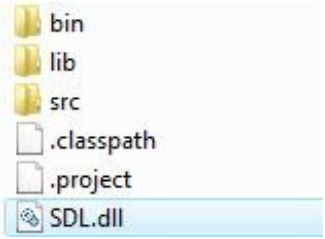
Les conventions de nommage pour ce tutoriel

Je nommerai :

- Répertoire racine : le répertoire qui contient bin & src ;
- Répertoire source : le répertoire qui contient vos sources, c'est-à-dire l'intérieur du package ;
- Même chose pour les binaires !

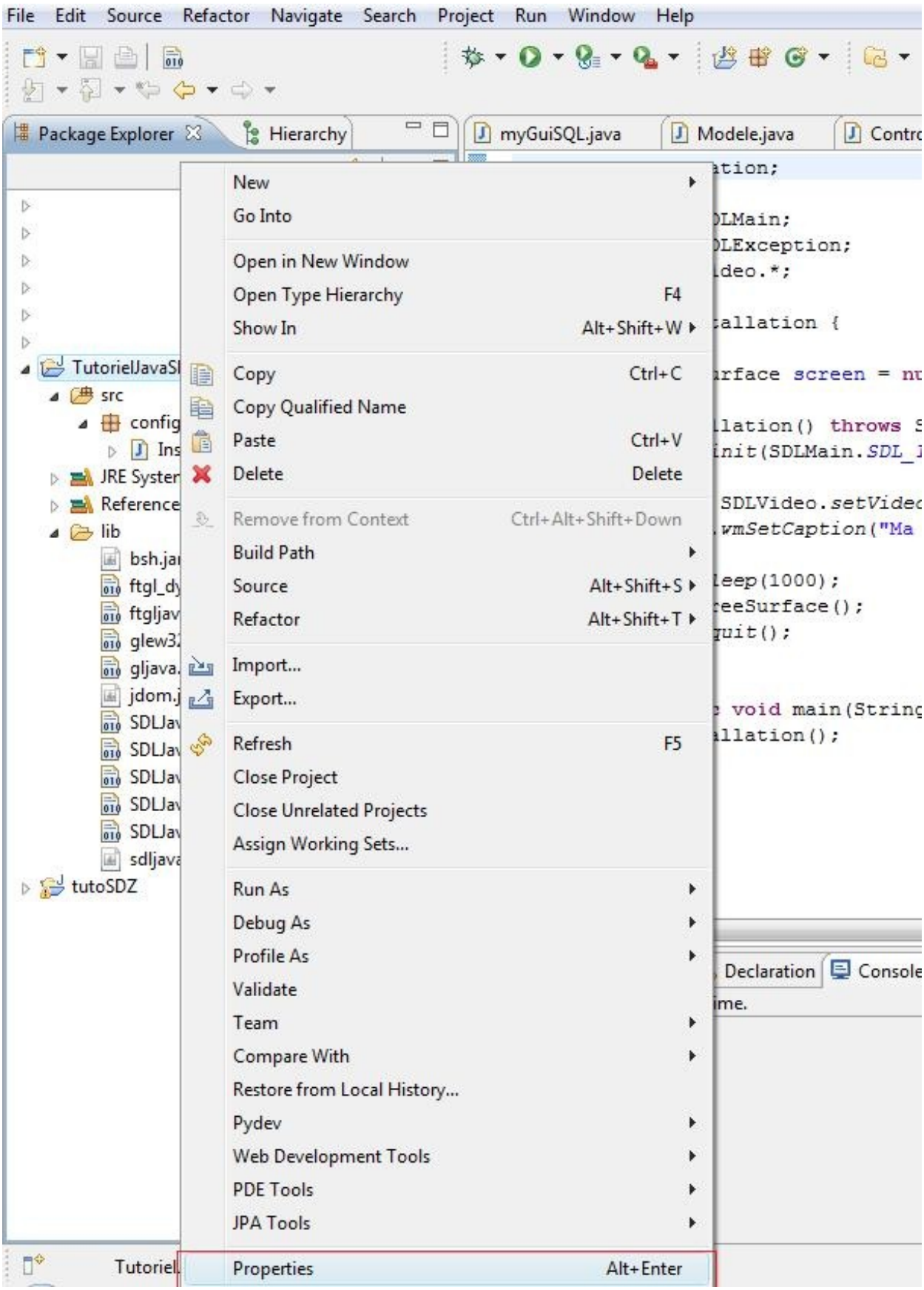
Installation de la bibliothèque

Cette partie est très simple : vous allez copier le dossier lib de sdljava à la racine de votre projet avec le fichier libSDLmain.a ou SDL.dll (selon votre OS). Votre racine doit ressembler à cela :

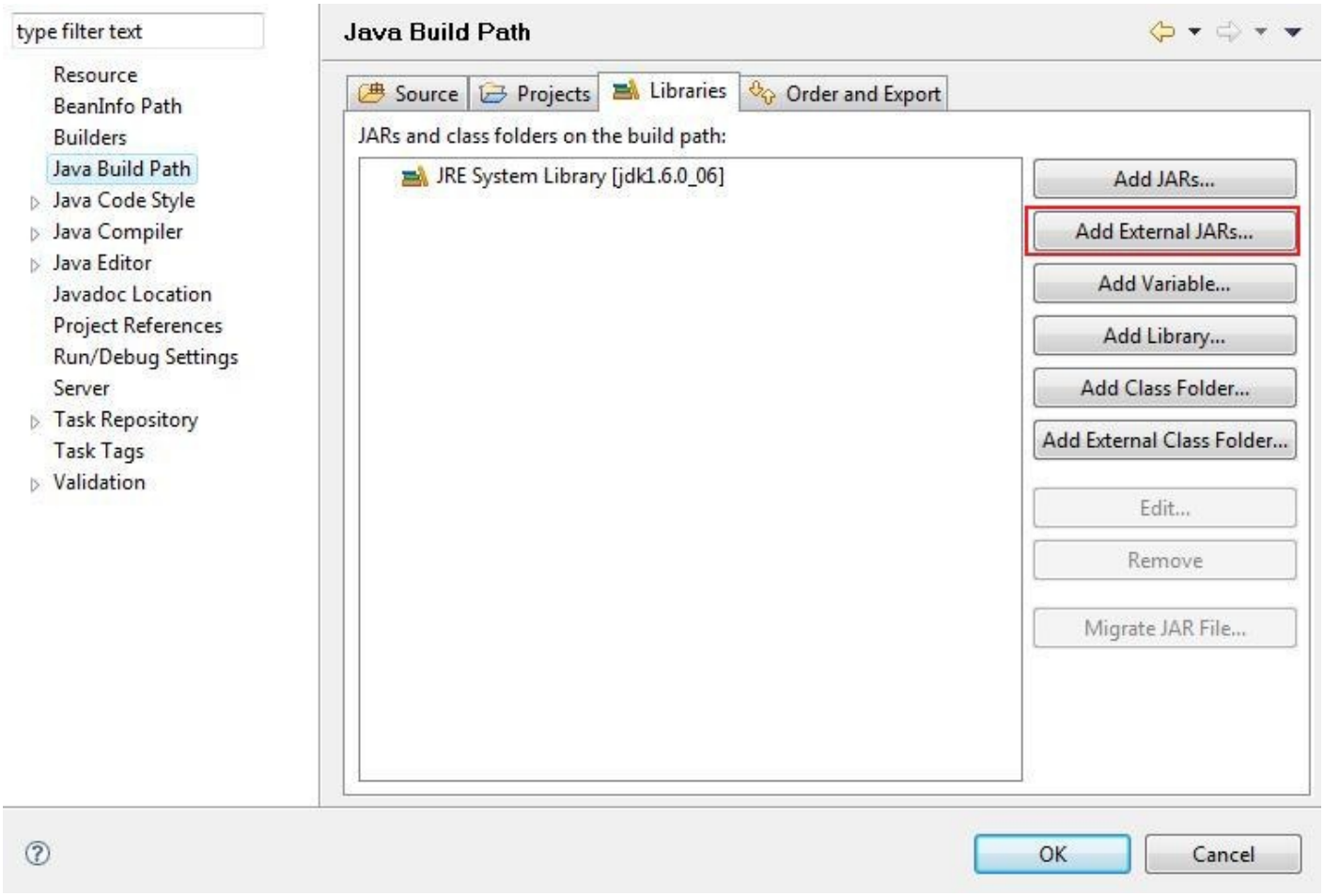


Configuration d'Eclipse

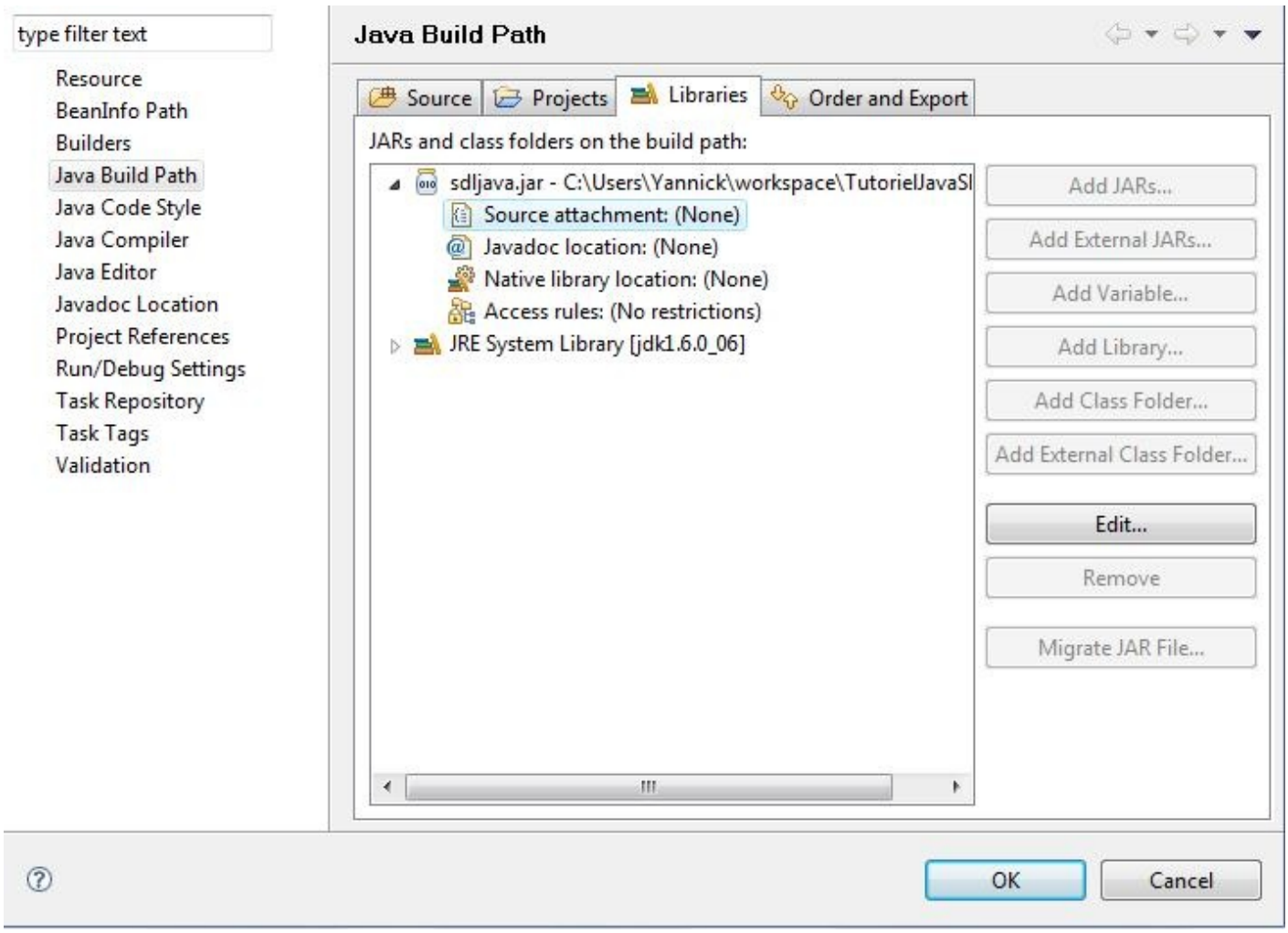
Faites un clic-droit sur votre projet puis sélectionnez Properties (Propriétés). Une fenêtre s'affiche.



Sélectionnez Java Build Path : c'est dans cette partie que nous allons indiquer à Eclipse où est sdljava (car Eclipse ne peut pas le deviner tout seul). Cliquez sur Add External JARs.

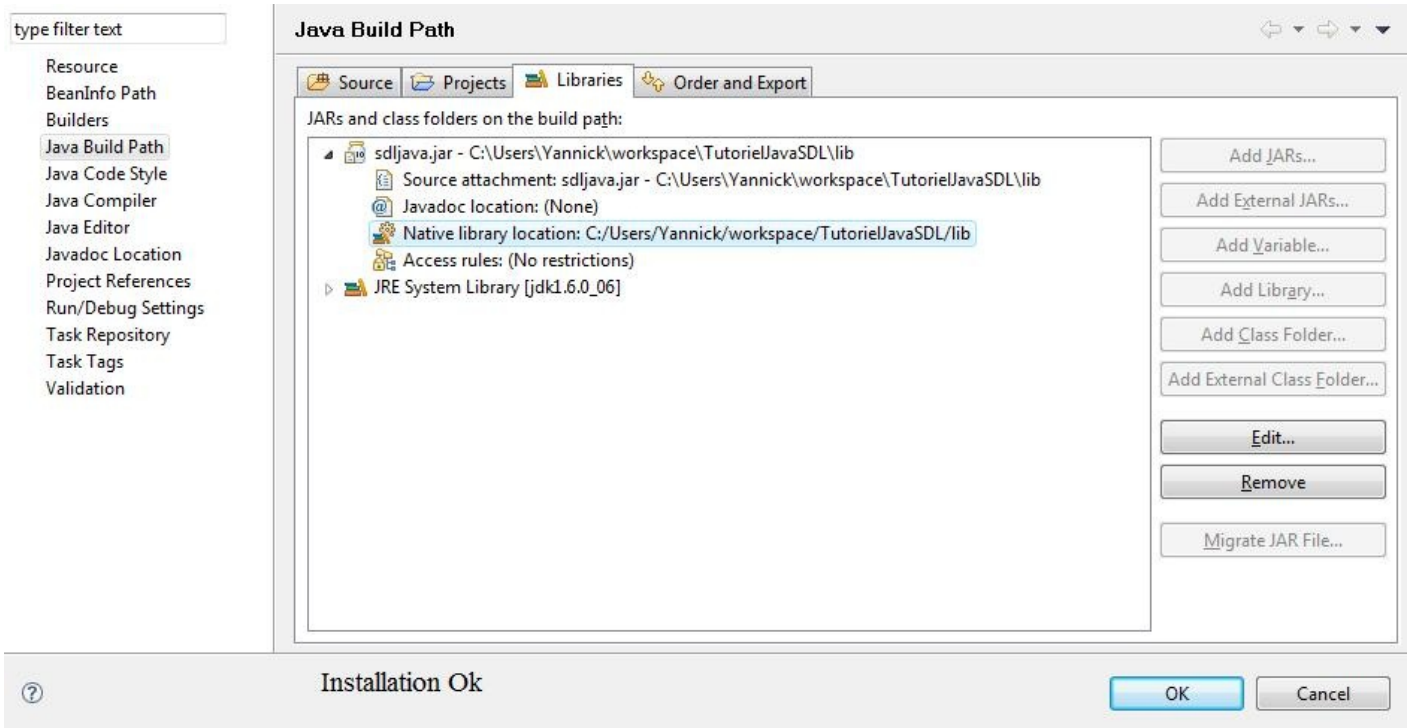


Vous devez sélectionner le fichier sdljava.jar qui se situe dans le répertoire lib de votre projet. Validez, puis cliquez sur la petite flèche noire à gauche de sdljava.jar dans la liste (voir screen).



Cliquez deux fois sur Source attachement et choisissez l'option External File... : vous devez sélectionner encore une fois sdljava.jar (qui, je le rappelle, est présent dans le dossier lib de votre projet).
Cliquez deux fois sur Native library location et choisissez l'option External Folder... : cette fois, vous devez indiquer dans quel répertoire se trouve sdljava ; c'est donc simple, il suffit d'indiquer le répertoire lib de votre projet.

Voilà ce que vous devez avoir après ces étapes :



L'installation est terminée ! Si, je vous assure. Nous allons exécuter un programme de test pour vérifier que tout fonctionne correctement. Si vous avez un souci avec le programme de test, recommencez l'installation : une chose vous a peut-être échappé.

Programme de test

Code : Java

```
package configuration;

import sdljava.SDLMain;
import sdljava.SDLException;
import sdljava.video.*;

public class Installation {

 private SDLSurface screen = null;

 public Installation() throws SDLException, InterruptedException {
  SDLMain.init(SDLMain.SDL_INIT_VIDEO);

  screen = SDLVideo.setVideoMode(640, 480, 32,
SDLVideo.SDL_DOUBLEBUF | SDLVideo.SDL_HWSURFACE);
  SDLVideo.wmSetCaption("Ma première fenêtre avec sdljava", null);

  Thread.sleep(1000);
  screen.freeSurface();
  SDLMain.quit();
 }

 public static void main(String[] args) throws SDLException,
InterruptedException {
  new Installation();
 }

}
```

N'essayez pas de comprendre ce code pour l'instant. Vous devez simplement voir une fenêtre s'ouvrir, puis se refermer et c'est normal ! En tout cas, toutes mes félicitations : la sdljava est opérationnelle. On va pouvoir passer aux choses sérieuses (enfin, on va rester simple quand même).
Dans le prochain chapitre, nous verrons comment créer des fenêtres et comment utiliser les modes vidéo...

À tout de suite !

Les fenêtres et les modes vidéo

Dans la vraie vie, quand vous parlez avec une personne vous commencez par lui dire "bonjour", ensuite vous parlez avec elle et enfin vous lui dites "au revoir" puis vous partez. Eh bien avec **SDL** c'est le même principe ! Pour travailler avec, vous devez l'initialiser (lui dire "bonjour"), puis à partir de ce moment vous avez le droit de travailler (parler avec elle), et quand vous avez fini, vous la quittez (vous lui dites "au revoir").

Initialisation de sdljava

Initialisation

On commence par importer les classes dont nous aurons besoin pour travailler. Pour l'instant trois suffiront, mais vous verrez qu'on va vite en importer encore et encore. 

