RISA 2011

Reduction d'Image et de Spectres d'Astrophysiques

Rapport de Projet
Master Physique et informatique
Université Montpellier 2

Etienne Gibaud / Emmanuel Hermellin
Juin 2011

Sommaire :

Présentation Générale :

A. Sujet du Projet :

L'objet du stage 2011 est d'abord de faire atteindre à RISA un niveau professionnel qui nous permettra de le diffuser de manière publique. Le souhait est de faire de RISA un logiciel pédagogique au service de l'enseignement de l'astrophysique au sein de différentes universités. Le travail demandé porte en grande partie sur une refonte de l'architecture actuelle du logiciel. La nouvelle architecture devra offrir la modularité, la maintenabilité et la flexibilité attendues pour tout outil d'analyse pouvant être partagé par l'ensemble d'une communauté.

Ainsi, RISA ne fonctionne bien aujourd'hui qu'au sein de l'environnement Windows alors qu'il a été développé avec le langage Java pour justement faciliter sa portabilité (Linux, MacOS). Ensuite, l'organisation actuelle du code ne permet pas de le faire évoluer de manière souple et ainsi rend difficile la correction des problèmes persistants ou encore la prise en charge de nouvelles fonctionnalités. Il faut donc retravailler les notions du développement orienté objet pour répondre à ces limites. Il est attendu d'exploiter différentes librairies Java dédiées comme la librairie de la Nasa nom.tam.fits ou encore la librairie d'ImageJ pour rendre efficace l'écriture du code. Le code devra également s'affranchir d'erreurs d'exécution. Dans ce sens, les mécanismes de gestion d'exception seront révisés et testés sur des exemples concrets. Le modèle conceptuel précisant les classes Java à développer sera fourni par les encadrants.

Une partie d'exploitation physique est incluse. Elle consistera par exemple en la réalisation d'une étude photométrique d'étoiles d'un amas avec RISA de données acquises au télescope.

B. Présentation de l'université et du laboratoire :

1. L'université Montpellier 2 :

L'UM2 est une Université de Recherche Intensive dont les formations et les activités de recherche couvrent l'ensemble des champs scientifiques et technologiques : biologie fondamentale, biologie appliquée et écologie, chimie, sciences de l'ingénieur, sciences de l'univers, physique et mathématiques, gestion, sciences de l'éducation. L'offre de formation scientifique, technologique et managériale y est diversifiée (L.M.D). Le dynamisme et le rayonnement international de ses laboratoires, ainsi qu'une politique volontariste de transfert vers le monde de l'entreprise sont autant d'atouts pour contribuer au développement socio-économique de la région et à l'insertion professionnelle de ses étudiants.

Environ 3000 personnels titulaires participent à la vie de l'établissement.

L'université accueille près de 15 000 étudiants répartis au sein de 8 composantes de formation.

L'Université est associée au travers d'une cinquantaine de laboratoires à la plupart des grands organismes de recherche français, au cœur de la cinquième concentration de forces de recherche française. Montpellier est un pôle d'excellence à vocation mondiale, en particulier dans le domaine des Sciences de l'Environnement, dont la situation géographique unique sur l'arc méditerranéen en fait un carrefour entre l'Europe et les pays du sud.

Une reconnaissance internationale :
L'Université Montpellier 2, Sciences et Techniques, est reconnue au niveau international comme étant parmi les meilleures universités européennes pour la qualité de son enseignement et de sa recherche, ce que confirme sa place dans les classements internationaux récemment publiés.

Dans le classement mondial de Shanghai publié en août 2010, l'UM2 est 8ème au rang national derrière Paris 6, Paris 11, l'École Normale Supérieure, Paris 7, Strasbourg, Grenoble 1 et Paris 5 et 3ème hors région parisienne (derrière Strasbourg et Grenoble 1).

Depuis peu, un nouveau classement international basé sur des critères de recherche, le SCImago Institutions Classement (SIR), effectue le classement de plus de 2000 institutions de recherche parmi les meilleures au monde. Ce classement comprend plusieurs indicateurs de performance de la recherche tels que : les publications et leur visibilité, la collaboration internationale et l'impact. La France est classée première avec le CNRS. L'Université Montpellier 2 apparaît dans le SIR 2009 World Report, en meilleure position que dans le classement de Shangaï : à la 7ème place des universités françaises et à la 3ème place des universités hors région parisienne, après Lyon 1 et Strasbourg 1.