Code : Java

```
import sdljava.SDLException;
import sdljava.SDLMain;
import sdljava.video.SDLVideo;
```

- La première va nous permettre de gérer les exceptions, car beaucoup de méthodes sont susceptibles de lever une exception !
- La deuxième va nous permettre d'initialiser SDL et ses composants, elle contient tout un paquet de constantes, ainsi que la méthode `init(constante)` ;
- La troisième nous permettra de créer une fenêtre (lui donner un titre, une icône, ...).

Les imports faits, nous pouvons commencer. On initialise **sdljava** avec la méthode statique `SDLMain.init(constante)` . Cette méthode prend en paramètres une ou plusieurs constantes, qui nous serviront à activer certains modules comme la vidéo ou le son par exemple. Voici quelques constantes :

- `SDLMain.SDL_INIT_VIDEO` : initialise le mode vidéo ;
- `SDLMain.SDL_INIT_AUDIO` : initialise le son ;
- `SDLMain.SDL_INIT_JOYSTICK` : initialise la prise en charge du joystick ;
- `SDLMain.SDL_INIT_CDROM` : initialise la prise en charge des lecteurs CD/DVD ;
- `SDLMain.SDL_INIT_EVERYTHING` : remplace TOUTES les constantes.

On pourra par exemple initialiser la vidéo et le son de cette manière :

Code : Java

```
SDLMain.init(SDLMain.SDL_INIT_VIDEO | SDL_INIT_AUDIO);
```

Le symbole | aussi appelé "pipe" permet de faire la séparation entre les constantes. Vous pouvez l'utiliser plusieurs fois pour charger plusieurs composants ; bien sûr si vous n'avez qu'une chose à initialiser vous n'aurez pas besoin de ce symbole. Si vous avez besoin de travailler avec tout (CDROM, AUDIO, VIDEO, etc.) vous pouvez tout initialiser d'un coup avec `SDL_INIT_EVERYTHING` , ce qui donne :

Code : Java

```
SDLMain.init(SDLMain.SDL_INIT_EVERYTHING);
```

Il y a encore d'autres constantes, et si cela vous intéresse (et je l'espère bien !), je vous invite à consulter la javadoc de sdljava (dossier **doc** vous vous souvenez ?).

Fermeture

Nous allons nous intéresser à la fermeture (le "au revoir" si vous préférez). Pour quitter SDL, c'est très simple, vous n'avez qu'à utiliser la méthode statique `quit()` de la classe `SDLMain` .

Code : Java

```
SDLMain.quit();
```

Ce code va libérer la mémoire et fermer SDL : vous ne pourrez donc plus travailler avec après cette étape.

Le programme final

Avant de vous montrer le programme final, il faut que je vous dise que les méthodes de sdljava peuvent lever des exceptions. Nous ne les traiterons pas directement au début, mais ça viendra et j'espère que vous êtes bien au point là-dessus, sinon faites un tour ici.

Code : Java

```
import sdljava.SDLException;
import sdljava.SDLMain;
import sdljava.video.SDLVideo;

public class ModeVideo {

 public static void main(String[] args) throws SDLException {

  // Initialisation de la SDL avec le mode video
  SDLMain.init(SDLMain.SDL_INIT_VIDEO);

  // Notre futur code !


  // On quitte la SDL et on libère la mémoire
  SDLMain.quit();
 }

}
```

Voilà qui est fait. Passons à la suite, je vois que vous en mourez d'envie.

Les surfaces : introduction

Nous avons créé une fenêtre noire avec un titre. Ce n'est pas encore super, donc nous allons mettre un peu de couleurs. Et pour cela nous devons étudier les `SDLSurface` !

Les surfaces avec SDL

Une surface est une forme géométrique sur laquelle on dessine des choses... Votre écran est une surface, la fenêtre que vous venez de créer est une surface (une `SDLSurface` en réalité). On peut donc conclure qu'une surface avec sdljava sera rectangulaire.

Elles vont nous servir à quoi tes `SDLSurface` ?

Pour faire simple, elle nous serviront à :

- afficher une fenêtre ;
- afficher une image ;
- afficher du texte.

De plus, on pourra noter que chaque `SDLSurface` aura des propriétés qui lui seront propres, comme :

- la taille ;
- la position ;
- la couleur de fond ;
- la couleur de transparence (canal Alpha) ;
- et d'autres...

Utilisation d'une SDLSurface

On doit importer la classe `SDLSurface` qui est dans le package `sdljava.SDLVideo` , puis créer un nouvel objet de ce type comme ceci :

Code : Java

```
SDLSurface screen = null;
```

J'ai volontairement nommé la surface **screen**, car nous allons l'utiliser pour l'affichage de la fenêtre. Ce que je ne vous avais pas dit, c'est que la méthode `SDLVideo.setVideoMode(...)` renvoie une SDLSurface, ce qui nous permet de travailler avec cette dernière par la suite (ajout de couleurs par exemple).

Voilà à quoi va ressembler notre nouveau code :

Code : Java

```
import sdljava.SDLException;
import sdljava.SDLMain;
import sdljava.video.SDLSurface;
import sdljava.video.SDLVideo;

public class ModeVideo {

 public static void main(String[] args) throws SDLException,
InterruptedException {

  SDLMain.init(SDLMain.SDL_INIT_VIDEO);

  /*
* Initialisation du mode vidéo
* 1 - screen est une SDLSurface qui ne fait référence à rien
* 2 - screen est maintenant la SDLSurface principale de l'écran :p
* - Elle contient les méthodes pour changer de couleur
* - Elle contient toutes les méthodes d'affichage (flippe de
l'écran par exemple)
* 3 - On change le titre de la fenêtre
*/
  SDLSurface screen = null;
  screen = SDLVideo.setVideoMode(400, 200, 32,
SDLVideo.SDL_HWSURFACE | SDLVideo.SDL_DOUBLEBUF);
  SDLVideo.wmSetCaption("Ma deuxième fenêtre en java avec SDL",
null);


  Thread.sleep(2000);
  SDLMain.quit();
 }

}
```

Le code est suffisamment commenté, et il n'a rien de complexe.

- On commence par créer une `SDLSurface` que l'on initialise à **`null`** ;
- La surface `screen` fait maintenant référence à la surface de l'affichage principal.

Vous pouvez aussi initialiser le mode vidéo comme ceci :

Code : Java

```
SDLSurface screen = SDLVideo.setVideoMode(320, 240, 32,
SDLVideo.SDL_HWSURFACE | SDLVideo.SDL_DOUBLEBUF);
```

Il nous reste un dernier point à aborder avant de nous amuser à colorier notre fenêtre : c'est la libération de la mémoire avec la méthode `freeSurface()` . En effet, lorsque vous avez fini d'utiliser une surface, il faut la détruire pour libérer la mémoire. Ici, c'est la surface qui correspond à la fenêtre principale. Vous me direz sûrement : "oui, mais quand le programme arrive à l'instruction `SDLMain.quit()` la mémoire est libérée !" ; eh bien c'est vrai, mais c'est une bonne habitude à prendre, surtout si vous travaillez avec une autre surface que celle de l'affichage. Nous n'avons qu'à rajouter `screen.freeSurface()` juste avant `SDLMain.quit()` et le tour est joué.

Maintenant que ce point est éclairci, nous allons donner un peu de couleurs à cette fenêtre toute triste.

Changer la couleur de fond

Nous utiliserons trois nouvelles méthodes, qui font partie de la surface screen. Les voici :

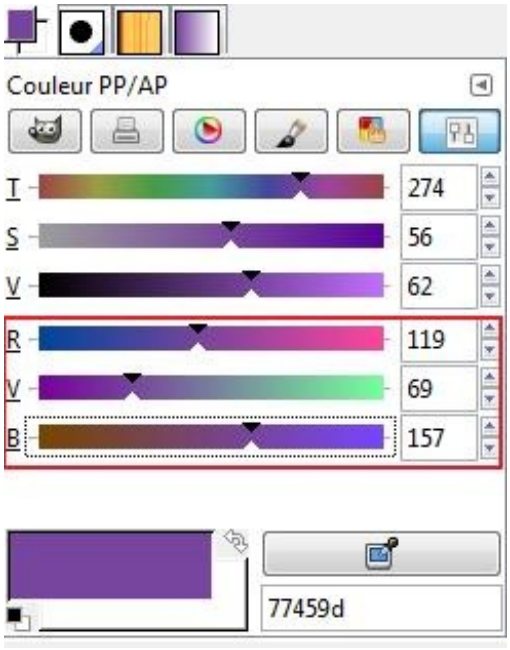
- `fillRect(long couleur)` ;
- `mapRGB(int rouge, int vert, int bleu)` ;
- `flip()` .

- La méthode `fillRect(long couleur)` permet de changer la couleur de fond de la surface qui l'appelle. Elle prend en paramètre un nombre (`long`), mais utiliser cette méthode comme cela n'est pas efficace car on ne sait pas quelle couleur correspond à quel nombre... C'est pour cela qu'on l'utilisera avec la prochaine méthode.
- La méthode `mapRGB(int rouge, int vert, int bleu)` renvoie un nombre de type `long` et prend en paramètres trois entiers (`int`) représentatifs des couleurs : rouge, vert et bleu. L'utilisation de la méthode précédente avec celle-ci sera donc particulièrement efficace !
- La méthode `flip()` permet d'appliquer les changements à l'écran. Cette méthode est donc à appeler à la fin de toutes nos manipulations.

Red, Green and Blue

La méthode `mapRGB()` prend en paramètres trois `int` comme nous l'avons vu. Ces valeurs varient entre 0 et 255. Par exemple, si vous voulez une couleur noire, il faut mettre toutes les valeurs à 0, car quand on les mélange, cela donne du noir ; l'inverse pour le blanc, c'est-à-dire 255 partout. Essayez donc de faire vos *"mix perso"*. Si vous ne comprenez pas, l'idéal, c'est de prendre le temps d'essayer chaque cas de figure ; vous verrez c'est vraiment tout bête à comprendre. Vous pouvez trouver une couleur avec son code RGB (red, green, blue) dans les logiciels d'édition d'images, comme *The Gimp* ou *Paint*.

Dans *The Gimp*, vous trouverez l'outil de couleurs dans le panneau de droite (à l'origine).



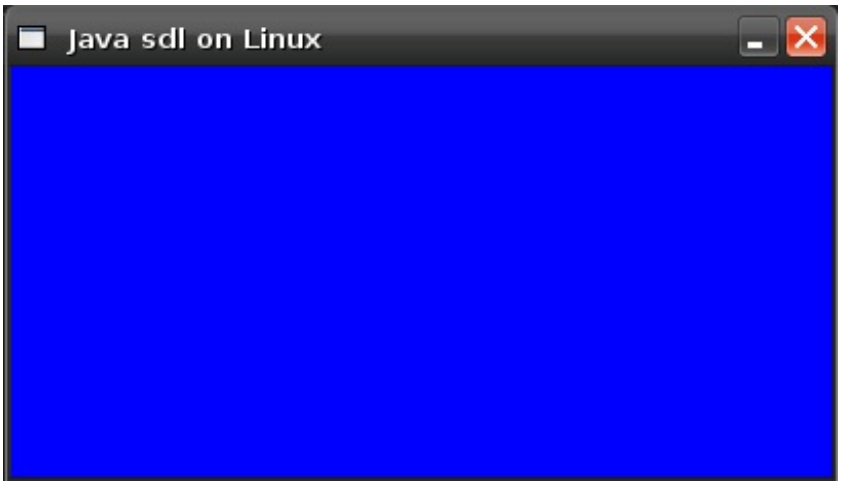
Vous voyez bien R pour Rouge, V pour Vert et B pour Bleu. C'est un bon moyen de se repérer et croyez-moi, nous aurons besoin de savoir avec quelle couleur travailler (quand on verra la transparence Alpha par exemple).

Voilà le code qui va changer la couleur de fond de votre fenêtre en... bleu !

Code : Java

```
import sdljava.SDLException;
import sdljava.SDLMain;
import sdljava.video.SDLSurface;
import sdljava.video.SDLVideo;

public class ModeVideo {

 public static void main(String[] args) throws SDLException,
InterruptedException {

  SDLMain.init(SDLMain.SDL_INIT_VIDEO);

  // Mode Video initialisé
  SDLSurface screen = SDLVideo.setVideoMode(400, 200, 32,
SDLVideo.SDL_HWSURFACE | SDLVideo.SDL_DOUBLEBUF);
  SDLVideo.wmSetCaption("Ma Troisième fenêtre en java avec SDL =)",
null);

  /*
* Changement de couleur
* 1 - Une variable couleur est créée et elle fait référence à la
couleur
* renvoyée par la méthode mapRGB()
* 2 - La méthode fillRect prend en paramètre la couleur que l'on
vient de définir
* 3 - On fait un "flip" de l'écran | On affiche les changements
*/
  long couleur = screen.mapRGB(0, 0, 255);
  screen.fillRect(couleur);
  screen.flip();

  Thread.sleep(2000);
                screen.freeSurface();
  SDLMain.quit();
 }

}
```

Et voilà le résultat :



S'il y a quelque chose qui vous échappe, prenez le temps de bien relire le tutoriel dans le calme. Sinon, vous pouvez passer à la suite.

Les modes vidéo

T'es bien gentil mais elle est où ma première fenêtre avec sdljava ?

On y arrive, ne vous en faites pas ! C'est d'ailleurs le but de cette partie : nous allons tout de suite passer à la création d'une fenêtre !

Pour créer une fenêtre, on utilise la classe `SDLVideo` avec la méthode `setVideoMode()` ; pour lui donner un titre, on utilise la méthode `wmSetCaption()` .

setVideoMode

`SDLVideo` contient plusieurs méthodes et plusieurs constantes. Nous les étudierons au fur et à mesure.

Pour initialiser la vidéo, nous ferons comme suit :

Code : Java

```
SDLVideo.setVideoMode(640, 480, 32, Constante1 | constante2 | ....);
```

Je vous détaille rapidement les trois premiers paramètres, puis les constantes.

- Le 1er paramètre est la largeur de l'écran en pixels ;
- Le 2ème paramètre est la hauteur de l'écran en pixels ;
- Le 3ème paramètre est la profondeur de l'écran (le nombre de couleurs).

Donc ici, nous sommes en 640x480 32 Bits, rien de bien compliqué à comprendre.

Les constantes

- `SDL_DISABLE` : cache le curseur de la souris ;
- `SDL_DOUBLEBUF` : active le *Double Buffering* ;
- `SDL_ENABLE` : affiche le curseur de la souris (actif d'origine) ;
- `SDL_FULLSCREEN` : affiche la fenêtre en plein écran ;
- `SDL_HWACCEL` : utilisation de l'accélération matérielle ;
- `SDL_HWSURFACE` : les surfaces sont stockées dans la mémoire vidéo (plus rapide) ;

- `SDL_OPENGL` : vous avez deviné non ? Initialisation de **OpenGL** ;
- `SDL_SWSURFACE` : les surfaces sont stockées dans la mémoire système (plus lent).

Il y a encore beaucoup d'autres constantes et encore une fois... consultez la doc. De toute façon, si vous ne le faites pas vous n'avancerez pas.
Généralement, on utilisera le *double buffering* et le *hardware surface* comme ceci :

Code : Java

```
SDLVideo.setVideoMode(640, 480, 32, SDLVideo.SDL_HWSURFACE |
SDLVideo.SDL_DOUBLEBUF);
```

C'est quoi le *Double Buffering* ?

En voilà une question intéressante !

C'est une technique utilisée principalement dans les jeux vidéo, pour éviter les scintillements de l'écran lorsqu'on affiche des éléments sur celui-ci. En gros c'est comme si vous aviez deux écrans (on parle en réalité de deux *buffers*) : votre écran physique, et un autre écran virtuel. L'écran virtuel dessine une image, puis la donne à l'écran physique. Pendant ce temps l'écran virtuel va redessiner une autre image (le temps que l'écran physique affiche la $1^{ère}$ image), puis quand l'écran physique aura terminé avec la $1^{ère}$ image, l'écran virtuel lui donnera la deuxième et ainsi de suite. Cela évite que l'écran physique soit seul à afficher les images, car il devrait alors préparer l'image puis l'afficher, puis l'effacer, puis préparer la deuxième, pour l'afficher... Vous comprenez l'avantage de cette technique ? Bien sûr pour afficher une fenêtre ça ne sert à rien, mais quand on affichera plusieurs images qui se déplaceront, cette technique nous montrera toute sa puissance !

Si vous voulez plus d'informations sur cette technique je vous conseille d'aller faire un petit tour ici.

wmSetCaption

On va maintenant donner un titre à notre fenêtre !

Code : Java

```
SDLVideo.wmSetCaption("Ma première fenêtre SDL en Java !", null);
```

Ce code permet de donner un titre à la fenêtre. Le premier paramètre est une chaîne de caractères (`String`) qui est le nom de la fenêtre, le deuxième paramètre est l'icône que vous voulez lui attribuer. Ici, nous ne voulons pas d'icônes, donc nous lui passons la référence `null`.

Le code final

Code : Java

```
import sdljava.SDLException;
import sdljava.SDLMain;
import sdljava.video.SDLVideo;

public class ModeVideo {

 public static void main(String[] args) throws SDLException,
InterruptedException {

  // Initialisation du mode vidéo
  SDLMain.init(SDLMain.SDL_INIT_VIDEO);
  /*
* Création d'une fenêtre de 320x240
```

```
        *Avec une profondeur de 32 bits
        * On utilise le mode Hardware Surface
        */
          SDLVideo.setVideoMode(320, 240, 32, SDLVideo.SDL_HWSURFACE);
          SDLVideo.wmSetCaption("Ma première fenêtre en java avec SDL",
       null);
          // On met en pause le Thread Principal pour que vous puissiez
       admirer votre fenêtre
          Thread.sleep(1000);
          // On quitte / Libération de la mémoire
          SDLMain.quit();
         }
       }
```

Et voilà, c'est fait ! Elle est pas belle la ~~vie~~ fenêtre ?

Exercice : changement de couleurs dynamique

Il le faut et c'est pour votre bien. Je vais vous mettre en garde maintenant : si vous ne lisez pas la doc et si vous ne pratiquez pas (ou peu), vous ne développerez pas de jeux ou d'applications qui vous rendront fiers. sdljava est une *librairy* simple et accessible et on programme en Java, alors profitez-en, ne vous limitez pas à ce tutoriel.

Le sujet : je vous demande de créer une fenêtre de la taille que vous voulez, dans laquelle nous changerons la couleur de fond 5 fois de suite.

- 1ère fois : le fond est noir ;
- 2ème fois : le fond est rouge ;
- 3ème fois : le fond est vert ;
- 4ème fois : le fond est bleu ;
- 5ème fois : le fond est blanc.