2. Le Laboratoire Univers et Particules de Montpellier :

Le Laboratoire Univers et Particules de Montpellier créé le 1er janvier 2011 est issu de la fusion de deux laboratoires. Il réunit l'ensemble des membres du Groupe de Recherche en Astronomie et Astrophysique du Languedoc (GRAAL) et une partie des membres du Laboratoire de Physique Théorique et Astroparticules (LPTA). Le LUPM est composé d'une soixantaine de personnes.

Le laboratoire est organisé autour de trois équipes de recherche :

- Astrophysique Stellaire (AS),
- Expériences et Modélisations en Astroparticules (EMA),
- Interactions Fondamentales, Astroparticules et Cosmologie (IFAC).

Quatre services de soutien à la recherche complètent sa géographie : administration, communication & documentation, informatique, instrumentation.

Equipes de recherche :
La recherche au LUPM est structurée en trois équipes autour des thématiques : particules, noyaux, astroparticules, astrophysique, cosmologie et physique théorique. Les équipes Expériences et modélisation en Astroparticules (EA), Astrophysique Stellaire (AS) et Interactions Fondamentales, Astroparticules et Cosmologie (IFAC) travaillent sur des sujets d'intérêts communs :

- étoiles massives, progéniteurs de sursauts gamma et vestiges de supernovae, univers primordial (lithium, premières étoiles), matière sombre,
- rayonnement cosmique,
- chimie du milieu interstellaire.

3. Les Encadrants :

Dans le cadre du Master 1 de Physique Informatique, nous sommes amenés à travailler sur le projet RISA (Réduction d'Images et de Spectres Astronomiques) sous la tutelle de trois professeurs :

CORDONI JeanPierre: Maître de Conférences au Laboratoire du Groupe de Recherche en Astronomie et Astrophysique du Languedoc (GRAAL).

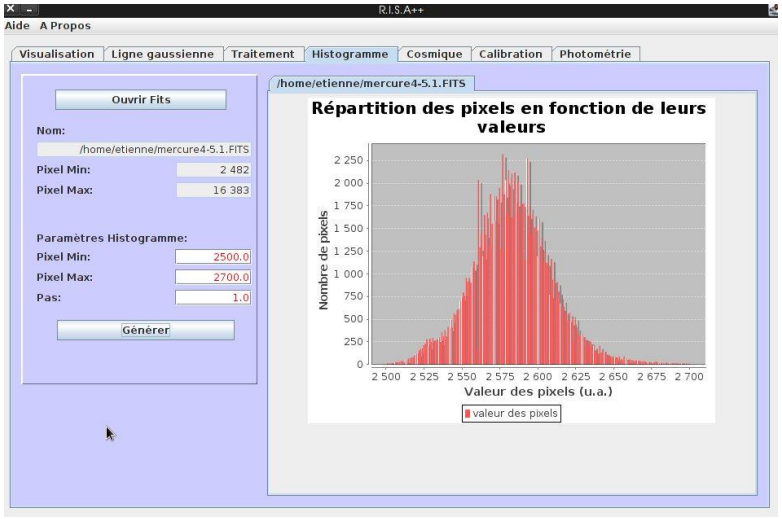
MOUGENOT Isabelle : Maître de conférences au Laboratoires d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM).

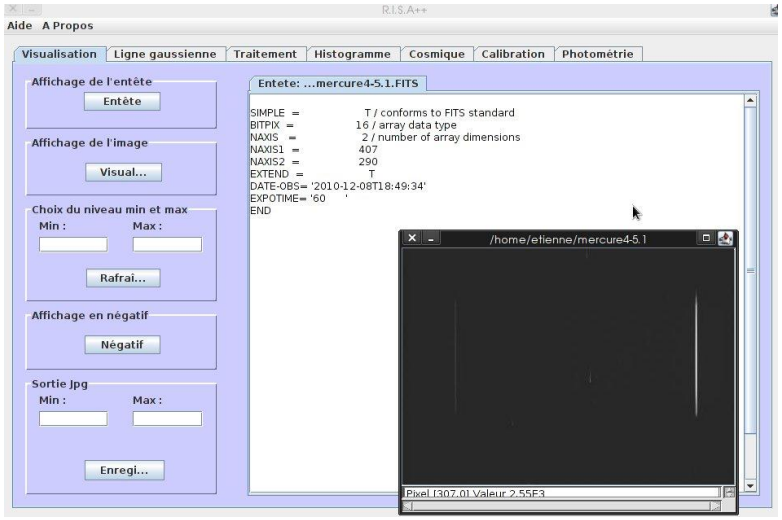
REBOUL Henri : Maître de Conférences au Laboratoire du Groupe de Recherche en Astronomie et Astrophysique du Languedoc (GRAAL).