Après ça, le programme se termine normalement. Pour réaliser cet exercice, vous avez plusieurs solutions (structures répétitives, alternatives, ...). Le but recherché est le suivant : la fenêtre affiche un fond noir, puis au bout d'un moment (que je vous laisse choisir), la couleur change et ainsi de suite. N'oubliez pas que nos amis les `Thread` sont là ! (une seul méthode de la classe `Thread` suffit, et elle ne devrait pas vous être méconnue, ... enfin je l'espère).

En avant la correction ! Qui, bien entendu, peut varier selon les implémentations de chacun. Ce n'est donc pas LA correction absolue, c'est UNE correction qui fonctionne.

Secret (cliquez pour afficher)

Code : Java

```
import sdljava.SDLException;
import sdljava.SDLMain;
import sdljava.video.SDLSurface;
import sdljava.video.SDLVideo;

public class ModeVideo {

 public static void main(String[] args) throws SDLException,
InterruptedException {

  int compteur = 0; // Variable de boucle
  /* Initialisation de SDL et de la Vidéo */
  SDLMain.init(SDLMain.SDL_INIT_VIDEO);
  SDLSurface screen = SDLVideo.setVideoMode(400, 200, 32,
SDLVideo.SDL_HWSURFACE | SDLVideo.SDL_DOUBLEBUF);
  SDLVideo.wmSetCaption("Ma Quatrième fenêtre en java avec SDL
=)", null);

  while(compteur <= 4) {
```

```
            /* On teste chaque valeur de compteur et on agit en
        conséquence */
            switch( compteur ) {
             case 0: screen.fillRect(screen.mapRGB(0, 0, 0)); // Noir
              break;
             case 1: screen.fillRect(screen.mapRGB(255, 0, 0)); // Rouge
              break;
             case 2: screen.fillRect(screen.mapRGB(0, 255, 0)); // Vert
              break;
             case 3: screen.fillRect(screen.mapRGB(0, 0, 255)); // Bleu
              break;
             case 4: screen.fillRect(screen.mapRGB(255, 255, 255)); //
        Blanc
              break;
            }

            screen.flip();
            Thread.sleep(1500);
            compteur++;
          }
          SDLMain.quit();
         }
        }
```

Vous avez appris pas mal de choses, et il est peut-être temps de faire le point et de vous reposer un moment. Allez prendre un café ou une pizza (je m'en fiche) et revenez quand vous serez prêts, car nous attaquons les deux derniers chapitres de base et il faut que vous soyez motivés (il n'y a rien de compliqué si ça peut vous rassurer).

Les surfaces

Résumons un peu :

- Vous avez installé sdljava ;
- Vous savez créer une fenêtre ;
- Vous savez changer la couleur de fond de cette dernière ;
- Vous savez lui ajouter un titre ;
- Vous connaissez les différents modes vidéo.

Je vous ai fait une brève introduction sur les `SDLSurface` et nous allons maintenant les voir de plus près (près comment monsieur ? T'inquiètes pas pour ça bonhomme). Dans un premier temps, nous allons en créer une nouvelle, puis la placer où nous le voulons sur la fenêtre, puis nous en placerons plusieurs. Une fois ce mécanisme compris, vous serez prêt à lire le chapitre de cette partie 1. Allez, on est parti.

Votre première SDLSurface

En réalité, ce ne sera pas votre première `SDLSurface`, puisque nous avons eu l'occasion d'en avoir un bref aperçu dans le chapitre précédent. Dans ce chapitre, nous allons utiliser plusieurs surfaces dans la fenêtre ; cela consistera au début à avoir un rectangle d'une couleur rouge par exemple dans la fenêtre, puis d'en avoir deux, trois... vous me suivez ? Nous allons aussi voir comment les positionner en fin de chapitre. Vous comprendrez aussi l'utilité de la constante `SDLVideo.SDL_HWSURFACE` qui nous servira donc à utiliser la mémoire de la carte graphique plutôt que la mémoire système, ce sera bien plus rapide.

Les SDLSurface

Une surface, dans notre cas, est une forme géométrique en 2 dimensions de forme rectangulaire. On peut en créer à l'aide de `SDLSurface maSurface`, mais bien sûr une fois créée, il faut bien qu'elle fasse référence à quelque chose, car dans le chapitre précédent, la surface que nous avons créée faisait référence à l'écran. Ici on utilisera la méthode `SDLVideo.createRGBSurface(...)`. Cette dernière prend 8 paramètres, mais ne vous en faites pas c'est très simple, les voici :

- Le mode vidéo. Dans notre cas : hardware ;
- La largeur de la surface ;
- La hauteur de la surface ;
- Le nombre de couleurs : on utilisera 32 bits comme pour la fenêtre ;
- Quatre paramètres de type `long` qui correspondent aux couleurs RGBA de mask. Pour des raisons de simplicité, nous les mettrons tous à 0.

On pourra donc créer une surface comme suit :

Code : Java

```
rectangle = SDLVideo.createRGBSurface(SDLVideo.SDL_HWSURFACE, 200,
150, 32, 0, 0, 0, 0);
```

Ici nous initialisons notre `SDLSurface` pour qu'elle soit utilisée dans la mémoire de la carte graphique, ensuite nous définissons les paramètres de largeur, de hauteur ainsi que de profondeur (nombre de couleurs). Les quatre derniers paramètres sont à 0 comme indiqué plus haut.

Le problème est qu'en écrivant juste ce code, ça ne va pas fonctionner ! Si vous touchez un peu à AWT/Swing, vous savez que quand on crée un `JButton`, il faut dans un premier temps le créer, puis l'attacher à la fenêtre, puis demander à la fenêtre de s'afficher. Eh bien une surface, c'est exactement pareil ! Et pour cela, on utilise la méthode `blitSurface(SDLSurface parent)` où `parent` est une `SDLSurface` et en l'occurrence la fenêtre principale, mais rien ne vous empêche de bliter une surface sur une autre.

Voici un code avec **Swing** pour que vous puissiez comparer :

Code : Java

```
// votre code d'initialisation
....
// On fabrique un bouton
JButton monBouton = new JButton("bonjour");
// On attache le bouton sur la zone d'affichage
getContentPane().add(monBouton);
// On affiche la fenêtre
setVisible(true);
```

Et le code en SDL :

Code : Java

```
// Code d'initialisation SDL, création des SDLSuface, etc...
....
// On crée un nouveau rectangle
rectangle = SDLVideo.createRGBSurface(SDLVideo.SDL_HWSURFACE, 200,
150, 32, 0, 0, 0, 0);
// On attache le rectangle sur la fenêtre (qui est une SDLSurface
du doux nom de screen)
rectangle.blitSurface(screen);
// On affiche le tout ! C'est la fenêtre qui contient la surface
rectangle, donc c'est elle qu'on flip
screen.flip();
```

Si vous rajoutez ce code comme ça, vous ne verrez rien et savez-vous pourquoi ? Vous vous souvenez de la couleur de fond d'origine de la fenêtre ? Elle est noire. Et vous souvenez-vous que la fenêtre est une `SDLSurface` ? Vous en avez donc sûrement déduit que notre surface rectangle sera, elle aussi, noire. Pour remédier à ce problème, nous pouvons changer la couleur exactement de la même manière que pour l'écran, c'est-à-dire avec les méthodes `fillRect(...)` et `mapRGB(...)` .

Ce qui va nous donner quelque chose comme cela (avouez ! Vous avez senti venir l'exercice hein).

Code : Java

```
import sdljava.SDLException;
import sdljava.SDLMain;
import sdljava.video.SDLSurface;
import sdljava.video.SDLVideo;

public class Surface {
 public static void main(String[] args) throws SDLException,
InterruptedException {

  SDLSurface screen = null,
       rectangle = null;

  SDLMain.init(SDLMain.SDL_INIT_VIDEO);
  screen = SDLVideo.setVideoMode(640, 480, 32,
SDLVideo.SDL_DOUBLEBUF | SDLVideo.SDL_HWSURFACE);
  SDLVideo.wmSetCaption("Les SDLSurface avec SDL en Java", null);

  rectangle = SDLVideo.createRGBSurface(SDLVideo.SDL_HWSURFACE, 200,
150, 32, 0, 0, 0, 0);
  rectangle.fillRect(rectangle.mapRGB(255, 0, 0));
  rectangle.blitSurface(screen);
  screen.flip();

  Thread.sleep(2000);
  rectangle.freeSurface();
                screen.freeSurface();
  SDLMain.quit();
```

```
    }
}
```

Positionner une surface

Comme vous avez pu le constater, la surface que nous avons créée s'est positionnée dans le coin en haut à gauche, mais comment faire pour la positionner en plein milieu de l'écran par exemple ? Ou ailleurs même ? On utilisera une nouvelle classe : `SDLRect` . Nous allons procéder par étapes comme nous en avons pris l'habitude.

1. Importer la classe `SDLRect` :

Code : Java

```
import sdljava.video.SDLRect;
```

2. Créer un objet de ce type :

Code : Java

```
SDLRect posRectangle = new SDLRect();

// Ou bien

SDLRect posRectangle = new SDLRect(int x, int y);
```

Vous remarquez qu'on peut initialiser notre objet de deux manières :

- En utilisant un constructeur sans paramètres ;
- En utilisant un constructeur avec paramètres.

Le deuxième constructeur prend en paramètres deux `int` qui représentent les coordonnées en X et en Y. Dans le cas du premier constructeur, il faudra, à l'aide des méthodes `setX(int x)` et `setY(int y)` , fournir ces valeurs.

Voilà par exemple le code pour centrer une surface à l'écran :

Code : Java

```
// Taille de la fenêtre
int width = 640, height = 480;
// Taille de la surface rectangle
int rectX = 200, rectY = 150;
// Position du rectangle centré
SDLRect posRectangle = new SDLRect();
posRectangle.setX((width / 2) - (rectX / 2));
posRectangle.setY((height / 2 ) - (rectY / 2));

/*
* On pourrait aussi faire comme ça :)
*
* SDLRect posRectangle = new SDLRect(((width / 2) - (rectX / 2)),
((height / 2 ) - (rectY / 2)));
*/
```

Si vous écrivez ça, encore une fois ça ne fonctionne pas, et je sais que vous savez pourquoi. Il faut maintenant lier notre

`SDLSurface` rectangle avec notre `SDLRect` posRectangle. On utilisera pour cela une méthode que vous connaissez (si si je vous assure), et d'ailleurs la voilà :

Code : Java

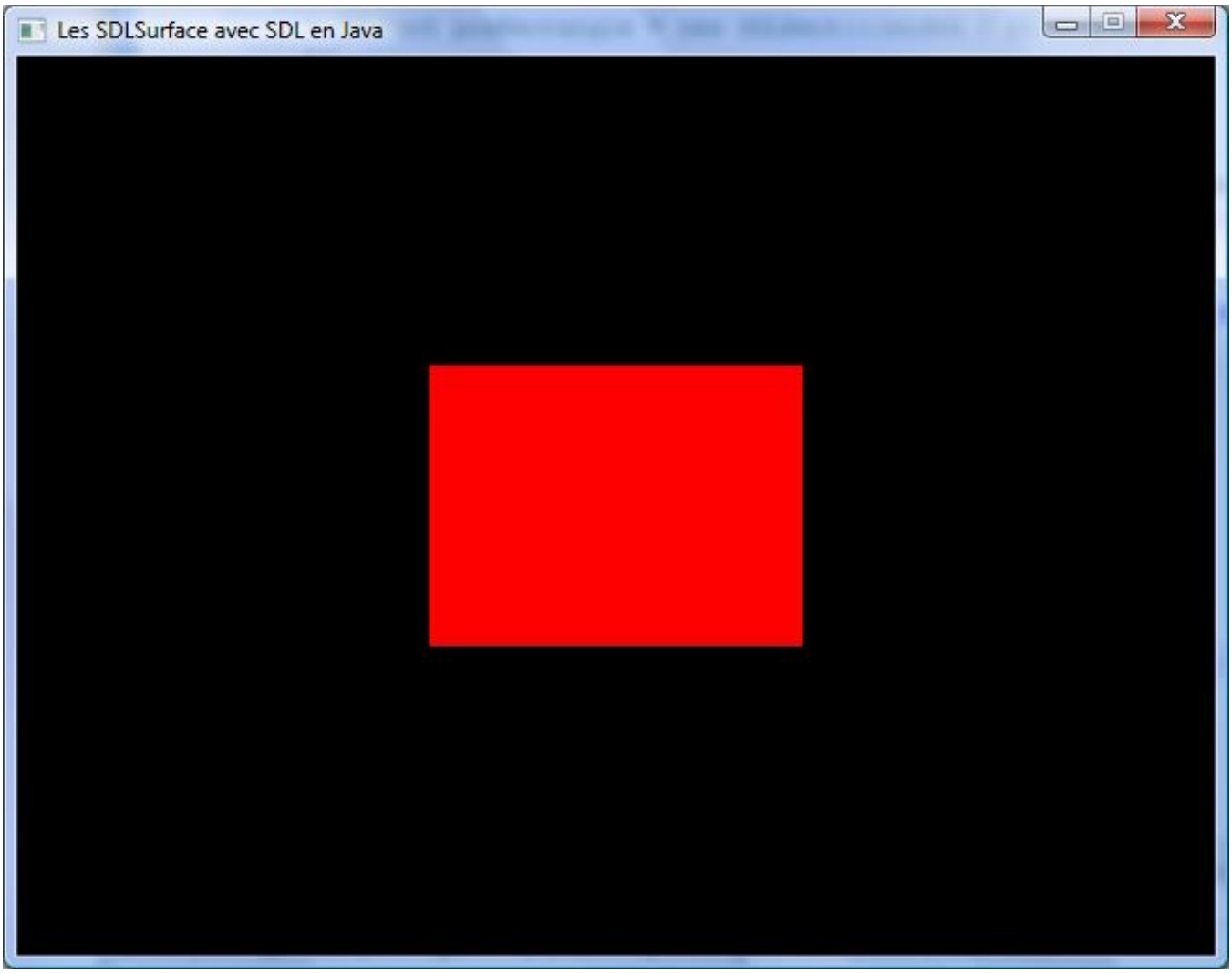
```
blitSurface(SDLSurface maSurface, SDLRect positionMaSurface);
```

Comme vous pouvez le voir, la méthode **blitSurface** peut prendre plusieurs paramètres. Je ne pense pas que vous ayez besoin de plus d'explications, voilà le code qui va centrer notre surface :

Code : Java

```
import sdljava.SDLException;
import sdljava.SDLMain;
import sdljava.video.SDLRect;
import sdljava.video.SDLSurface;
import sdljava.video.SDLVideo;

class Surface {
 public static void main(String[] args) throws SDLException,
InterruptedException {

  // Taille de la fenêtre
  int width = 640, height = 480;
  // Taille de la surface rectangle
  int rectX = 200, rectY = 150;
  // Position du rectangle centré
  SDLRect posRectangle = new SDLRect();
  posRectangle.setX((width / 2) - (rectX / 2));
  posRectangle.setY((height / 2 ) - (rectY / 2));

  /*
* On pourrait aussi faire comme ça :)
*
* SDLRect posRectangle = new SDLRect(((width / 2) - (rectX / 2)),
((height / 2 ) - (rectY / 2)));
*/

   // Initialisation SDL et Vidéo
  SDLMain.init(SDLMain.SDL_INIT_VIDEO);
  SDLSurface screen = SDLVideo.setVideoMode(width, height, 32,
SDLVideo.SDL_DOUBLEBUF | SDLVideo.SDL_HWSURFACE);
  SDLVideo.wmSetCaption("Les SDLSurface avec SDL en Java", null);

  // Création d'une SDLSurface rectangle
  SDLSurface rectangle =
SDLVideo.createRGBSurface(SDLVideo.SDL_HWSURFACE,
    rectX, rectY, 32, 0, 0, 0, 0);
  // Changement de couleur en rouge
  rectangle.fillRect(rectangle.mapRGB(255, 0, 0));
  // On blit sur l'écran la surface rectangle avec ses coordonnées
(via SDLRect)
  rectangle.blitSurface(screen, posRectangle);
  screen.flip();

  Thread.sleep(2000);

  rectangle.freeSurface();
  screen.freeSurface();
  SDLMain.quit();

 }
}
```

Et voilà le travail !



Faites des tests et essayez de changer les valeurs de `SDLRect` pour voir.

Gérer plusieurs surfaces

Nous allons voir très rapidement comment utiliser plusieurs `SDLSurface` dans notre programme.

Nous avons déjà créé deux `SDLSurface` : la `SDLSurface` qui correspond à l'écran et la `SDLSurface` qui correspond au rectangle. Maintenant pour en créer d'autres, c'est très simple : il faut faire pareil que pour le rectangle, c'est-à-dire :

- Créer une `SDLSurface` ;
- Positionner cette `SDLSurface` avec `SDLRect` ;
- Donner une couleur ;
- Bliter cette dernière sur la surface écran.

Petit exercice

Vous allez créer un programme qui affiche 4 `SDLSurface` (carrés ou rectangles comme vous voulez).

Prenez votre temps pour faire cet exercice (bien qu'il ne soit vraiment pas dur). S'il le faut, utilisez une feuille de papier et un crayon (ou un stylo c'est vous qui voyez) et dessinez ce qui doit être affiché, cela vous aidera beaucoup croyez-moi.

Allez voilà la correction, j'espère que vous avez réussi.

Version 1 : Les mauvaises habitudes.

Code : Java

```
import sdljava.SDLException;
import sdljava.SDLMain;
import sdljava.video.SDLRect;
import sdljava.video.SDLSurface;
import sdljava.video.SDLVideo;

class Surface {
 public static void main(String[] args) throws SDLException,
InterruptedException {

  // Taille de la fenêtre
  int width = 320, height = 240;

  // Taille des surfaces
  int carre1X = 60, carre1Y = 60,
   carre2X = 60, carre2Y = 60,
   carre3X = 60, carre3Y = 60,
   carre4X = 60, carre4Y = 60;

  // Position des surfaces
  SDLRect posCarre1 = new SDLRect(30, 30);
  SDLRect posCarre2 = new SDLRect(200, 30);
  SDLRect posCarre3 = new SDLRect(30, 120);
  SDLRect posCarre4 = new SDLRect(200, 120);


   // Initialisation SDL et Vidéo
  SDLMain.init(SDLMain.SDL_INIT_VIDEO);
  SDLSurface screen = SDLVideo.setVideoMode(width, height, 32,
SDLVideo.SDL_DOUBLEBUF | SDLVideo.SDL_HWSURFACE);
  SDLVideo.wmSetCaption("4 SDLSurface !", null);

  // Création des surfaces
  SDLSurface carre1 =
SDLVideo.createRGBSurface(SDLVideo.SDL_HWSURFACE, carre1X, carre1Y,
32, 0, 0, 0, 0);
  SDLSurface carre2 =
SDLVideo.createRGBSurface(SDLVideo.SDL_HWSURFACE, carre2X, carre2Y,
32, 0, 0, 0, 0);
  SDLSurface carre3 =
SDLVideo.createRGBSurface(SDLVideo.SDL_HWSURFACE, carre3X, carre3Y,
32, 0, 0, 0, 0);
  SDLSurface carre4 =
SDLVideo.createRGBSurface(SDLVideo.SDL_HWSURFACE, carre4X, carre4Y,
32, 0, 0, 0, 0);

  // On donne des couleurs différentes à chaques surfaces
  screen.fillRect(screen.mapRGB(0, 45, 78));
  carre1.fillRect(carre1.mapRGB(255, 125, 135));
  carre2.fillRect(carre2.mapRGB(0, 89, 32));
  carre3.fillRect(carre3.mapRGB(25, 20, 65));
  carre4.fillRect(carre4.mapRGB(55, 240, 12));

  // On blit tout ça sur l'écran :)
  carre1.blitSurface(screen, posCarre1);
  carre2.blitSurface(screen, posCarre2);
  carre3.blitSurface(screen, posCarre3);
  carre4.blitSurface(screen, posCarre4);

  // On flip et on regarde ce que ça donne ;)
  screen.flip();

  Thread.sleep(5000);

  // On quitte
```

```
    carre1.freeSurface();
    carre2.freeSurface();
    carre3.freeSurface();
    carre4.freeSurface();
    screen.freeSurface();
    SDLMain.quit();

   }
  }
```

Ce code est moche, on pourrait utiliser des tableaux pour stocker les surfaces, les positions et les couleurs.

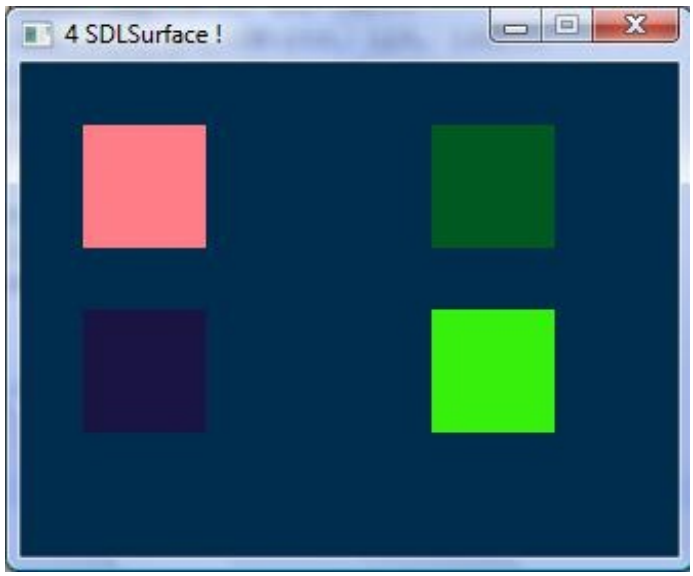
Version 2 : un code plus "propre" et plus court !

Code : Java

```
import sdljava.SDLException;
import sdljava.SDLMain;
import sdljava.video.SDLRect;
import sdljava.video.SDLSurface;
import sdljava.video.SDLVideo;

class Surface {
 public static void main(String[] args) throws SDLException,
InterruptedException {

  // Taille de la fenêtre
  int width = 320, height = 240;


  /*
* Positions des surfaces
*
* Les surfaces n'ont pas la même position, on remplit donc à la
main les positions
* avec les méthode setX(int x) et setY(int y)
*/

  SDLRect posCarre[] = new SDLRect[4];
  posCarre[0] = new SDLRect(30 , 30);
  posCarre[1] = new SDLRect(200, 30);
  posCarre[2] = new SDLRect(30, 120);
  posCarre[3] = new SDLRect(200, 120);


   // Initialisation SDL et Vidéo
  SDLMain.init(SDLMain.SDL_INIT_VIDEO);
  SDLSurface screen = SDLVideo.setVideoMode(width, height, 32,
SDLVideo.SDL_DOUBLEBUF | SDLVideo.SDL_HWSURFACE);
  SDLVideo.wmSetCaption("4 SDLSurface !", null);

  // Création des surfaces
  SDLSurface carre[] = new SDLSurface[4];
  for (int i = 0 ; i < 4 ; i++) {
   carre[i] = SDLVideo.createRGBSurface(SDLVideo.SDL_HWSURFACE, 60,
60, 32, 0, 0, 0, 0);
  }

  /*
* On donne des couleurs pour chaques surfaces
*
* Vue que chaque surface a des couleurs différentes, on n'utilise
pas de boucle for
* On peut utiliser une boucle for pour qu'elles soient toutes de la
même couleur
* ou en utilisant des variables R, G, B que l'on incrémente à
```

```
        chaque passage de boucle
        */
          screen.fillRect(screen.mapRGB(0, 45, 78));
          carre[0].fillRect(carre[0].mapRGB(255, 125, 135));
          carre[1].fillRect(carre[1].mapRGB(0, 89, 32));
          carre[2].fillRect(carre[2].mapRGB(25, 20, 65));
          carre[3].fillRect(carre[3].mapRGB(55, 240, 12));

          // On blit tout ça sur l'écran :)
          for (int i = 0 ; i < 4 ; i++) {
           carre[i].blitSurface(screen, posCarre[i]);
          }

          // On flip et on regarde ce que ça donne ;)
          screen.flip();

          Thread.sleep(5000);

          // On quitte
          for (int i = 0 ; i < 4 ; i++) {
           carre[i].freeSurface();
          }

          screen.freeSurface();
          SDLMain.quit();

         }
        }
```

Et voilà le travail !



Lisez bien les commentaires pour comprendre ce que j'ai fait. Comme vous pouvez le constater, ce code est plus clair, plus light et pourtant, il fait exactement la même chose ! L'utilisation des tableaux est très importante si vous voulez créer des jeux (que ce soit en 2D ou en 3D) car ils permettent d'avoir un code lisible et ils sont à la base de beaucoup de choses comme les sprites, les maps, etc.
Si ce que vous avez vu vous a paru compliqué, je vous conseille vivement de pratiquer et de relire cette partie. Sinon, je vous invite à passer au dernier chapitre. Un chapitre concernant les images ! Mais je ne vous en dis pas plus, à tout de suite.

Les images

Nous allons maintenant nous intéresser au chargement d'images et à leur manipulation, nous verrons :

- le chargement,
- la couleur de transparence (canal Alpha),
- comment rendre l'image transparente.

Une image n'est qu'une `SDLSurface` sans couleur sur laquelle on à posé une image (je simplifie), donc ce chapitre ne sera pas très long, mais sera riche en nouvelles connaissances !

Je ferai aussi une brève présentation des événements avec SDL, ce qui nous évitera d'avoir recours à la méthode `sleep()` de la classe `Thread` . Mais avant toute chose, nous devrons parler de "l'installation" de SDL_image.

Installation et configuration de SDL_image

Installation sous Linux

Amis linuxiens, nous avons du travail ! En effet par rapport à l'installation sous Windows, nous aurons quelques manipulations de plus à effectuer, mais ne vous en faites pas ce n'est pas très compliqué.

Mis à part le fichier libSDL_image.a nous aurons besoin d'autres bibliothèques qui doivent normalement être installées ; si ce n'est pas le cas, il faudra aller télécharger les sources et les installer. Vous devez donc avoir ces bibliothèques :

- libjpeg,
- libpng,
- libtiff,
- zlib.

Regardez plus bas pour savoir si vous les avez.

Si la bibliothèque SDL ainsi que ses extensions sont déjà installées sur votre machine

Allez dans le répertoire `/usr/lib` et recherchez les fichiers suivants :

- libjpeg.a ;
- libpng12.a ;
- libSDL_image.a ;
- libtiff.a ;
- libz.a.

Si vous avez compilé SDL_image à la main, il est fort probable que le fichier libSDL_image.a se trouve dans `/usr/local/lib` . Une fois que vous avez ces fichiers copiez-les simplement à la racine de votre répertoire de projet (à coté de libSDLmain.a).

Voila c'est fini. Profitez-en donc pour créer un nouveau dossier que vous nommerez par exemple images. Il contiendra les images que nous utiliserons dans 5 minutes.

Si la bibliothèque ainsi que ses extensions ne sont pas installées sur votre machine

Allez dans le répertoire `/usr/lib` et recherchez les fichiers suivants :

- libjpeg.a ;
- libpng12.a ;
- libtiff.a ;
- libz.a.

Une fois que vous les avez, copiez-les à la racine de votre répertoire projet (à coté de libSDLmain.a). On peut passer à la suite. Rendez-vous sur le site de SDL_image pour y télécharger les sources. Décompressez les sources dans un dossier approprié (personnellement j'utilise `$HOME/Sources`), puis ouvrez un terminal et compilez la bibliothèque comme ceci :

Sur les distributions type Mandriva, Fedora, OpenSuse, etc.

Code : Bash

```
./configure
make
su
#Entrez votre mot de passe
make install
```

Sur les distributions type Debian, Ubuntu, etc.

Code : Bash

```
./configure
make
sudo make install
# Entrez votre mot de passe
```

Rendez-vous ensuite dans `/usr/local/lib` et récupérez-y le fichier libSDL_image.a et enfin copiez-le avec les autres bibliothèques, à la racine de votre répertoire de projet. Je vous invite à créer un dossier images, qui contiendra les images que nous utiliserons dans 5 minutes.

Voilà à quoi doit ressembler votre répertoire :

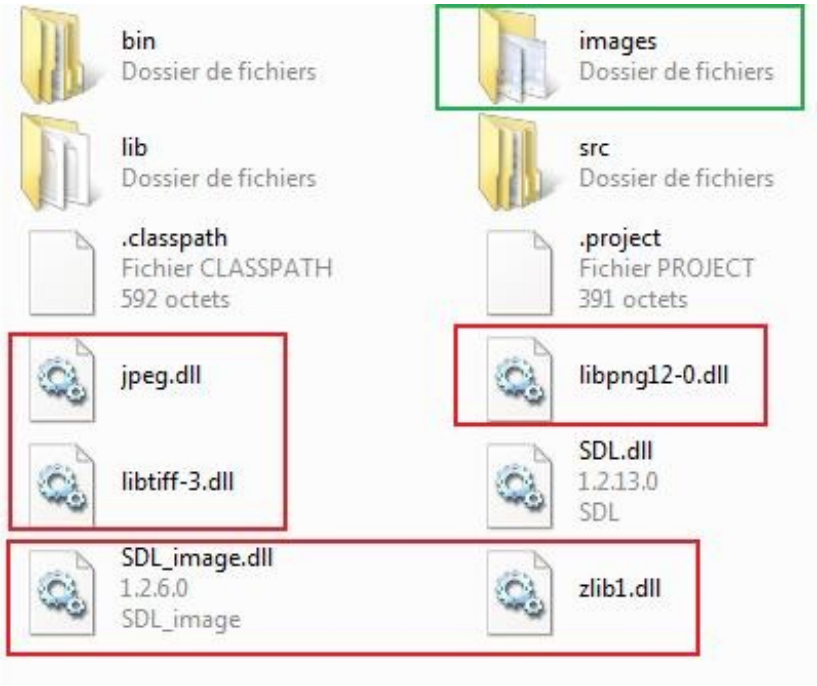
Suivant votre distribution et le type d'installation que vous avez fait de la bibliothèque SDLJava, vous n'aurez besoin que de libSDL_image.a

Installation sous Windows

Comme sous Linux vous aurez besoin de plusieurs fichiers dll, la bonne nouvelle est que l'on va récupérer une archive qui contient toutes ces dll. Vous devez vous rendre sur le site de SDL_image pour y télécharger SDL_image-1.2.xx-win32.zip. Une fois cette archive téléchargée, vous devez décompresser son contenu (plusieurs fichiers .dll) à la racine de votre répertoire projet (à coté de SDL.dll).

L'installation est terminée, profitez-en pour créer un nouveau dossier que vous pourrez nommer images, qui contiendra les images que nous utiliserons dans 5 minutes.

Voilà à quoi doit ressembler votre répertoire projet (en vert le nouveau dossier images, et en rouge les nouveaux fichiers dll) :



Charger une image

Pour charger une image c'est facile, vous vous souvenez des surfaces ? Eh bien figurez-vous qu'une image sera vue par la SDL comme une `SDLSurface` , ce qui veut dire que pour charger et afficher une image on a juste une méthode qui change !

Tout d'abord nous devons importer la classe `SDLImage` comme nous en avons pris l'habitude :

Code : Java

```
import sdljava.image.SDLImage;
```

Ensuite il nous faut une image, oui mais quel format ? Eh bien je risque peut-être d'en décevoir certains, mais les formats jpeg et tif sont très mal pris en charge, nous pourrons par contre utiliser des images aux formats gif, png et bmp. Pour notre exemple je prendrai une image de 640x480 au format png. Cette image devra se trouver dans le répertoire images que nous avons créé plus haut.

Nous devons maintenant créer une nouvelle surface que l'on nommera image, mais à la différence d'une surface classique, nous l'initialiserons avec la méthode `load("images/monImage.png");` de la classe `SDLImage` .

Une fois l'initialisation faite, c'est exactement comme une `SDLSurface` , donc on blit et on flip.

Mais attend ! Et la taille de l'image on la définit pas ?

Eh bien non, la méthode qui charge l'image va créer une `SDLSurface` de la taille de l'image, ce qui est très pratique, donc pas besoin de se soucier de la taille de l'image (juste le format).

Voici un code qui charge une image et qui l'affiche sur l'écran, par contre j'ai volontairement enlevé `Thread.sleep()` pour le remplacer par un gestionnaire d'événements, que nous verrons dans le prochain chapitre. Ne prenez donc pas peur si vous ne comprenez pas (bien que ça ne soit pas compliqué du tout), mais en gros ce code permet de laisser la fenêtre ouverte tant que vous ne cliquez pas sur la croix dans la barre de tâche. Voilà tout est dit, place au code.

Code : Java

```
import sdljava.SDLException;
import sdljava.SDLMain;
import sdljava.event.SDLEvent;
import sdljava.video.SDLRect;
import sdljava.video.SDLSurface;
import sdljava.video.SDLVideo;
import sdljava.image.SDLImage;
```

```
public class Images {

 public static void main(String[] args) throws SDLException {

  // Gestion des evenements 1/2
  SDLEvent event = null;
  boolean boucle = true;

  // Initialisation de SDL et de l'affichage
  SDLMain.init(SDLMain.SDL_INIT_VIDEO);
  SDLVideo.wmSetCaption("Des images avec sdljava", null);
  SDLSurface screen = SDLVideo.setVideoMode(640, 480, 32,
SDLVideo.SDL_DOUBLEBUF | SDLVideo.SDL_HWSURFACE);

  // Chargement d'une image
  SDLSurface image = SDLImage.load("images/aerith.png");
  SDLRect imagePos = new SDLRect(0, 0);


  /*
* Gestion des événement 2/2
*
* Boucle Principale
*
* Le code est exécuté TANT QUE la variable booléene boucle est vrai
* SI l'événement SDL_QUIT est vrai ALORS la variable boucle est
fausse et donc on sort de la boucle
*/
  while (boucle)
  {
   event = SDLEvent.waitEvent();
   if (event.getType() == SDLEvent.SDL_QUIT)
    boucle = false;

   image.blitSurface(screen, imagePos);
   screen.flip();
  }

  // On quitte :: Libération de la mémoire
  image.freeSurface();
  screen.freeSurface();
  SDLMain.quit();
 }

}
```

Tadaaa, elle est pas belle ~~Aeris~~ la vie ?

Un petit mot sur les événements

Bon, c'est bien parce que c'est vous, on commence par créer un objet de type `SDLEvent` (bien qu'en réalité on dirait qu'on crée un objet qui fait référence à un objet de type `SDLEvent`) et une variable booléenne qui vaut VRAI. L'objet `event` une fois initialisé nous informera des événements reçus par le programme. La variable `boucle` permet de lancer la boucle principale du programme (nous reviendrons sur la "boucle principale" dans le prochain chapitre), on aurait aussi pu écrire **`while`**`(`**`true`**`)` et quitter la boucle avec une instruction **`break`** , mais c'est plus "sale".

L'initialisation de l'objet `event` se fait grâce à la méthode `waitEvent()` de la classe `SDLEvent` (nous verrons qu'il existe plusieurs types d'événements). `waitEvent()` va mettre le programme en pause tant qu'il n'y a pas d'événements.

On teste ensuite l'événement (on utilisera des **`switch`** dans le prochain chapitre, car ils seront plus adaptés). `SDLEvent.SDL_QUIT` est une constante qui correspond au clic sur la croix de fermeture de la fenêtre.

Ensuite on blit l'image et on flip l'écran.

Pourquoi on blit et on flip dans la boucle ?

Car si on blit ou flip après la boucle, le programme se termine donc on ne voit rien et de plus quand on déplacera des surfaces, il faudra que ce soit à chaque passage de boucle, je vous montre donc comment on fait dès le début !

Ajout d'une autre image

C'est exactement pareil que d'afficher plusieurs `SDLSurface` sur la fenêtre, on charge, on donne les coordonnées, on blit et on flip. Bon pour vous montrer je vais ajouter une personne qui tiendra compagnie à Aeris...

Code : Java

```
public class Images {

 public static void main(String[] args) throws SDLException {

  // Gestion des événements 1/2
  SDLEvent event = null;
  boolean boucle = true;

  // Initialisation de SDL et de l'affichage
  SDLMain.init(SDLMain.SDL_INIT_VIDEO);
  SDLVideo.wmSetCaption("Des images avec sdljava", null);
  SDLSurface screen = SDLVideo.setVideoMode(640, 480, 32,
SDLVideo.SDL_DOUBLEBUF | SDLVideo.SDL_HWSURFACE);

  // Chargement des images
  SDLSurface aeris = SDLImage.load("images/aerith.png");
  SDLSurface zack = SDLImage.load("images/zack.png");
  SDLRect aerisPos = new SDLRect(0, 0);
  SDLRect zackPos = new SDLRect(300, 50);


  while (boucle)
  {
   event = SDLEvent.waitEvent();
   if (event.getType() == SDLEvent.SDL_QUIT)
    boucle = false;

   aeris.blitSurface(screen, aerisPos);
   zack.blitSurface(screen, zackPos);
   screen.flip();
  }

  // On quitte :: Libération de la mémoire
```

```
            aeris.freeSurface();
            zack.freeSurface();
            screen.freeSurface();
            SDLMain.quit();
        }

    }
```

Ce qui nous donne :

Mais c'est horrible ! Oui il fraudait que l'image ne soit pas rectangulaire, et qu'elle ait la forme de Zack... Bon pas de panique on va voir ça.