4. Présentation du logiciel RISA :

Il existe des logiciels professionnels (MIDAS, IRAF, …) pour la réduction des données astronomiques recueillies aux foyers des télescopes ou de leurs instrumentations auxiliaires. Ils sont d'un apprentissage très lourd et masquent souvent la physique de la réduction de données. C'est pour pallier ces deux inconvénients qu'avait été réalisé à l'UM2, avec Christian Sam, stagiaire d'IUT informatique, en 1995 le logiciel IRIS (Initiation à la Réduction d'Images et de Spectres) qui tournait sous DOS (puis sous Windows sous sa forme Iris-fv) dans les salles de TP du CR2I puis de l'UFR. Il utilisait le format professionnel FITS, les algorithmes professionnels précis mais, pédagogique, son interface posait des questions physiques à l'utilisateur. Depuis 2007 RISA reprend et amplifie cette facilité.

RISA (**R**éduction d'**I**mages et de **S**pectres **A**strophysiques) est un logiciel créé par Vannak Kol, Alexandre-Florent Gonthier et Julien Horte lors de leur stage de M1 Phy-Info en 2007. Le cahier des charges était de reproduire les fonctionnalités de base de l'ancien « IRIS-fv » mais avec les nouvelles possibilités que permettaient le fenêtrage et l'indépendance du système d'exploitation et de rajouter de nouvelles fonctionalités notamment en photométrie. Le travail a été complété en 2008 par Hervé Dauberton, Jenny Bravo et Julien Roupie avec encore l'aide d'Alexandre-Florent Gonthier. En 2009 c'est Mounir Bensadia, Serge-Henri Cachia et Boualam Hasnoun qui ont poursuivi ce travail pour corriger quelques bugs et développer la partie « photométrie » (mesures sur images). En 2010 Donia Aissa, Nicolas Glaizette et Karim Bammou ont repris un certain nombre de dysfonctionnements dans le traitement des données et de leur affichage, d'amélioration de l'interactivité et développé l'aide en ligne. RISA, en versions d'essai sert depuis 2007 pour la réduction des observations en M1. Il reste encore quelques bugs dans le traitement de certains fichiers et des « incommodités ». Par ailleurs la partie photométrie n'a pas été testée en vraie grandeur avec les étudiants (coupole Est en cours de rénovation).





Analyse de l'existant :

A. Le Projet en 2010 :

RISA est un projet qui existe depuis plusieurs années. Certains choix de programmation ont été faits (faire une interface complète dès la première année, ne pas utiliser de librairie standard) au détriment de la qualité du code et du bon fonctionnement du logiciel. Nous héritons donc d'un logiciel fonctionnel mais fortement instable et contenants de nombreux bugs dont certains provoquant la fermeture brutale de RISA.

Les fonctions de RISA permettent de :

- visualiser une image au format Fits et son entête.
- traiter un fichier Fits avec les lignes gaussiennes.
- réaliser diverses opérations (mathématiques, arithmétiques, de convolution, d'extraction et de symétrie sur les images Fits.
- visualiser un fichier Fits sous forme d'histogramme.
- visualiser une courbe de calibration d'un fichier Fits.
- réaliser un travail de photométrie.

B. Les limites du logiciel :

Comme nous le disions, des choix de programmations ont été fait. Un des choix les plus handicapant pour RISA est d'avoir utilisé l'interface de developpement NETBEANS qui, même si il facilite grandement la programmation d'interface en JAVA, rend le code généré très complexe et lourd. Se plonger dans des lignes de codes réalisées sous cet environement devient un véritable calvaire.
Il en résulte que la classe interface comporte environ 8500 lignes de codes qui ont permis de construire une interface de façon intuitive. Cette classe comporte, en plus de l'interface, du code « calculatoire » et est par conséquent difficilement lisible, ce qui rend la maintenance et l'amélioration de RISA difficile. De plus des méthodes liées à l'interface sont présentes dans de nombreuses autres classes.

Deuxième problème assez important est que la manipulation des fichiers FITS ainsi que le traitement des images a été conçu à la main alors qu'il existe des librairies spécifiques qui offrent des performances beaucoup plus grandes et une meilleure gestion des entêtes FITS en particulier. En effet, certains fichiers traités avec RISA ne s'ouvrent plus avec ce dernier à cause d'un problème de sauvegarde des entête (du à la position du dernier bit de l'entête qui est fixé en dur dans le code). Ce choix coupe RISA d'une grande communauté de programmeurs,qui font évolué jour après jour les librairies généralistes (ImageJ et TAM dans notre cas) mais aussi d'une possibilité d'évolution de ses fonctions.