Couleur de transparence

La bibliothèque SDL permet de définir une couleur de transparence, qui ne sera donc pas affichée. Vous avez vu que l'image que j'ai ajoutée a un fond rose, qui correspond au code de couleur 255, 0, 255 : on va donc dire à SDL de la rendre transparente, ce qui aura pour effet l'affichage correct de la deuxième image. Pour réaliser cela nous devrons utiliser UNE nouvelle méthode (oui juste une), la méthode `setColorKey()` qui prendra deux paramètres :

- une constante,
- la couleur à rendre transparente.

Pour le choix de la constante nous utiliserons `SDLVideo.SDL_SRCCOLORKEY` , elle indique à SDL qu'il faut rendre une couleur transparente. Pour la couleur on l'indiquera avec la méthode `mapRGP(int r, int g, int b)` .

Il faudra donc ajouter ce code avant la boucle principale (avant de bliter cela va de soit) :

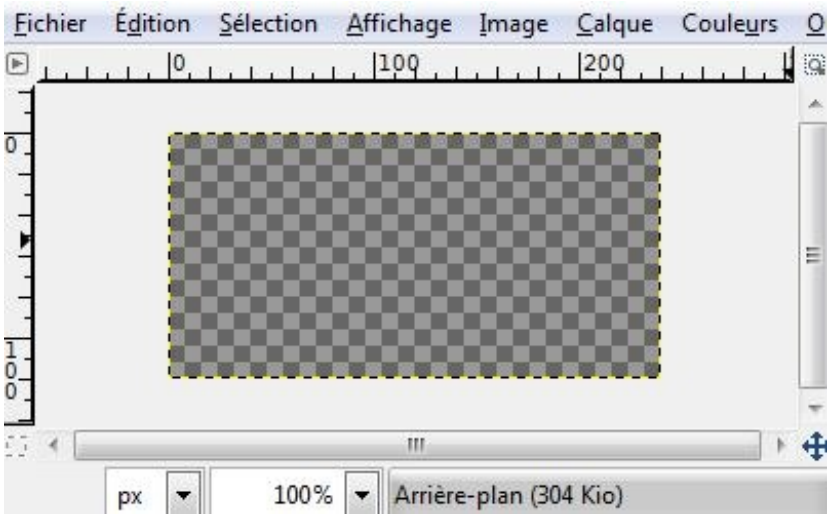
Code : Java

```
// Affichage avec transparence
zack.setColorKey(SDLVideo.SDL_SRCCOLORKEY, zack.mapRGB(255, 0,
255));
```

Vous pouvez essayer ça fonctionne, mais il y a encore un autre moyen qui évite l'usage de `setColorKey` : l'utilisation d'une image au format png, qui inclue un canal alpha !

C'est quoi un canal Alpha ?

Pour faire simple c'est une couleur de transparence, qui est généralement représentée par des damiers.



La transparence dans The Gimp.

Il faut donc que votre image contienne un canal alpha avec cette couleur (les damiers) et qu'elle soit au format png. Lorsque vous la chargerez, SDL comprendra tout de suite que cette image contient un canal alpha et l'appliquera automatiquement (merci SDL_image).

Voilà le résultat :

C'est bon cette fois-ci.

Je ne vous redonne pas tout le code, car il y a juste une méthode à ajouter avant la boucle `while` , donc je pense que vous vous en sortirez sans problème.

Donner de la transparence à l'image

Nous allons voir comment donner de la transparence à une image avec la méthode `setColorKey()` . Cette méthode prend deux paramètres qui sont :

- une constante,
- le taux de transparence, compris entre 0 et 255.

La constante sert à activer le mode Alpha, cela va en quelque sorte fondre l'image sur le fond d'écran, nous utiliserons donc pour cela `SDLVideo.SDL_SRCCOLORKEY` . Sachez que si à la place de cette constante vous mettez 0, cela aura pour effet de ne pas activer la couleur Alpha. Pour ce qui est de la transparence, une valeur de 0 rendra votre image complètement transparente, une valeur de 255 la rendra complètement visible. Vous pouvez donc essayer ce code :

Code : Java

```
        // Chargement d'une image et position
        SDLSurface jim = SDLImage.load("images/jim.bmp");
```

```
        jim.setColorKey(SDLVideo.SDL_SRCCOLORKEY, jim.mapRGB(255, 0, 255));
        SDLRect jimPos = new SDLRect(120, 50);

        // Gestion de la transparence
        jim.setAlpha(SDLVideo.SDL_SRCALPHA, 15);
```

Commentons se code : jim est le nom de l'image et le nom de mon objet de type `SDLSurface` . J'applique dans un premier temps la couleur de transparence de l'image avec `jim.setColorKey(...);` , puis je définis une position sur la fenêtre. Ensuite j'applique un taux de transparence à mon image de 15 (on ne voit presque pas l'image).

Voilà quelques exemples :

Taux de transparence : 15

Taux de transparence : 80

Taux de transparence : 200

Voici le code qui a permis de réaliser ces captures d'écran :

Code : Java

```
import sdljava.SDLException;
import sdljava.SDLMain;
import sdljava.event.SDLEvent;
import sdljava.image.SDLImage;
import sdljava.video.SDLRect;
import sdljava.video.SDLSurface;
import sdljava.video.SDLVideo;

public class Alpha {

 public static void main(String[] args) throws SDLException {

  // Evenements 1/2
  SDLEvent event = null;
  boolean boucle = true;

  // Initialisation de SDL et de l'affichage
  SDLMain.init(SDLMain.SDL_INIT_VIDEO);
  SDLVideo.wmSetCaption("Gestion de la transparence", null);
  SDLSurface screen = SDLVideo.setVideoMode(320, 240, 32,
SDLVideo.SDL_DOUBLEBUF | SDLVideo.SDL_HWSURFACE);

  // Chargement d'une image et position
  SDLSurface jim = SDLImage.load("images/jim.bmp");
  jim.setColorKey(SDLVideo.SDL_SRCCOLORKEY, jim.mapRGB(255, 0,
255));
  SDLRect jimPos = new SDLRect(120, 50);

  // Gestion de la transparence
  jim.setAlpha(SDLVideo.SDL_SRCALPHA, 15);

  // Evénements 2/2
  while (boucle)
  {
   event = SDLEvent.waitEvent();
   if (event.getType() == SDLEvent.SDL_QUIT)
    boucle = false;

   screen.fillRect(screen.mapRGB(20, 150, 78));
   jim.blitSurface(screen, jimPos);
   screen.flip();
  }
```

```
        // Fin du programme
        jim.freeSurface();
        screen.freeSurface();
        SDLMain.quit();
      }

    }
```

Cette partie est terminée, je vous avais dit que c'était court, mais avouez que vous avez appris plein de nouvelles choses non ? On a enfin fini ! Vous êtes donc capables (rapidement normalement) de créer une petite application SDL qui affiche des formes et / ou des images.

La première partie est terminée. Faites une pause histoire de bien tout assimiler (le bourrage de crâne n'est pas une bonne solution).

Partie 2 : Notions intermédiaires

Dans cette partie nous verrons (en partie) comment fonctionnent les événements, mais aussi comment ajouter du son, de la musique et comment écrire sur la fenêtre. Vous l'avez compris, nous n'allons pas nous ennuyer. Je vous proposerai aussi une introduction à OpenGL, ainsi qu'un petit TP pour clôturer cette partie.

Les événements

Nous voilà dans une partie riche, car c'est ici que nous allons communiquer avec notre fenêtre et par la même occasion lui donner un peu de vie. Dans un premier temps nous verrons la boucle d'événement avec les différents événements, puis nous nous intéresserons aux événements clavier, qui nous permettrons par exemple de déplacer un sprite sur l'écran. Sachez qu'après ce chapitre vous pourrez tenter de réaliser le jeu Mario Sokoban proposé par M@teo21 dans son tutoriel C/SDL. Mais je vois que vous salivez déjà, alors c'est parti !

Les bases

La lib SDL met à notre disposition un mécanisme assez simple pour gérer les événements, on distinguera d'ailleurs deux types d'événements dans ce tutoriel :

- `SDLEvent.waitEvent()`
- `SDLEvent.pollEvent()`

`SDLEvent.waitEvent()` permet d'attendre qu'un événement se soit produit pour continuer ; dans le cas où il n'y a pas d'événement, cette méthode bloque le programme en attendant qu'elle en reçoive un, alors que `SDLEvent.pollEvent()` n'attend pas qu'un événement se soit produit et permet au programme de continuer ses traitements. Donc `SDLEvent.waitEvent()` utilisera moins de ressources processeur, alors que `SDLEvent.pollEvent()` utilisera plus de ressources processeur.

Dans quel cas les utiliser ?

Eh bien prenons le cas du jeu Mario Sokoban : si le joueur ne touche pas aux commandes, rien ne se passe à l'écran, le personnage ne bouge pas et le décor non plus ; en revanche, dès que le joueur utilise le clavier pour contrôler le personnage, celui-ci se déplace et le décor peut être amené à changer. On peut aussi penser au jeu d'échec, etc.

Maintenant prenons l'exemple du jeu Snake (non pas Metal Gears), ce bon vieux serpent doit se déplacer tout le temps quoi qu'il arrive, et si on utilise un événement de type `SDLEvent.waitEvent()` , alors celui-ci ne se déplacera que quand le joueur utilisera son clavier (sa souris ou sa manette).

Pour résumer : avec `SDLEvent.waitEvent()` l'affichage est mis à jour uniquement si un événement est détecté, et avec `SDLEvent.pollEvent()` , l'affichage est mis à jour tout le temps.

Nous utiliserons dans un premier temps les événements de type `SDLEvent.waitEvent()` . Pour gérer les événements vous devrez importer une nouvelle classe : `sdljava.event.SDLEvent` , et aussi créer ce que l'on appellera la *boucle principale* du programme.

Voilà comment vont se dérouler les opérations :

Code : Autre

```
Importation des classes

Début du programme principal
    Initialisation de sdljava
    Variable Running -> true : boolean
    TantQue Running = true Faire
        Détecter les événements
        Si événements Faire
            Traitements
        Mettre à jour l'affichage
```

```
        FinTantQue
    Fin du programme principal
```

C'est le schéma typique d'un programme avec SDL. Si vous avez bien suivi ce que j'ai dit à propos de `SDLEvent.waitEvent()` , l'affichage ne sera mis à jour que si un événement est détecté. Passons aux choses sérieuses.

Le package `sdljava.event.*;` contient des méthodes static qui permettent de mettre en place le gestionnaire d'événements, mais aussi tout un tas de constantes très utiles comme `SDLEvent.SDL_QUIT` qui représente la fermeture de la fenêtre lorsque l'on clique sur la croix en haut à droite.

Voici tout de suite un exemple (vous avez assez patienté) :

Code : Java

```
import sdljava.SDLMain;
import sdljava.SDLException;
import sdljava.video.*;
import sdljava.event.SDLEvent;


public class TutorielEvent {

 public static void main(String[] args) throws SDLException,
InterruptedException {

  // Initialisation de notre contexte SDL
  SDLMain.init(SDLMain.SDL_INIT_VIDEO);
  SDLSurface screen = SDLVideo.setVideoMode(800, 600, 32,
SDLVideo.SDL_DOUBLEBUF | SDLVideo.SDL_HWSURFACE);
  SDLVideo.wmSetCaption("Les événements avec SDLJava", null);

  // Variable de boucle
  boolean Running = true;

  while (Running) {

   SDLEvent event = SDLEvent.waitEvent();
   if (event.getType() == SDLEvent.SDL_QUIT) Running = false;
   // Mise à jour de l'affichage
   screen.flip();
  }
  screen.freeSurface();
  SDLMain.quit();
 }
}
```

Vous constaterez que ce code respecte à la lettre le schéma que j'ai indiqué plus haut. La boucle **`while`** va permettre de récupérer les événements **TantQue** la variable `Running` est vraie. Pour cela on initialise un objet de type `SDLEvent` nommé ici `event` , avec la méthode static `SDLEvent.waitEvent()` . Une fois celle-ci opérationnelle, nous testons si un événement a eu lieu ou non. On utilise par exemple une structure conditionnelle **`if`** , mais vous verrez qu'un **`switch`** sera beaucoup plus efficace tout à l'heure.

L'événement généré peut être de plusieurs types, et je vous invite à consulter la documentation sur SDLEvent pour plus d'informations. Grâce à la méthode `getEvent()` nous pouvons donc récupérer le type d'événement. On le compare ensuite à une des constantes ; ici il s'agit de la constante chargée de fermer le fenêtre qui nous intéresse, soit `SDLEvent.SDL_QUIT` , donc lorsque celle-ci sera détectée la variable `Running` sera fausse, et la boucle principale sera terminée, ce qui terminera notre programme.

Voici une petite liste de constantes :

Constante	Déclenchement

SDLEvent.SDL_QUIT	Lorsque l'utilisateur clique sur la croix de fermeture de la fenêtre.
SDLEvent.SDL_KEYDOWN	Lorsqu'une touche est enfoncée.
SDLEvent.SDL_KEYUP	Lorsqu'une touche n'est pas enfoncée.
SDLEvent.SDL_VIDEORESIZE	Lorsque l'utilisateur modifie la taille de la fenêtre.

Il y en a d'autres, et vous les trouverez toutes dans la documentation.

Les événements clavier

Le titre parle de lui-même, nous allons voir tout de suite comment détecter des événement venant du clavier, en particulier lorsque l'utilisateur va appuyer sur une touche. Nous utiliserons dans un premier temps un événement de type `SDLEvent.waitEvent()` , puis nous verrons ensuite comment fonctionne `SDLEvent.pollEvent()` .

Avec SDLEvent.waitEvent()

Pour gérer le clavier nous devons en premier lieu savoir si une touche a été pressée : si c'est le cas, nous devrons aviser en conséquence, sinon le programme continuera son exécution normale (c'est le principe de la boucle d'événements si vous vous souvenez).
Nous devons importer deux nouvelles classes pour cela : `sdljava.event.SDLKeyboardEvent` et `sdljava.event.SDLKey` . Ensuite il faudra créer un nouvel objet de type `SDLKeyboardEvent` pour interagir avec le clavier, et enfin utiliser les constantes de `SDLKey` pour attribuer un traitement à une touche.

Voici un exemple qui utilise les événements clavier. Dans ce dernier, vous pouvez fermer la fenêtre avec la croix en haut à droite (`SDLEvent.SDL_QUIT`), mais vous pouvez aussi fermer l'application en appuyant sur la touche *Echap* (`SDLKey.SDLK_ESCAPE`).

Code : Java

```
import sdljava.SDLMain;
import sdljava.SDLException;
import sdljava.video.*;
import sdljava.event.SDLEvent;
import sdljava.event.SDLKeyboardEvent;
import sdljava.event.SDLKey;


public class TutorielEvent {

 public static void main(String[] args) throws SDLException,
InterruptedException {

  // Initialisation de notre contexte SDL
  SDLMain.init(SDLMain.SDL_INIT_VIDEO);
  SDLSurface screen = SDLVideo.setVideoMode(800, 600, 32,
SDLVideo.SDL_DOUBLEBUF | SDLVideo.SDL_HWSURFACE);
  SDLVideo.wmSetCaption("Les événements avec SDLJava", null);

  // Variable de boucle
  boolean Running = true;

  while (Running) {

   SDLEvent event = SDLEvent.waitEvent();

   if (event.getType() == SDLEvent.SDL_QUIT) Running = false;
   // Si l'utilisateur presse une touche
   if (event.getType() == SDLEvent.SDL_KEYDOWN)
   {
    // On génère un nouvel événement de type SDLKeyboardEvent
    // Attention au cast !
    SDLKeyboardEvent eventK = (SDLKeyboardEvent) event;

    // Les méthodes getType() et getSym() se ressemblent sur le
```

```
        principe d'utilisation
            switch (eventK.getSym()) {

            // Si la touche espace est pressée
            case SDLKey.SDLK_ESCAPE: Running = false;        break;
            }
           }
           // Mise à jour de l'affichage
           screen.flip();
          }
          screen.freeSurface();
          SDLMain.quit();
         }
        }
```

Je pense que vous avez compris comment fonctionnent les événements clavier, mais je vais reprendre quelques points car de nouveaux éléments sont apparus. 

```
if (event.getType() == SDLEvent.SDL_KEYDOWN)
```

Ici, on effectue un test pour voir si une touche (n'importe laquelle) a été pressée.

```
SDLKeyboardEvent eventK = (SDLKeyboardEvent) event;
```

Quel code barbare vous ne trouvez pas ?  L'objet `event` créé tout à l'heure ne peut pas gérer les touches du clavier, il faut donc faire appel à un autre objet de type `SDLKeyboardEvent` pour effectuer ce traitement. Ce nouvel objet est initialisé avec l'objet `event` précédent, sauf que celui-ci sera vu comme un événement de type `SDLKeyboardEvent`. Voilà pourquoi on fait un cast (ou conversion explicite).

```
switch (eventK.getSym())
```

J'ai utilisé une instruction **`switch`**, mais j'aurais pu utiliser une instruction **`if`** comme tout à l'heure. D'ailleurs pourquoi le **`switch`** ? Eh bien ici, aucun intérêt, mais plus bas vous comprendrez pourquoi. En effet, lorsque vous aurez plusieurs touches à gérer, cette méthode sera particulièrement efficace. On utilise la méthode `getSym()` qui nous permet de savoir quelle touche a été enfoncée ; cette méthode est comparable à `getType()`.

```
case SDLKey.SDLK_ESCAPE: Running = false;
```

Et c'est ici, grâce à la constante `SDLKey.SDLK_ESCAPE` que l'on passe notre variable `Running` à la valeur **`false`**, ce qui aura pour effet de fermer l'application.

Nous voilà au point sur les événements clavier de type `waitEvent()`, mais qu'en est-il avec `pollEvent` ?

Avec SDLEvent.pollEvent()

Voici le même code avec une gestion des événements de type `pollEvent` :

Code : Java

```
import sdljava.SDLMain;
import sdljava.SDLException;
import sdljava.video.*;
import sdljava.event.SDLEvent;
import sdljava.event.SDLKeyboardEvent;
import sdljava.event.SDLKey;


public class TutorielEvent {

 public static void main(String[] args) throws SDLException,
InterruptedException {
```

```
        // Initialisation de notre contexte SDL
        SDLMain.init(SDLMain.SDL_INIT_VIDEO);
        SDLSurface screen = SDLVideo.setVideoMode(800, 600, 32,
      SDLVideo.SDL_DOUBLEBUF | SDLVideo.SDL_HWSURFACE);
        SDLVideo.wmSetCaption("Les événements avec SDLJava", null);

        // Variable de boucle
        boolean Running = true;

        while (Running) {

         // On utilise pollEvent() comme gestionnaire d'événements
         SDLEvent event = SDLEvent.pollEvent();

         // Si un événement de type SDLEvent est détecté faire
         if (event instanceof SDLEvent ) {
          switch (event.getType()) {
          case SDLEvent.SDL_QUIT:  Running = false; break;
          }
         }

         // Si un événement de type SDLKeyboardEvent est détecté faire
         if (event instanceof SDLKeyboardEvent ) {
          SDLKeyboardEvent eventK = (SDLKeyboardEvent) event;

          switch (eventK.getSym()) {

          case SDLKey.SDLK_ESCAPE: Running = false; break;
          }
         }
         // Mise à jour de l'affichage
         screen.flip();
        }
        screen.freeSurface();
        SDLMain.quit();
       }
      }
```

Wahoo, les choses ont changé, ça ne fonctionne plus pareil !

En effet, comme `pollEvent()` n'attend pas qu'un événement se soit produit pour rendre la main au programme principal, la détection n'est plus la même, mais cela ne doit pas vous faire peur pour autant. Si vous voulez plus d'informations sur l'opérateur **`instanceof`**, je vous invite à consulter ce lien. Reprenons quelques lignes ensemble :

```
if (event instanceof SDLEvent )
```

Si je traduis littéralement, cela nous donne quelque chose comme : *« Si l'objet event est de type SDLEvent alors... »*. La suite vous la connaissez, car elle est identique aux deux types de gestion des événements.

Voilà qui conclut cette partie, nous allons voir maintenant quelques petits trucs sympathiques et utiles.

La gestion du temps

Maintenant que vous connaissez la différence entre `pollEvent()` et `waitEvent()` et tout ce qui est relatif à l'affichage, nous allons nous intéresser au temps.

Souvenez-vous, `pollEvent()` n'attend pas qu'il y ait un événement déclenché pour rendre la main au programme, alors que `waitEvent()` est dite « bloquante », car elle attend qu'un événement soit déclenché. Cela veut dire que dans un cas, on aura un programme qui ne consommera presque pas de ressources quand l'utilisateur ne fera rien, mais que dans un autre cas, si on ne maîtrise pas le temps, notre programme va monopoliser toutes les ressources processeur !

Nous allons commencer tout de suite avec deux exemples, un avec `pollEvent()` et un autre `waitEvent()` ; vous comprendrez ensuite où je veux en venir.

Voici les règles du jeu :

Je vous demande simplement d'exécuter l'application et de ne **RIEN** faire ni **RIEN** bouger de façon à ne pas déclencher d'événements. Au bout de 20 secondes ou plus vous appuierez sur la touche *Echap* pour quitter et vous lirez les informations dans la console.

Avec waitEvent()

Code : Java

```
import java.io.IOException;
import sdljava.SDLMain;
import sdljava.SDLException;
import sdljava.video.*;
import sdljava.event.SDLEvent;
import sdljava.event.SDLKeyboardEvent;
import sdljava.event.SDLKey;

public class TutorielEvent {

 public static void main(String[] args) throws SDLException,
IOException {

  // Initialisation de notre contexte SDL
  SDLMain.init(SDLMain.SDL_INIT_VIDEO);
  SDLSurface screen = SDLVideo.setVideoMode(800, 600, 32,
SDLVideo.SDL_DOUBLEBUF | SDLVideo.SDL_HWSURFACE);
  SDLVideo.wmSetCaption("La gestion du temps avec SDL", null);

  // Temps d'initialisation : on divise par 1000 pour avoir une
valeur en secondes
  long lastFrame = System.currentTimeMillis() / 1000;

  long i = 0;

  // Variable de boucle
  boolean Running = true;

  while (Running) {

   SDLEvent event = SDLEvent.waitEvent();

   switch (event.getType()) {

    case SDLEvent.SDL_QUIT:  Running = false; break;

    case SDLEvent.SDL_KEYDOWN:
     SDLKeyboardEvent kev = (SDLKeyboardEvent) event;
     switch (kev.getSym()) {

      case SDLKey.SDLK_ESCAPE: Running = false; break;
    }
   }

   // Mise à jour de l'affichage
   screen.flip();
   i++;
  }

  long frame = System.currentTimeMillis() / 1000;
  System.out.println("Temps passé avant la fermeture : " + (frame -
lastFrame) + " secondes");
  System.out.println("L'affichage a été mis à jour : " + i + "
```

```
fois");

      screen.freeSurface();
      SDLMain.quit();
    }
  }
```

En effectuant le test 33 secondes, voici ce que j'obtiens :

Code : Console

```
Temps passé avant la fermeture : 33 secondes
L'affichage a été mis à jour : 3 fois
```

Qu'est-ce que ça veut dire docteur ? Je croyais que la méthode `waitEvent()` était bloquante !

C'est normal, lorsque vous ne faites rien, l'affichage n'est pas mis à jour. Par contre quand vous avez appuyé sur la touche *Echap*, il y a un temps pour que l'application se termine. Maintenant voyons ce que ce même code donne avec l'autre type d'événement... (les règles sont identiques ).

Avec pollEvent()

Code : Java

```
import java.io.IOException;
import sdljava.SDLMain;
import sdljava.SDLException;
import sdljava.video.*;
import sdljava.event.SDLEvent;
import sdljava.event.SDLKeyboardEvent;
import sdljava.event.SDLKey;

public class TutorielEvent {

 public static void main(String[] args) throws SDLException,
IOException {

  // Initialisation de notre contexte SDL
  SDLMain.init(SDLMain.SDL_INIT_VIDEO);
  SDLSurface screen = SDLVideo.setVideoMode(800, 600, 32,
SDLVideo.SDL_DOUBLEBUF | SDLVideo.SDL_HWSURFACE);
  SDLVideo.wmSetCaption("La gestion du temps avec SDL", null);

  // Temps d'initialisation : on divise par 1000 pour avoir une
valeur en secondes
  long lastFrame = System.currentTimeMillis() / 1000;

  long i = 0;

  // Variable de boucle
  boolean Running = true;

  while (Running) {

   // On utilise pollEvent() comme gestionnaire d'événement
   SDLEvent event = SDLEvent.pollEvent();

   // Si un événement de de type SDLEvent est détecté faire
```

```
        if (event instanceof SDLEvent ) {
         switch (event.getType()) {
         case SDLEvent.SDL_QUIT:  Running = false; break;
         }
        }

        // Si un événement de de type SDLKeyboardEvent est détecté faire
        if (event instanceof SDLKeyboardEvent ) {
         SDLKeyboardEvent eventK = (SDLKeyboardEvent) event;

         switch (eventK.getSym()) {

         case SDLKey.SDLK_ESCAPE: Running = false; break;
         }
        }


        // Mise à jour de l'affichage
        screen.flip();
        i++;
       }

       long frame = System.currentTimeMillis() / 1000;
       System.out.println("Temps passé avant la fermeture : " + (frame -
    lastFrame) + " secondes");
       System.out.println("L'affichage a été mis à jour : " + i + "
    fois");

       screen.freeSurface();
       SDLMain.quit();
      }
     }
```

En effectuant le test 19 secondes, voici ce que j'obtiens :

Code : Console

```
Temps passé avant la fermeture : 19 secondes
L'affichage a été mis à jour : 50028 fois
```

Wahoooo l'affichage est mis à jour 50028 fois !!!

Oui, et si vous avez suivi ce que j'ai dit jusqu'ici, c'est tout à fait normal car la méthode n'est pas bloquante. Je vous laisse donc imaginer l'impact sur les performances si on ne gère pas le temps...

Tu as une solution ?

Bien sûr ! SDLJava nous propose une classe `Timer` qui permet de faire deux choses (notez que la version C permet de faire plus de choses, mais nous avons un binding inachevé) :

- Mettre le programme en pause **n <u>milisecondes</u>** avec `SDLTimer.delay(long ms)` ;
- Récupérer le temps passé depuis l'initialisation du programme avec `SDLTimer.getTicks()` .

Mais dis-moi, Java n'a pas déjà des fonctions similaires ?

Si, et vous les connaissez bien justement, puisque nous les avons déjà utilisées. Une dans la 1ère partie qui est `Thread.sleep(long ms)` et une autre dans cette partie qui est `System.currentTimeMillis()` .

Pour ce tutoriel, j'utiliserai donc les méthodes incluses directement dans Java, car c'est exactement la même chose. Si vous voulez utiliser les méthodes de SDLJava, il vous faudra importer le package `sdljava.SDLTimer;` et dans les deux cas il vous faudra gérer ou pas l'exception `InterruptedException` .

Bon c'est très bien tout ça mais si j'ai à utiliser `pollEvent()` comment dois-je faire ?

En limitant le taux de mise à jour de l'affichage bien sûr !

Pour cela il existe plusieurs méthodes, comme détecter le nombre de frames par secondes, et de limiter en conséquence. Mais une solution assez simple est de placer un `Thread.sleep(15)` après la mise à jour de l'affichage.

Voici un exemple :

Code : Java

```
import sdljava.SDLMain;
import sdljava.SDLException;
import sdljava.video.*;
import sdljava.event.SDLEvent;
import sdljava.event.SDLKeyboardEvent;
import sdljava.event.SDLKey;


public class TutorielEvent {

 public static void main(String[] args) throws SDLException,
InterruptedException {

  // Initialisation de notre contexte SDL
  SDLMain.init(SDLMain.SDL_INIT_VIDEO);
  SDLSurface screen = SDLVideo.setVideoMode(800, 600, 32,
SDLVideo.SDL_DOUBLEBUF | SDLVideo.SDL_HWSURFACE);
  SDLVideo.wmSetCaption("La gestion du temps avec SDL", null);

  // Temps d'initialisation : on divise par 1000 pour avoir une
valeur en secondes
  long lastTime = System.currentTimeMillis() / 1000;

  // Variables pour compter le nombre d'itérations
  long i = 0;

  // Variable de boucle
  boolean Running = true;

  while (Running) {

   // On utilise pollEvent() comme gestionnaire d'événement
   SDLEvent event = SDLEvent.pollEvent();

   // Si un événement de type SDLEvent est détecté faire
   if (event instanceof SDLEvent ) {
    switch (event.getType()) {
    case SDLEvent.SDL_QUIT:  Running = false; break;
    }
   }

   // Si un événement de type SDLKeyboardEvent est détecté faire
```

```
        if (event instanceof SDLKeyboardEvent ) {
         SDLKeyboardEvent eventK = (SDLKeyboardEvent) event;

         switch (eventK.getSym()) {

         case SDLKey.SDLK_ESCAPE: Running = false; break;
         }
        }

        // Mise à jour de l'affichage
        screen.flip();
        i++;
        Thread.sleep(10);

       }

       long time = System.currentTimeMillis() / 1000;
       System.out.println("Temps écoulé : " + (time - lastTime) + "
secondes");
       System.out.println("L'affichage a été mis à jour : " + i + "
fois");
       System.out.println("Soit une moyenne de : " + i / (time -
lastTime) + " frames par secondes");

       screen.freeSurface();
       SDLMain.quit();
      }
     }
```

Et le résultat console :

Code : Console

```
Temps écoulé : 10 secondes
L'affichage a été mis à jour : 1055 fois
Soit une moyenne de : 105 frames par secondes
```

Vous constatez qu'il y a moins de mises à jour de l'affichage, et c'est normal, nous sommes maintenant dans de bonnes conditions pour gérer l'affichage avec `pollEvent()` .

Passons maintenant à un exercice, histoire de voir si vous avez retenu la leçon...

Exercice : diriger un sprite à l'écran !

Le sujet

Nous y voilà... vous allez enfin pouvoir charger un objet et le déplacer sur la fenêtre. Pour ce petit exercice je vais vous fournir un sprite à déplacer (bien que vous puissiez utiliser celui que vous voulez). Le but est de déplacer un petit avion dans quatre directions en gérant la vitesse de déplacement. Si on appuie sur la **touche Shift Gauche** lors du déplacement alors l'avion se déplace deux fois plus vite, sinon il se déplace à sa vitesse normale.

Voici le sprite en question :

Quelques indications avant de commencer.

Pour contrôler la vitesse de déplacement vous devrez créer une variable de type `boolean` d'état qui vaudra **`true`** si vous avez appuyé sur la touche gauche shift, ou **`false`** dans le cas où vous n'appuyez plus sur cette touche.

Vous devrez donc tester si un événement de type `SDLEvent.SDL_KEYUP` a été déclenché et traiter la variable en conséquence (ne la brutalisez pas non plus, hein ?).

Avant la boucle principale je vous conseille d'appeler la méthode `SDLEvent.enableKeyRepeat(long delay, long intervale)` où sont exprimés en millisecondes :

- `long delay` : le temps que doit rester enfoncée une touche pour activer la répétition des touches ;
- `intervale` : l'intervalle entre deux générations d'événements.

Personnellement je fixe les valeurs à 10 toutes les deux, essayez de les changer, vous verrez le déplacement sera plus ou moins rapide.

Je vous ai tout dit, à vous de jouer !

Correction

Code : Java

```
import sdljava.SDLException;
import sdljava.SDLMain;
import sdljava.event.SDLEvent;
import sdljava.event.SDLKey;
import sdljava.event.SDLKeyboardEvent;
import sdljava.image.SDLImage;
import sdljava.video.SDLRect;
import sdljava.video.SDLSurface;
import sdljava.video.SDLVideo;

public class ExerciceEvent {


 public static void main(String[] args) throws SDLException {

  // Initialisation de SDL
  SDLMain.init(SDLMain.SDL_INIT_VIDEO);
  SDLSurface screen = SDLVideo.setVideoMode(640, 480, 32,
SDLVideo.SDL_DOUBLEBUF | SDLVideo.SDL_HWSURFACE);
  SDLVideo.wmSetCaption("My Java Shoot Them Up Baby ;)", null);

  // Création de l'objet joueur
  SDLSurface plane = SDLImage.load("plane.gif");
  int positionPlaneX = screen.getWidth() / 2 - plane.getWidth() / 2;
  int positionPlaneY = screen.getHeight() / 2 - plane.getHeight() /
2;
  boolean acceleration = false;

  boolean isRunning = true;
  SDLEvent.enableKeyRepeat(10, 10);

  do
  {
   SDLEvent event = SDLEvent.waitEvent();

   if (event.getType() == SDLEvent.SDL_QUIT)
    isRunning = false;

   // Si la touche Shift de gauche n'est pas pressée,
   // pas d'accélération
   if (event.getType() == SDLEvent.SDL_KEYUP) {
```

```
            SDLKeyboardEvent keyEventUp = (SDLKeyboardEvent) event;

            if ( keyEventUp.getSym() == SDLKey.SDLK_LSHIFT )
             acceleration = false;
           }

           if (event.getType() == SDLEvent.SDL_KEYDOWN) {

            SDLKeyboardEvent keyEvent = (SDLKeyboardEvent) event;

            switch (keyEvent.getSym()) {

            case SDLKey.SDLK_LSHIFT:
             acceleration = true; break;

            case SDLKey.SDLK_ESCAPE:
             isRunning = false;  break;

            // On gère le déplacement avec ou sans accélération
            case SDLKey.SDLK_UP:
             if (acceleration)
              positionPlaneY -= 4;
             else
              positionPlaneY -= 2;
             break;

            case SDLKey.SDLK_DOWN:
             if (acceleration)
              positionPlaneY += 4;
             else
              positionPlaneY += 2;
             break;

            case SDLKey.SDLK_LEFT:
             if (acceleration)
              positionPlaneX -= 4;
             else
              positionPlaneX -= 2;
             break;

            case SDLKey.SDLK_RIGHT:
             if (acceleration)
              positionPlaneX += 4;
             else
              positionPlaneX += 2;
             break;
            }
           }

           // On met à jour l'affichage
           screen.fillRect(screen.mapRGB(255, 255, 255));
           blit.blitSurface(positionPlaneX, positionPlaneY, plane, screen);
           screen.flip();

          } while (isRunning);

          // Libération de la mémoire
          plane.freeSurface();
          screen.freeSurface();
          SDLMain.quit();
         }
        }

        /*
        * Cette méthode permet de bliter une SDLSurface sans avoir besoin
        de créer
        * plusieurs objets SDLRect. Dans le cas de cet exemple elle n'est
        pas très utile
        * mais dans le cas où vous avez plusieurs SDLSurfaces à bliter,
        elle montrera toute
```

```
        * son efficacité !
        */

        class blit {
         static void blitSurface(int x, int y, SDLSurface src, SDLSurface
        dst) throws SDLException {
          SDLRect temp = new SDLRect(x, y);
          src.blitSurface(dst, temp);
         }
        }
```

Ce n'était pas trop dur, mais ne vous en faites pas, le but n'était pas de vous piéger, mais de vous faire pratiquer un peu. 
Eh bien nous avons vu pas mal de choses mine de rien.  J'espère que cette partie vous aura plu, car c'est la plus importante du tutoriel. En effet à partir de maintenant vous pouvez laisser libre cours à votre imagination et créer des petits jeux. J'ai tout de même une recommandation concernant les blocs `try... catch` : n'en abusez pas, car si vous faites un jeu utilisant de multiples classes, etc. ceux-ci vous limiteront et vous causeront bien des erreurs.

Jouez du son et de la musique

Je vous l'annonce tout de suite : le plus dur est passé (quoique ?) ! Dans ce chapitre et le suivant nous allons aborder des points très faciles à comprendre et à mettre en place. Comme l'indique le titre du chapitre, nous allons jouer de la musique et des sons avec SDL. Pour cela nous utiliserons les bibliothèques fournies avec SDLJava, j'ai nommé : SDL_audio et SDL_mixer.

Installation de SDL_mixer

Comme vous devez vous en douter, *SDL_mixer* ne fait pas partie de *SDL* à l'origine, c'est une bibliothèque tierce, tout comme *SDL_image*. SDL_mixer permet d'apporter une couche au-dessus de *sdl_audio* (qui elle fait partie de SDL). Nous allons donc dans un premier temps récupérer les fichiers à ajouter à la racine de votre projet. Cependant je ne détaillerai pas l'installation cette fois-ci, car je l'ai déjà fait pour *SDL* et *SDL_image*, donc vous devez vous douter que les choses ne changent pas pour *SDL_mixer*.