Enfin l'un des aspects de RISA qui limite son utilisation est l'impossibilité de l 'éxécuter sur d'autre plateforme que Windows. Linux et MacOS ne sont pas supporté par le logiciel de par son developpement sous NETBEANS mais aussi de par la différence des chemins de fichies sous ces differents OS. Exemple :

Chemin d'un fichier sous une distribution linux.

Chemin d'un fichier sous windows.

Problématiques et solutions :

A. Les objectifs en 2011 :

Bien qu'étant fonctionnelle, l'application RISA souffre d'une architecture complexe et souffrant de nombreux problèmes. Notre principal objectif en 2011 est de développer une nouvelle version de RISA, prenant en compte les contraintes et les besoins du présent, mais aussi ceux à venir.

Cette idée se décompose en plusieurs points sur lesquels nous travaillerons pour faciliter l'expérience RISA, côté utilisateur et côté développeur, notre travail est donc de lancer RISA dans une direction plus saine pour les futurs développeurs amenés à modifier le code source de l'application. Certains de ces objectifs sont uniquement envisagés et ne seront peut-être qu'accomplis dans les années à venir, cependant la ligne directrice de notre travail se base sur les décisions suivantes :

- Evaluer les besoins des physiciens ou étudiants physiciens en contact avec RISA.
- Restructurer entièrement le code source : Mise en place d'une architecture logicielle plus adaptée (orientée objet) et utilisation des librairies nom.tam et ImageJ.
- Produire un code source lisible et clair, conforme aux conventions Sun, et éventuellement mettre en place une API Javadoc.
- Développer une nouvelle interface modulable et exempte de bugs.
- Favoriser l'interopérabilité.

En bref, l'idée principale est de professionnaliser le développement du logiciel et de permettre une utilisation de celui-ci sur tous les systèmes, en limitant les défauts au maximum.
Pour celà, nous souhaitons produire un code source rigoureux et bien pensé pour permettre la modification, l'amélioration et tout simplement l'accès aux sources du logiciel.

B. Nos Recherches :

La librairie Tam :
La librairie nom.tam.fits est une librairie Java développée par la NASA et qui permet de lire et écrire les fichiers FITS.

Les fichiers FITS sont des fichiers couramment utilisés par les astrophysiciens pour conserver et partager leurs données expérimentale de façon standard. Cette librairie est particulièrement adaptée à notre problème puisqu'elle gère de façon simple les en-têtes et les images des fichiers FITS, ce qui nous permet de manipuler ces fichiers à l'intérieur de classes spécialisés utilisant les fonctionnalités de tam.

La librairie ImageJ :

La librairie ImageJ, comme son nom l'indique, permet la manipulation d'images. Elle a été développée par un organisme gouvernemental américain (National Institute of Health). Elle offre de nombreuses fonctionnalités dans l'analyse et le traitement d'image :

- Visualisation et ajustement des niveaux de gris.
- Débruitage.
- Transformation de Fourier directe et inverse.
- Seuillage, opérations logiques et arithmétiques entre images, etc.

Bien qu'initialement conçue pour des applications biomédicales, ses fonctionnalités se sont grandement enrichies et l'utilisation de nombreux format est maintenant possible (notamment FITS).
Nous avons choisi ImageJ pour plusieurs raisons, tout d'abord, son efficacité est indéniable, et de plus le fait que cette librairie soit libre correspond bien à l'esprit universitaire et garanti une technologie pérenne. Enfin, ImageJ a été développé avec un principe fondateur : la modularité, c'est pour cela qu'il existe de nombreux plug-ins – plus d'un centaine – qui augmentant les possibilités au fil du temps.

L'interface graphique :

L'interface graphique originelle de RISA est assez complexe, et à été développer sans prendre en compte la possibilité d'une refonte totale du cœur.
Bien que la réécriture de l'interface graphique en elle-même ne fasse pas partie du sujet qui nous est proposé, nous avons décidé de commencer à développer une base pour les futurs étudiants, qui pourraient la finaliser dans les années à venir.

C. Problèmes rencontrés :

RISA :

Ce projet nous a permis de comprendre beaucoup de choses à propos de la programmation objet et du langage Java. En effet, lors de la première approche à ce projet, il nous a tout d'abord été difficile de trouver la direction à prendre pour notre travail :

- Faut-il diriger notre travail vers une remise à zéro du projet ?
- Faut-il nettoyer et ré-adapter le code source ?

Après avoir regardé pendant un certain temps le code de RISA, nous nous sommes rendu à l'évidence : il serait quasiment impossible de réutiliser l'interface graphique de l'ancien RISA. Globalement, la conception objet n'était pas vraiment respectée ou optimisée, et nous avons pensé que passer du temps à adapter cette interface, la corriger au fur et à mesure nous prendrait finalement beaucoup trop de temps.

L'interface :

En effet m'interface a été une source de problèmes importante.Les enseignants voulant garder la même interface avec le nouveau logiciel, nous avons décidé de découpler l'ancien GUI du code source pour le réutiliser ensuite. Cepedant l'ancienne interface a été écrite en utilisant un générateur de code (Netbeans) qui ne permet pas réellement de construire le programme en conservant une philosophie objet et en créant des classes très lourdes et incomprehensibles.
Nous avons donc décidé de repartir de zéro, et de faire notre propre interface, de façon propre sans générateur automatique de code, pour remettre le projet sur de bonnes bases, pour que ce logiciel soit relativement facile à maintenir.

Java :
En ce qui concerne Java, il a été difficile pour nous de démarrer le projet : nous avons reçu une formation de base sur Java, faisant travailler quelques classes Java de concert, modifiant des données. Nous n'avons pas eu tout de suite les réponses à nos questions :

- Comment bien concevoir un logiciel ?
- Comment optimiser la programmation, et l'organisation du code ?
- Que sont les exceptions, et comment les gérer ?
- Comment collaborer sur le projet ?

Il nous à donc fallu du temps pour prendre le recul nécessaire à la création d'un logiciel à proprement dit.

Les Libraires :
Un de nos premier obstacle à été de comprendre comment utiliser les librairies tam et ImageJ. En effet, avant ce projet, nous n'avions jamais utilisé de librairies en connaissance de cause.

Tam : En ce qui concerne la librairie Tam, elle nous mis en défaut pendant un temps assez long : il nous fallait comprendre le format FITS (Voir Annexe pour mieux le comprendre), et ensuite comprendre comment les développeurs de cette librairie ont choisi d'orienter la façon dont elle gère ces fichiers. Une fois que nous avions saisi le fonctionnement général de Tam, nous avons pu mettre en place nos premières briques de gestion des fichiers.

ImageJ : Pour ImageJ, il nous à été très difficile de comprendre son fonctionnement, puisque le contenu sur internet, est relativement tourné vers le côté « Application » d'ImageJ, et pas vers son côté « librairie java » en tant que telle. Nous avons donc finalement dû demander à nos encadrants de nous donner quelques indices et premiers exemples sur son fonctionnement, car au départ cette librairie paraît très complexe puisque très complète.

Les Exceptions :
Les exceptions sont un domaine très important de Java, permettant de continuer l'exécution lorsqu'un utilisateur amène le programme à manquer d'information pour construire les données. Le concept d'exception en Java ne semble pas être un domaine facile d'accès, et de notre point de vue nécessite un apprentissage long et une certaine expérience de la programmation objet en Java. Nous avons donc eu du mal à comprendre totalement ce concept particulier, mais nous avons réussi à appréhender certaines de ses utilisations que nous avons pu intégrer au programme.

Conception du logiciel :
RISA n'est pas un logiciel fondamentalement compliqué à écrire. Cependant, il nous à fallu du temps pour appréhender le fonctionnement en Java d'un tel logiciel, et donc de trouver une architecture optimisée et fonctionnelle. Ce fût un long travail de prise de recul, de tests, de recherches d'indices dans les documentations pour arriver à conceptualiser les classes du programme, nous permettant aussi d'en apprendre beaucoup sur Java et son fonctionnement.

D. Solutions logicielles & logistiques :

Eclipse :
Nous avons décidé d'utiliser le logiciel Eclipse car c'est un environnement de développement particulièrement adapté à Java, et permet une utilisation relativement simple de Subversion (SVN). Cela nous permettra donc de développer dans les meilleurs conditions, de façon à pouvoir collaborer le plus efficacement possible. De plus, il est important de noter que c'est un des environnement de développement les plus utilisé dans la communauté de programmation. Enfin, Eclipse et « The Eclipse Foundation » sont une initiative réellement orientée vers le logiciel libre, non-lucrative et basée sur le soutient de ses membres, qui pour nous une philosophie qui correspond à celle d'un logiciel universitaire tel que RISA, autant dans sa diffusion que de son architecture et sa documentation.