Récupérer les bons fichiers

Si vous vous rendez sur la page dédiée à SDL_mixer, vous avez le choix entre télécharger les sources, ou les fichiers binaires. Si vous êtes sous Linux et que vous n'avez pas installé libsdl-mixer, vous pouvez au choix, compiler les sources et récupérer comme nous l'avons fait le fichier .so ou bien installer le paquet qui correspond à votre distribution. Sous Windows vous avez juste à télécharger la version normale (celle qui n'est pas suffixée par -devel).

Dans tout les cas vous devrez copier les fichiers dans votre répertoire de projet.

Et les fichiers, parlons-en !

Sous Linux vous devez avoir :

- libSDL_mixer.a
- libogg.a
- libvorbis.a
- libvorbisfile.a
- libsmepg.a

Pour pouvoir lire des fichiers midi, vous aurez besoin d'installer le paquet timidity. De plus j'ai constaté que sur certaines distributions la lecture des fichiers midi était impossible (la faute à SDLJava pas à Linux).

Suivant votre distribution et le type d'installation que vous avez fait de la bibliothèque SDLJava, vous n'aurez besoin que de libSDL_mixer.a.

Sous Windows vous devez avoir :

- libogg-0.dll
- libvorbis-0.dll
- libvorbisfile-3.dll
- SDL_mixer.dll
- smpeg.dll

Maintenant que vous avez ces fichiers copiés, vous êtes prêt à travailler, alors passons à la suite.

Jouons du son et... de la musique

Je ne sais pas si vous vous en souvenez, mais lorsque l'on initialise SDL avec la méthode `SDLMain.init()` , il faut spécifier des `flags` (drapeaux), et pour l'instant nous initialisions SDL uniquement en mode vidéo. Eh bien cela va changer, puisque nous allons maintenant initialiser en plus le son avec la constante `SDLMain.SDL_INIT_AUDIO` .

De plus nous devons importer plusieurs nouvelles classes, donc dans un souci de simplicité nous ferons un import... de groupe.

Code : Java

```
import sdljava.audio.*;
import sdljava.mixer.*;
```

La classe `SDLAudio` contient des constantes que nous utiliserons pour initialiser le son. Quant à la classe `SDLMixer`, elle contient des méthodes **`static`** pour jouer des sons et de la musique.

Concernant les formats, vous pouvez charger du wave, mp3, ogg, midi. Pour une liste plus complète consultez la documentation.

Vous en mourez d'envie, eh bien voilà un exemple qui permet avec les touches du pavé numérique de jouer un son et une musique. Nous commenterons cet exemple plus bas ne vous en faites pas. 

Code : Java

```
import sdljava.SDLMain;
import sdljava.SDLException;

import sdljava.audio.*;
import sdljava.mixer.*;
import sdljava.event.*;
import sdljava.video.*;

public class Son {
 public static void main(String [] args) throws SDLException {

  // On initialise en plus la constante SDL_INIT_AUDIO
  SDLMain.init(SDLMain.SDL_INIT_VIDEO | SDLMain.SDL_INIT_AUDIO);
  SDLSurface screen = SDLVideo.setVideoMode(320, 240, 32,
SDLVideo.SDL_HWSURFACE | SDLVideo.SDL_DOUBLEBUF);
  SDLVideo.wmSetCaption("Du son avec SDL_mixer en Java", null);

  // Initialisation de SDLMixer
  SDLMixer.openAudio(44100, SDLAudio.AUDIO_S16SYS, 2 , 1024);
  // On passe le nombre de canaux de mixage à 32
  int result = SDLMixer.allocateChannels(32);
  if (result != 32) throw new SDLException("Impossible d'allouer 32
canaux de mixage");

  // Un objet de type MixChunk est utilisé pour charger un son
  MixChunk monSon = SDLMixer.loadWAV("beep.wav");
  // Un objet de type MixMusix est utilisé pour charger une musique
  MixMusic maMusique = SDLMixer.loadMUS("earth.ogg");

  boolean Running = true;

  do
  {
   SDLEvent event = SDLEvent.waitEvent();

   if (event.getType() == SDLEvent.SDL_QUIT) Running = false;
   if (event.getType() == SDLEvent.SDL_KEYDOWN)
   {
    SDLKeyboardEvent keyEvent = (SDLKeyboardEvent) event;

    switch (keyEvent.getSym())
    {
    case SDLKey.SDLK_ESCAPE: Running = false;   break;
    case SDLKey.SDLK_KP1:  SDLMixer.playChannel(-1, monSon, 0);
break;
    case SDLKey.SDLK_KP2:  SDLMixer.playMusic(maMusique, 0); break;
    case SDLKey.SDLK_KP3:  SDLMixer.pauseMusic();   break;
    case SDLKey.SDLK_KP4:  SDLMixer.resumeMusic();   break;
    case SDLKey.SDLK_KP5:  SDLMixer.haltMusic();   break;
    }
   }
  } while (Running);
```

```
        screen.freeSurface();
                    // Libération des sons et musiques de la mémoire
                    SDLMixer.freeChunk(monSon);
        SDLMixer.freeMusic(maMusique);
        SDLMixer.close();
        SDLMain.quit();
      }
    }
```

Certaines parties vous ont fait peur ? Mais non ! Vous allez voir c'est très simple, étudions un peu ce code :

Code : Java

```
SDLMain.init(SDLMain.SDL_INIT_VIDEO | SDLMain.SDL_INIT_AUDIO);
```

Ici rien de bien nouveau, on utilise le « pipe » (opérateur OU bits à bits) pour ajouter une constante à l'initialisation, ici la vidéo et le son.

Code : Java

```
SDLMixer.openAudio(44100, SDLAudio.AUDIO_S16SYS, 2 , 1024);
```

Voilà enfin un peu ~~d'action~~ de nouveauté. On utilise cette méthode (qui est **`static`** pour changer) pour initialiser SDLMixer, car on a dit à SDL d'initialiser la partie audio, mais nous n'avons pas initialisé la bibliothèque tierce SDL_mixer. La méthode `openAudio()` prend 4 paramètres de type entier `int` (rien que ça) qui sont :

La fréquence : suivant la qualité de vos sons et musiques vous pouvez utiliser la fréquence que vous voulez. Il faut savoir que 44100 Hz est la fréquence utilisée par les CD audio.

Le format de sortie : suivant votre carte son vous pouvez utiliser les modes 8 et 16 bits. Vous noterez qu'il y a le suffixe SYS, qui correspond à Système. En fait suivant les systèmes et les cartes son, les formats de sorties sont codés en « little-endian » ou LSD (bit de poids faible) ou en « big-endian » ou MSB (bit de poids faible), le suffixe SYS prend le format de sortie du système sur lequel le programme s'exécute, vous n'avez donc pas à vous soucier de ça. Merci qui ?

Le nombre de canaux sonores : suivant votre configuration, 1 pour du son mono, 2 pour du stéréo, etc.

La taille du buffer de sortie : c'est la taille à allouer au buffer de sortie tout simplement, vous pouvez augmenter la valeur du buffer si vous sollicitez beaucoup les sons/musiques, une valeur de 1024 par défaut fonctionne dans la plupart des cas.

Code : Java

```
int result = SDLMixer.allocateChannels(32);
if (result != 32) throw new SDLException("Impossible d'allouer 32
canaux de mixage");
```

Cette méthode permet de choisir le nombre de canaux de mixage. On fixe généralement cette valeur à 32 avec les cartes son actuelles, mais si vous avez une Sound Blaster ou autre monstre de puissance sonore, vous pouvez augmenter cette valeur. On récupère dans une variable un code de retour. Si celui-ci est différent du nombre de canaux de mixage désiré, on lance une exception (cela veut bien entendu dire que l'allocation a échoué). Vous noterez que cette étape est facultative, en effet SDLMixer initialise lui-même avec des valeurs par défaut le nombre de canaux de mixage.

Code : Java

```
MixChunk monSon = SDLMixer.loadWAV("beep.wav");
MixMusic maMusique = SDLMixer.loadMUS("earth.ogg");
```

Vous devez vous en douter, il faut bien créer de nouveaux objets pour « contenir » nos sons et musiques. Les objets de type MixChunk permettent donc de stocker des sons aux formats wave et ogg. Les objets de type MixMusic permettent eux de « stocker » (une musique est un flux on ne la stocke pas vraiment) les musiques aux formats wave, ogg, mp3 et midi.

Les fichiers mp3 peuvent être longs à lancer (ce n'est pas une généralité).
Les fichiers midi sont mal gérés.

Code : Java

```
SDLMixer.freeChunk(monSon);
SDLMixer.freeMusic(maMusique);
```

Vous noterez qu'à la fin de la boucle principale, à l'endroit où habituellement on libère les ressources comme les images, on libère maintenant les sons et musiques, ce qui est totalement logique non ?

On finit avec les méthodes (oui vous le savez mais je le répète  `static...`) :

Code : Java

```
SDLMixer.playChannel(-1, monSon, 0);
SDLMixer.playMusic(maMusique, 0);
SDLMixer.pauseMusic();
SDLMixer.resumeMusic();
SDLMixer.haltMusic();
```

La méthode `SDLMixer.playChannel()` permet de jouer un son, elle prend 3 paramètres :

- un numéro de canal pour y jouer le son ;
- un objet de type `MixChunk` (donc votre son) ;
- le nombre de répétitions (loop).

Le numéro de canal a été fixé à -1, cela veut dire qu'on utilisera tous les canaux disponibles (dans notre exemple : 2 canaux).
La méthode `SDLMixer.playMusic(maMusique, 0)` permet de jouer une musique, elle prend en paramètre un objet de type `MixMusic` (votre musique), ainsi que le nombre de répétitions (loop).

Les autres méthodes parlent d'elles-même je pense, pas vous ? Vous constaterez quand même qu'on ne peut contrôler qu'une musique à la fois avec cette structure, ce qui est normal. Par contre je vous invite à consulter la documentation de SDL_mixer (version native) ainsi que celle de SDLJava pour voir le nombre de méthodes qu'il existe pour contrôler le son.

Vous pouvez par exemple :

Méthode	Effet
`haltChannel(int channel)`	Stoppe un canal en particulier (peut être pratique pour jouer des effets à gauche ou à droite).
`setMusicPosition(double position)`	Permet de vous « déplacer » dans la musique (avancer ou reculer ).

`volume(int channel, int volume)`	Permet de changer le volume général (son et musique).
`volumeMusic(int volume)`	Permet de ne changer le volume que des musiques (voir `volumeChunk(MixChunk chunk, int volume)` pour les sons).

Étant donné que le chapitre n'est pas très long mais qu'il y a des éléments essentiels, je vous propose de finir sur un QCM. Voilà vous savez tout. Nous allons dans le prochain chapitre écrire sur la fenêtre avec la bibliothèque SDL_ttf, vous verrez, c'est encore plus simple.

Du texte dans votre programme

Nous allons maintenant voir comment afficher du texte dans une fenêtre. Cela sera pratique pour afficher un menu, écrire un score ou afficher des informations diverses par exemple. Pour cela nous allons utiliser la bibliothèque SDL_ttf. Il vous faudra dans un premier temps procéder à l'installation de *SDL_ttf*, puis en dernier lieu télécharger des *fonts* pour les utiliser dans vos programmes.

Installation et préparation

Installation

Nous revoilà dans notre phase d'installation (c'est cool, non ?). Dans un premier temps direction le site officiel de SDL_ttf, pour y récupérer le paquet binaire contenant les fichiers dll dont nous avons besoin. Pour nos confrères Linuxiens les manipulations restent les mêmes. Copiez les fichiers dans votre répertoire de projet et vous êtes fin prêt pour commencer.

Pour résumer un peu, voici les nouveaux fichiers que vous devez avoir :

Sous Linux vous devez avoir les fichiers suivants :

- libSDL_ttf.a ;
- libz.a ;
- vous devez avoir freetype installé.

Suivant votre distribution et le type d'installation que vous avez faite de la bibliothèque SDLJava, vous n'aurez besoin que de libSDL_ttf.

Sous Windows vous devez avoir les fichiers suivants :

- SDL_ttf.dll ;
- libfreetype-6.dll ;
- zlib1.dll.

Préparation

Savez-vous ce qu'est un font ?

Un font est un fichier de police d'écriture, c'est exactement comme quand sous OpenOffice vous sélectionnez une police d'écriture : c'est ce qu'on appelle un font. Dans ce tutoriel nous allons utiliser la bibliothèque tierce TrueTypeFont, et grâce à un fichier font au format .ttf nous pourrons afficher du texte sur la fenêtre. Si vous désirez plus d'informations sur ce format je vous recommande de visiter la page de Wikipédia.

Bon c'est cool tout ça, mais les fichiers de font on les récupère où ?

Une recherche sur Google vous donnera bien des réponses (comme souvent), mais je vous propose un site qui est vraiment très complet, il s'appelle dafont et vous y trouverez vraiment tout ce que vous voulez comme style d'écriture ! Du moderne au gothique en passant par le techno. Pour nos exemples, j'en utiliserai trois différents pour que vous puissiez voir les différences entre plusieurs fonts.

Vous avez téléchargé votre font ? Vous êtes prêt ? Alors *let's rock* !

Afficher du texte

Je vous le dis tout de suite, un texte **est** une `SDLSurface` , donc nous utiliserons les même méthodes pour afficher du texte

que pour afficher une image ou autre chose. Voyons un peu de quoi nous allons avoir besoin dans notre code.

Les imports

Vous devez dans un premier temps importer deux nouvelles classes contenues dans le package `sdljava.ttf` .

`sdljava.ttf.SDLTTF` contient des méthodes (vous le savez pas vrai ? **`static`**) qui permettent d'initialiser la bibliothèque, de la fermer, d'instancier des objets de type `SDLTrueTypeFont` en chargeant des fonts, et quelques autres qui ne nous seront pas utiles (mais rien ne vous empêche d'aller jeter un coup d'œil à la documentation).

`sdljava.ttf.TrueTypeFont` va nous permettre de choisir le type de rendu, ainsi que la couleur du texte à afficher.

Déroulement des opérations

- Initialiser *SDL_ttf* ;
- Créer un objet de type `SDLTrueTypeFont` ;
- Créer une `SDLSurface` et l'instancier avec la méthode adéquate ;
- Afficher la `SDLSurface` dans la boucle principale ;
- Libérer la `SDLSurface` ;
- Libérer l'objet `SDLTrueTypeFont` ;
- Fermer *SDL_ttf* ;
- ... c'est fini !

Ça fait un paquet de choses à faire, mais c'est nécessaire. Voyons tout de suite quelques méthodes clés, puis passons à l'exemple. Nous commencerons dans l'ordre de la liste.

Pour initialiser SDL_ttf nous utiliserons la méthode (ok j'ai compris je m'en vais avec mes static) `SDLTTF.init()` . Pour la quitter nous utiliserons la méthode `SDLTTF.quit()` , les choses ne sont pas bien faites ? Exactement comme pour `SDLMain` .

Ensuite il faudra créer un ou plusieurs objet(s) de type `SDLTrueTypeFont` que nous initialiserons avec la méthode `SDLTTF.openFont()` . Intéressons-nous 30 secondes à cette méthode. `SDLTTF.openFont()` prend en paramètres : un fichier font (c'est le fichier .ttf que vous avez récupéré tout à l'heure), et un entier représentatif de la taille du texte. On pourra donc créer un objet font comme suit :

Code : Java

```
SDLTrueTypeFont monFont = SDLTTF.openFont("sega.ttf", 24);
```

À partir de là, vous êtes prêt à créer une `SDLSurface` qui va accueillir votre texte, et justement parlons-en. Pour créer un texte nous procéderons de cette manière :

Code : Java

```
SDLSurface monTexteSolid = monFont.renderTextSolid("Salut les zero",
new SDLColor(0, 255, 0);
```

La méthode `renderTextSolid(String texte, SDLColor couleur)` permet donc de créer une `SDLSurface` qui affichera votre texte, avec la couleur désirée. Vous noterez que pour la couleur j'ai utilisé la classe `SDLColor` , qui prend en paramètres trois entiers de 0 à 255, représentant votre couleur au format RGB. La classe `SDLColor` est à importer à partir du package `sdljava.SDLVideo` . Vous pouvez si vous le désirez créer des objets `SDLColor` avant, ce qui vous permettra dans le cas où vous avez à utiliser une couleur souvent de ne pas réécrire **`new`** `SDLColor(...)` .

Pour finir vous n'avez plus qu'à blitter avec la méthode `blitSurface()` .

Tu nous a parlé des types de rendu tout à l'heure, non ?

Oui, c'est vrai, et si votre éditeur comporte l'autocomplétion vous vous serez aperçu qu'il y a trois types de rendu de texte différents. Prenons un exemple :

Code : Java

```
SDLColor rouge = new SDLColor(255, 0, 0);
SDLColor vert = new SDLColor(0, 255, 0);
SDLColor bleu = new SDLColor(0, 0, 255);

SDLSurface monTexteSolid = monFont.renderTextSolid("Salut les
zeros", bleu);
SDLSurface monTexteShaded = monFontCute.renderTextShaded("comment ca
va", bleu, vert);
SDLSurface monTexteBlended =
monFontPlainBlackWide.renderTextBlended("???", rouge);
```

Comme vous le constatez, il y a 3 méthodes différentes pour choisir le type de rendu. La 1re est la plus simple, elle affiche du texte rapidement et simplement (pas de filtrage, rien), par contre la qualité n'est pas des meilleures. La deuxième par contre permet d'afficher le texte avec une couleur d'avant-plan (*foreground*) et une couleur d'arrière-plan (*background*) et est plus jolie, par contre elle est plus lente à afficher. Quant à la dernière elle est encore plus lente à afficher, mais c'est elle qui est la plus propre.

Mais qu'entends-tu par rapide, lent ?

Tout est relatif, mais un texte à rendu solid sera plus rapide à afficher qu'un texte à rendu blended. Faites donc le test avec `System.currentTimeMillis()` par exemple, mais de toute façon ça ne sera pas visible à l'œil nu (à part si votre programme ne fait qu'afficher du texte et surtout s'il en affiche abondamment !).