SVN :
Nous avons choisi d'utiliser Subversion (SVN), car il nous semblait indispensable de trouver une façon de travailler en collaboration sans pour autant devoir se retrouver physiquement. Nous nous sommes donc, encore une fois, orienté vers une plate-forme gratuite, open source et assez intuitive. De plus, il permet énormément de possibilités au niveau du versioning, et de la gestion de l'historique du code, malgré que cela puisse le rendre parfois complexe à appréhender.

* L'adresse du svn : svn://owilab.org/RISA2011

MediaWiki :
Après avoir mis en place les éléments précédents nous permettant de travailler efficacement, nous avons finalement décidé de créer un Wiki pour RISA, de manière à pouvoir écrire concrètement, nos remarques, nos objectifs, notre planning, l'avancement du projet.
C'est aussi une façon de laisser aux futurs développeurs de ce logiciel, une orientation, ou tout du moins une mise en situation du projet et du code source.
Nous avons choisi la solution MediaWiki en particulier pour les mêmes raison que précédemment, c'est-à-dire son caractère libre, gratuit et simple d'accès.

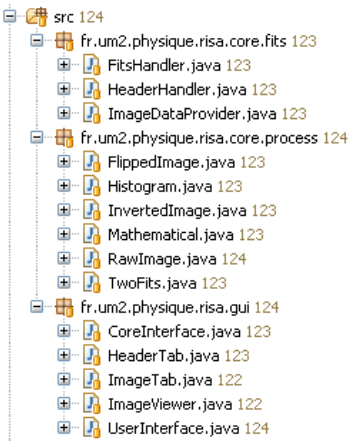
* L'adresse du wiki : http://risa.owilab.org

Notre Travail :

A. Les Packages

Pour notre programme nous avons commencé par établir une architecture de base, à partir des packages. C'est une façon de faire une première séparation, une hiérarchie entre les classes, en fonction de leur utilisation.

Nous avons choisi une organisation relativement simple :



- Package « core »
  - sous-package « fits »
  - sous-package « process »
- Package « gui »

1. Le package « core »

Le sous-package « fits »

Ce package contient les classes qui utilisent la librairie nom.tam.fits dont nous avons parlé plus tôt, elles sont au nombre de trois :

- FitsHandler
- HeaderHandler

Il est à noter qu'il y a aussi une troisième classe : ImageDataProvider, que nous avons laissée vide pour l'instant. La classe FitsHandler est une classe simple qui permet d'instancier la classe Fits avec le fichier voulu :

Classe FitsHandler

```
public FitsHandler(File file) throws FitsException {
	try {
		Fits srcFile= new Fits(file);
		this.header = new HeaderHandler(srcFile.read([0].getHeader());

	} catch (FitsException e) {
		//e.printStackTrace();
		throw e;
	}
}

	public HeaderHandler getHeader() {
	return header;
}
```

Ici, on voit que la classe instancie aussi la classe HeaderHandler. Cette dernière classe permet la gestion de l'en-tête des fichiers fits, à l'aide de la librairie tam. Elle construit et retourne une « Map » Java contenant les paires clé/valeur de l'en-tête :

Classe HeaderHandler :

```
public HeaderHandler(Header rawHeader) throws IllegalArgumentException {
        if (rawHeader == null) {
                throw new IllegalArgumentException("Header null");
        }

        Cursor cursor = rawHeader.iterator();
        while (cursor.hasNext()) {

                HeaderCard hCard = (HeaderCard) cursor.next();
                if (hCard == null)
                        continue;
                if (hCard.getKey().equals("END")) {
                        continue;
                }
                headerMap.put(hCard.getKey(), hCard.getValue());
        }


public Map<String, String> getMap() throws NullPointerException {
        try {
                return headerMap;
        }
        catch (NullPointerException npe) {
                npe.printStackTrace();
                throw npe;
        }
}
```

Ces deux classes permettent donc de gérer les fichier fits et leurs en-têtes. Nous avons choisi de créer une Map pour contenir les informations de l'en-tête car cela nous paraissait être le format le plus « basique » à retourner, qui pourrait être éventuellement traité dans une autre classe.

La classe ImageDataProvider est pour l'instant vide, elle était au départ destinée à accueillir le traitement des données binaires de l'image à partir de la librairie tam, mais ImageJ nous permettant de gérer et modifier l'image directement. Nous avons choisi de la laisser pour l'instant, ne sachant pas si la meilleure solution est de traiter l'image uniquement grâce à ImageJ.

Le sous-package « process »

C'est dans ce sous package que toutes les opérations effectuées sur les images Fits sont effectuées.
Il contient plusieurs classes permettant de renvoyer une ImagePlus grâce à des accesseurs, ImagePlus est en fait une classe d'ImageJ, qui est une image que l'on peut afficher :

- RawImage.
- InvertedImage.
- FlippedImage.
- Mathematical.
- Histogram.
- TwoFits.

Classe RawImage :

```
public class RawImage {
	ImagePlus imp;
	String type = "Raw";

	public RawImage() throws NullPointerException {
		imp =new ImagePlus(CoreInterface.getFile().getAbsolutePath());
	}

	public ImagePlus getImage() {
		return imp;
	}

	public String getType() {
		return type;
	}
}
```

Comme nous pouvons le voir ci-dessus, notre classe RawImage, crée une instance d'ImagePlus, à partir du fichier Fits sans la modifier.

Classe FlippedImage :

```
public class FlippedImage {
	ImagePlus imp;
	String type = "Flip";
	ImageProcessor imgpr;


	public FlippedImage()  throws NullPointerException {
		imp =new ImagePlus(CoreInterface.getFile().getAbsolutePath());
		imgpr = imp.getProcessor();
		}
}

public ImagePlus getVertFlipImage() {
	imgpr.flipVertical();
	type="Vertical " + type;
	return imp;
}
```

Nous voyons ici que la classe FlippedImage est très similaire à RawImage, sauf qu'un ImageProcessor est crée, c'est un objet qui permet de modifier une image donnée. Nous nous en servons donc dans cette classe pour effectuer des symétries ou des rotations de l'image.

Ainsi, la méthode getVertFlipImage() n'est pas la seule présente, et à chaque type de rotation, il y a une méthode correspondante où le type est changé, et l'image modifiée de la façon souhaitée.

Nous noterons que les autres classes sont quasiment identiques : InvertedImage produit le négatif de l'image, Mathematical, propose les opérations arithmétiques avec constantes sur l'image. Pour cela on utilise de nombreuses méthodes du ImageProcessor qui nous permettent d'obtenir les images désirées.

Classe Mathematical :

Cette classe est destinée à effectuer toutes les opérations mathématiques sur les fichiers FITS et à renvoyer l'ImagePlus correspondante.

```
public Mathematical() throws NullPointerException {
	try {
		imp =new ImagePlus(CoreInterface.getFile().getAbsolutePath());
		imgpr = imp.getProcessor();
	} catch (NullPointerException npe) {
		npe.printStackTrace();
		throw npe;
	}
}
```

Le constructeur de Mathematical, créant l'ImagePlus et l'ImageProcessor. Pour l'addition d'une constante, nous avons cette méthode :

```
public ImagePlus getAddImage(double value) {
	imgpr.add(value);
	type = type + " Add " + value;
	return new ImagePlus(type, imgpr.convertToFloat());
}
```

On créer une ImagePlus à partir de l'ImageProcessor, que l'on convertit en flottant 32-bit à la fin, puisque la méthode add(...) pour un ImageProcessor ne fonctionne pas si nous le faisons avant. Toutes les autres fonctions mathématiques sont effectuées de la même manière.

Classe TwoImages :

Cette classe permet l'addition et la soustraction de deux fichiers FITS différents et de les visualiser. Elle renvoie l'ImagePlus correspondante.

```
public TwoImages(File file) {
	secFile = file;
	imp1 = new ImagePlus(mainFile.getAbsolutePath());
	imp2 = new ImagePlus(secFile.getAbsolutePath());
	imgpr1 = imp1.getProcessor().convertToFloat();
	imgpr2 = imp2.getProcessor().convertToFloat();
}
```

Le constructeur de TwoImages, crée les ImagePlus, et les processeurs, qui sont convertis en processeurs de flottants réels (32-bit) pour conserver une cohérence sur les valeurs des pixels.
Ensuite on a la méthode d'addition des fits : (équivalente à celle de soustraction)

```
public ImagePlus addFits() {
	for(int col=0; col < imp1.getWidth(); col++) {
		for(int line=0; line < imp1.getHeight(); line++) {
			float value = imgpr1.getPixelValue(col, line)
			+ imgpr2.getPixelValue(col, line);
			imgpr1.putPixelValue(col, line, value);
		}
	}
	type = "Added : " + secFile.getName() +" to ";
	return new ImagePlus(type + CoreInterface.getFile().getName(),imgpr1);
}
```

On est obligé de créer une nouvelle ImagePlus à partir de l'ImageProcessor converti en 32-bits, car le fait de mettre des valeurs dans les pixels avec putPixelValue(..), ne s'applique qu'au processeur et pas à l'ImagePlus dont il provient.