Et pour finir voici le code complet qui permet d'obtenir cette belle fenêtre  :

Code : Java

```
import sdljava.SDLMain;
```

```
import sdljava.SDLException;

import sdljava.video.*;
import sdljava.event.*;

import sdljava.ttf.SDLTTF;
import sdljava.ttf.SDLTrueTypeFont;


public class Affichage {

 public static void main(String[] args) throws SDLException {

  SDLMain.init(SDLMain.SDL_INIT_VIDEO | SDLMain.SDL_INIT_AUDIO);
  SDLSurface screen = SDLVideo.setVideoMode(320, 240, 32,
SDLVideo.SDL_HWSURFACE | SDLVideo.SDL_DOUBLEBUF);
  SDLVideo.wmSetCaption("Ecrivons avec sdl_ttf", null);

        // Initialisation de la bibliothèque SDL_ttf
        SDLTTF.init();
        // Ouverture d'un font
        SDLTrueTypeFont monFont = SDLTTF.openFont("data/sega.ttf",
24);
        SDLTrueTypeFont monFontCute =
SDLTTF.openFont("data/cute.ttf", 36);
        SDLTrueTypeFont monFontPlainBlackWide =
SDLTTF.openFont("data/plainblackwide.ttf", 36);


        SDLColor rouge = new SDLColor(255, 0, 0);
        SDLColor vert = new SDLColor(0, 255, 0);
        SDLColor bleu = new SDLColor(0, 0, 255);


        // Un texte à l'écran n'est qu'une SDLSurface
        SDLSurface monTexteSolid = monFont.renderTextSolid("Salut
les zero", bleu);
        SDLSurface monTexteShaded =
monFontCute.renderTextShaded("comment ca va", bleu, vert);
        SDLSurface monTexteBlended =
monFontPlainBlackWide.renderTextBlended("???", rouge);

  boolean Running = true;

  do
  {
   SDLEvent event = SDLEvent.waitEvent();

   if (event.getType() == SDLEvent.SDL_QUIT) Running = false;

   // On efface l'écran
            screen.fillRect(screen.mapRGB(0, 0, 0));

            // On blit ;)
            Util.blit(10, 10, monTexteSolid, screen);
            Util.blit(50, 100, monTexteShaded, screen);
            Util.blit(200, 200, monTexteBlended, screen);

   screen.flip();

  } while (Running);

        monTexteSolid.freeSurface();
        monTexteBlended.freeSurface();
        monTexteShaded.freeSurface();
        // Attention 2 nouvelles ressources à libérer en plus des
SDLSurface !
        monFont.closeFont();
        screen.freeSurface();
  SDLTTF.quit();
```

```
            SDLMain.quit();
        }

    }
    /**
    * Classe Util
    * Une classe à toujours avoir sous la main !
    */
    final class Util {
        public static void blit(int x, int y, SDLSurface source,
    SDLSurface destination) throws SDLException {
            SDLRect position = new SDLRect(x, y);
            source.blitSurface(destination, position);
        }
    }
```

Ce chapitre n'était pas très dur mais je vous l'avais dit. Faisons le point, vous devez normalement savoir :

- Installer SDLJava, ainsi que ses dépendances ;
- Écrire un programme à base de `SDLSurface` ;
- Gérer les événements avec le clavier ;
- Rendre votre application plus vivante avec du son et de la musique ;
- Afficher les bonnes informations au bon endroit.

Nous allons donc voir dans le prochain chapitre comment utiliser OpenGL pour le rendu graphique de votre application. Vous verrez qu'il n'y a pas grand-chose à changer.

Je ne ferai pas un cours sur OpenGL, pour ça vous avez plusieurs tutoriels disponibles sur ce site, dont celui de Kayl qui introduit OpenGL en C avec SDL.

Cependant j'aborderai certaines notions fondamentales, qui sont assez mal expliquées dans plusieurs tutoriels et que je me dois de vous exposer à ma façon (peut-être comprendrez-vous mieux).

Vous en voulez encore ? Eh bien il ne vous reste plus qu'à pratiquer, et à laisser libre cours à votre imagination. La consultation des annexes est vivement conseillée.

Partie 3 : Annexes

Dans cette partie, vous trouverez un complément indispensable à votre apprentissage de sdljava, mais aussi diverses ressources pour vos futures créations.

Se documenter avec SDLJava

Voici une petite partie sur la documentation et les exemples que vous pouvez trouver sur internet pour vous aider dans votre développement. Comme vous le verrez nous avons à notre disposition trois sources d'informations principales :

- La documentation C de SDL
- La Javadoc de sdljava
- Les exemples du dépôt CVS

La documentation Officielle SDLJava

Je vais vous présenter rapidement la documentation officielle qui n'est en fait que la javadoc. Il n'existe à ma connaissance aucun tutoriels traitent du sujet de sdljava en particulier, mais pas de panique ! Vous allez voir qu'avec sdljava c'est exactement comme SDL en C et même OpenGL ! Un tutoriel C/C++ fait l'affaire  bien sur il vous faudra chercher par vous même le comportement de certaines fonctions (par exemple le type `unsigned byte` n'existe pas en Java, donc il sera remplacé par un équivalent, etc...).

Le site officiel de sdljava

Alors rendez vous sur le site officiel, rubrique documentation. Vous aurez alors trois choix (c'est peux je sais )

- SDL API
- Javadoc API Docs
- Tutorials : How to setup Eclipse on Windows

Celui qui va nous intéresser pour le moment sera le deuxième : Javadoc API Docs. J'espère que la javadoc ne vous est pas étrangère, si c'est le cas il existe plusieurs ressources sur internet traitant du sujet. Nous avons donc acces à toutes les méthodes et constantes de sdljava, au passage vous remarquerez qu'il y a un module `sdljavax.gui` qui permet de créer des IHM.

Je vous présente rapidement le contenu des packages :

Package	Contenu
opengl	Pour un rendu avec OpenGL (contient GL, GLU, Glew, ftgl, etc...)
sdljava	Constantes d'initialisation ainsi que les Timer, mais aussi tous les autres packages
audio	Méthodes relatives à sdl_audio
cdrom	Méthodes relatives aux accès CD-ROM
event	Méthodes relatives aux événements avec les constantes etc...
image	Méthode `load()` pour charger les images
joystick	Gestion du joystick dans les événements
mixer	Méthodes relatives à sdl_mixer
ttf	Méthodes relatives à sdl_ttf
util	Permet de créer des buffers pour utiliser les utiliser avec des flux
video	Constantes et méthode pour la gestion de la vidéo
sdljavax.swig	Swig est l'outil qui permet de faire les binding
sdljavax.gfx	Méthodes pour faire des rotations etc... (sdl_gfx)

sdljavax.gui	Permet de créer des interfaces graphiques personnalisée avec OpenGL

Ouf ! qu'est ce que c'est lourd à lire pas vrai (mais efficace tout de même non ?) ! en plus nous ce que l'on aimerais c'est des exemples ! et bien vous allez être servie (enfin tout est relatif hein)

Le dépôt CVS de sdljava

Mais non ne partez pas ! Vous allez voir que c'est encore plus parlant, et que grâce à lui vous serez capable de vite vous débrouiller dans cette jungle qu'est la documentation. Donc rendez vous sur le dépôt CVS (que je vous conseil de mettre en marque page). Nous avons deux répertoires, un qui contient le site de sdljava (si, si) et l'autre qui contient les sources de sdljava, allez y justement, vous vous retrouvez à la racine des sources (wahoo, ça fait peur un peu...), pas de panique, voila comment ça se passe :

Répertoire	Contenu
bin	Des exemples compilés
docs	La documentation
etc	Divers fichiers de configuration
lib	Le jar de sdljava
src	Les sources du binding
testdata	Des tests que font les développeurs sur le binding pour voir si il est stable
testsrc	Le répertoire qui nous intéresse

Entrons dans ce nouveau répertoire et qu'avons nous la ? Encore des répertoires

Alors vous vous souvenez les packages que je vous aient décrit tout à l'heure ? et bien à chaque répertoire contient des sources d'exemple de ces packages ! sauf pour jsdl qui est l'ancien nom du binding, donc n'y allez pas (sauf si vous êtes très très curieux, mais ça ne sert à rien, puisqu'on ne s'en servira jamais on utilise le binding actuel).

Par exemple si vous voulez voir un exemple d'utilisation des méthodes qui permettent de gérer le CD-ROM, rendez vous dans /sdljava/testsrc/sdljava/cdrom et vous trouverez un fichier java `SDLCdromTest.java`. Vous n'avez plus qu'a examiner les sources pour comprendre comment elles fonctionne et les adapter dans votre programme.

La documentation Officiel SDL

Comme je vous l'est déjà signalé, les exemples C/C++ sont facilement transformables pour être utilisés avec java, et justement ce qui est bien, c'est que le site de SDL, contient une documentation assez bien faites (dont une en français s'il vous plais), ainsi que des exemples.

Dans un premier temps je vous recommande de consulter la documentation d'introduction à SDL en Français, vous pouvez d'ailleurs la télécharger à cette adresse.

Vous avez ensuite à votre disposition les tutoriels officiels SDL pour SDL, qui je vous le rappel sont facilement transformables, car bien que SDL soit programmée en C, le nom des fonction est dans la major partie des cas identique.

Et enfin Le wiki, qui contient des ressources, tutoriels, et j'en passe.

Je vous conseil aussi de consulter les documentations de :

- sdl_image
- sdl_mixer
- sdl_ttf
- sdl_gfx

Les autres ressources

Je vous propose maintenant quelques liens sur SDL, qui vous permettrons d'avancer encore un peu plus, par contre je vous

préviens, certains liens sont en anglais, et je pense que vous avez compris que comprendre l'anglais technique en programmation est essentiel quand on veut avancer.

Le tutoriel de M@teo21 : Création de jeux 2D en SDL (fr)

La section SDL du site developpez.com (fr)

La FAQ SDL du site developpez.com (fr)

Le site sdltutorials qui regorge de tutoriels et d'exemples en tout genre (us)

The game programming wiki : Vous y trouverez pas mal de ressources
Et bien voila qui clôture cette première annexe, j'espère que vous ne manquerez pas de documentations  . Si vous trouvez d'autres ressources intéréssentes qui auraient une place dans cette annexe, n'hésitez pas à me contacter par MP.

Installation d'une lib en dur dans la JVM

Dans ce tutoriel vous avez à chaque projet créer un répertoire lib qui contenais les fichiers jar et dll de sdljava, cependant il existe une autre méthode qui consiste à copier directement ces fichiers dans la JVM. Certains trouverons cette méthode inadaptée, d'autres au contraire la trouverons beaucoup plus pratique. Pour ma part je fonctionne désormais comme cela et pour chaque grosse lib. Le résultat est exactement le même, sauf qu'a la redistribution de votre application l'utilisateur devra soit lui aussi copier ces fichiers dans sa JVM (on peut par exemple utiliser un installeur qui le fait automatiquement), soit il faudra distribuer les fichiers à la racine du répertoire de l'application. C'est une méthode comme une autre et je vous la propose tout de suite. Cette méthode fonctionne bien évidement sur Linux et Windows.

Le répertoire de la JVM et du JDK

Localisation des répertoires

Petite piure de rappel :

JRE : Java Runtime Environment --> c'est la machine virtuelle java qui permet d'exécuter vos programmes
JDK : Java Development Kit --> Ce sont les outils qui permettent de compiler des fichiers java et faire d'autre choses

Sous Linux

Suivant votre distribution, vous aurez plusieurs possibilités, par exemple sur les dérivées de Debian, la JVM et le JDK, lorsqu'ils ont été installés via apt-get se trouvent dans `/usr/lib/jvm"`, pour d'autres distributions ce sera dans `/opt/`, vous devrez donc chercher vous même où vous avez installé la JVM ainsi que le JDK, vous noterez aussi que parfois la JVM et le JDK peuvent être dans le même répertoire, dans ce cas, l'installation sera plus rapide.

Pour ma part ils se trouvent dans `/usr/lib/jvm`, et comme je suis sur un système 64 bits j'ai plusieurs versions de java :

- ia32-java-6-sun-1.6.0.10 : la machine virtuel java 32 bits (JVM)
- java-6-openjdk : la machine virtuel avec le plugins de l'OpenJDK 64 bits (une autre JVM)
- java-6-sun-1.6.0.10 : ma JVM et mon JRE en 64 bits (dans le même répertoire)

Donc si vous n'avez qu'un répertoire ET que vous avez installé la JVM et le JDK, pas de panique, référencez vous aux explications pour l'installation sur le JDK.

Nous aurons à copier des fichiers dans ces répertoires, si vous travaillez en mode graphique, assurez vous d'être en mode super-utilisateur (par exemple `sudo dolphin` pour KDE 4 ou `sudo nautilus` pour gnome

Sous Windows

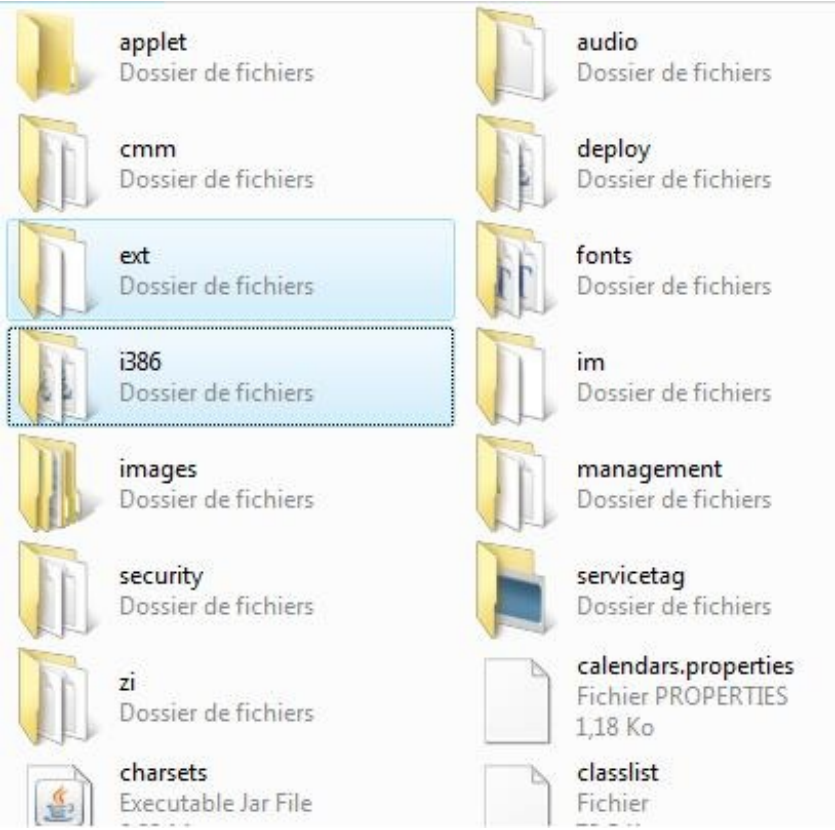
L'installation par défaut de Java est dans `c:\program files\java\` pour un système 32 bits et dans `c:\program files(x86)\java\` pour un système 64 bits. Vous aurez normalement deux répertoires qui seront d'une part celui de la JVM nommé jre6 et d'autre part, celui du JDK nommé suivant sa version jdk1.x_u.

Structure des répertoires

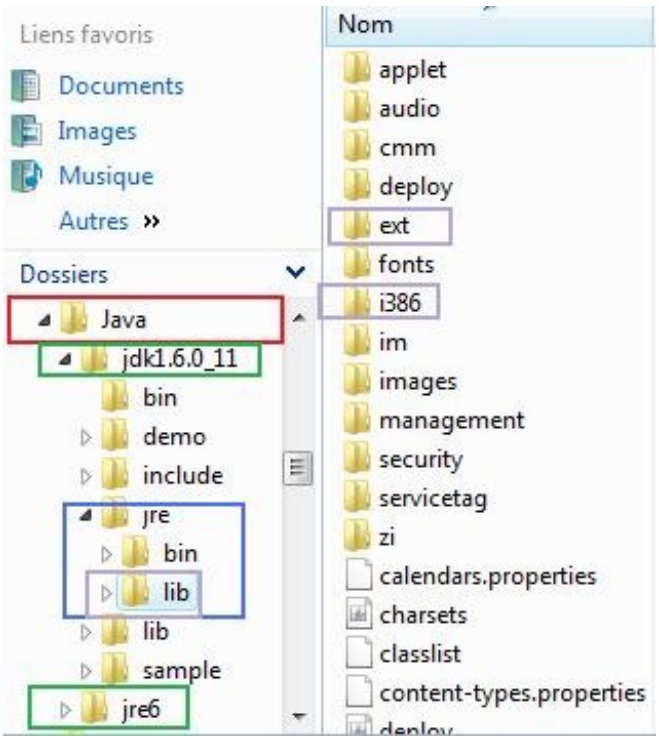
Ce qui suis est valable aussi bien sous Linux que sous Windows et fonctionne avec n'importe quelle version de Java, même avec OpenJDK.

Nous commencerons par analyser le répertoire de la JVM, il contient deux répertoires (bin et lib). Le répertoire bin contient les fichiers exécutables de java tandis que lib contient les bibliothèques, ressources et autres fichiers de configurations. Nous nous

intéresserons uniquement au répertoire lib pour la JVM. Rendez vous dans lib et localisez les répertoires **ext** et **i386**



Laissez cette fenêtre de coté, nous allons faire de même avec celle du JDK. Rendez vous ou vous avez installé le JDK et listez son contenu. Il y a beaucoup plus d'éléments que pour la JVM et c'est bien normal, la JVM sert à exécuter le bytecode, alors que le JDK sert à compiler ET exécuter le bytecode (et à faire aussi d'autres choses, mais c'est hors de ce tutoriel). Le JDK permet comme je vous l'ai dit d'exécuter du bytecode, donc il contient lui aussi une JVM ! Et bien figurez vous qu'en sélectionnant le répertoire jre vous tomber... sur un nouveau jre avec les même répertoires, donc vous savez lesquelles utiliser 



Copie des fichiers

Maintenant, il n'y a rien de compliqué, vous devez copier :

Tous les fichiers `*.jar` dans les répertoires `ext`

Tous les fichiers `*.so ou *.dll` dans les répertoires `i386`

Une fois que cela est fait, votre installation en dur est terminée, vous pouvez créer un nouveau projet dans eclipse, en sélectionnant bien le jdk qui est installé sur votre machine (et pas le compilateur intégré à Eclipse), vous n'aurez alors qu'a copier les fichiers dll relatif à la lib SDL native à la racine de votre projet. Aprés compilation et lancement de votre projet  plus de répertoire lib.
Voila qui est fait ! Si vous avez plusieurs projets à gérer en même temps cette méthode se révélera bien plus pratique.

Le tutoriel **n'est pas** terminé, n'hésitez pas à revenir de temps en temps pour voir son évolution, et si vous avez des questions, n'hésitez pas à les poster sur le forum Java. Un grand merci à la zCorrection, ainsi qu'aux personnes qui m'ont encouragé à rédiger ce tutoriel.