Classe Histogram :

Cette classe est pour l'instant en développement, elle devrait afficher les histogrammes correspondants à une image FITS, elle nécessite l'utilisation d'autres librairies et un peu de travail de mise en forme.
Nous espérons les mettre en place avant la fin du projet.

2. Le package « gui »

Ce package contient toutes les classes gérant l'interface avec l'utilisateur, mais aussi une classe d'interface avec le cœur, ainsi que la classe permettant de visualiser les image.

- CoreInterface
- UserInterface
- HeaderTab
- ImageTab
- ImageViewer

Classe CoreInterface :

CoreInterface est une classe qui permet d'avoir accès aux informations générées par le cœur du programme, elle charge également le fichier dans l'application :

```
public static void loadFile(File file) throws Exception {
	FitsHandler testFitsHandler;
	try {
		testFitsHandler = new FitsHandler(file);
	} catch (Exception e) {
		//e.printStackTrace();
		throw e;
	}
	srcFile = file;
	fitsHandler = testFitsHandler;
}
```

Elle gère donc tous les appels aux classes du « core » par l'interface :

- Accéder à la Map contenant les informations de l'en-tête

```
public static Map<String,String> getHeaderMap() {
	return fitsHandler.getHeader().getMap();
}
```

Afficher les différentes images

- Image brute :

```
public static void displayRaw() {
	RawImage raw = new RawImage();
	new ImageViewer(raw.getImage(), raw.getType());
}
```

- Image en négatif :

```
public static void displayInverted(){
	InvertedImage inv = new InvertedImage();
	new ImageViewer(inv.getImage(), inv.getType());
}
```

- Image avec ajout d'une constante :

```
public void displayAdd(double value) {
	Mathematical add = new Mathematical();
	new ImageViewer(add.getAddImage(value), add.getType());
}
```

Et ainsi de suite pour toutes les types d'images disponibles.

Classe ImageViewer :

```
public ImageViewer(ImagePlus imp, String imageType) {

	super(imageType+" : "+CoreInterface.getFileName());
	currentImp = imp;
	type=imageType;
	//Listener to be able to quit displayed window.
	addWindowListener(new WindowAdapter() {
		public void windowClosing(WindowEvent e){
			windowActivated(e);
			setVisible(false);
			dispose();
		}
	});

	Image img = imp.getImage();
	if (img==null) System.exit(0);

	ImageTab imgtab = new ImageTab();
	JPanel imgPanel = new JPanel();
	ImageCanvas ic = new ImageCanvas(imp);

	imgPanel.add(ic);
	imgPanel.add(imgtab.saveImg());
	add(imgPanel);

	// Added borders height & width, to display the full image.
	setSize(imp.getWidth()+50, imp.getHeight()+100);
	setVisible(true);
}
```

La classe ImageViewer est en fait juste une « coquille vide » qui reçoit une instance d'ImagePlus (que l'on recupère à travers la CoreInterface) et qui affiche simplement l'image voulue. On choisi de travailler sur ImagePlus et une variable Type en entrée, pour rendre l'ImageViewer indépendant du type de transformation appliquée à l'image.

On peut aussi voir sur la fenêtre de l'ImageViewer, l'affichage des coordonnées du point où se trouver la souris, et l'intensité du pixel correspondant et une méthode (et son bouton) permettant l'enregistrement en jpeg des images FITS.

```
private void saveImage(File file){
        IJ.saveAs(currentImp, "jpeg", file.getAbsolutePath());
}

public void mouseMoved(MouseEvent me)  {
   xpos = me.getX();
   ypos = me.getY();
   Positionarea.setText("x: "+ xpos +"  y: "+ ypos);
}
```

Classe HistrogramViewer :

Cette classe est en construction et permettra comme l'ImageViewer, de visualiser n'importe quel type d'histogramme dans une fenêtre séparée.

Gestion des exceptions

Au départ, nous ne savions pas comment gérer les exceptions, nous avons donc dû trouver un moyen d'apprendre à les gérer, et notre but principal était de permettre l'exécution en continue du programme, tout en avertissant l'utilisateur de l'erreur produite. Pour cela, nous faisons « remonter » l'exception depuis le départ.

Dans le constructeur du FitsHandler, nous « catchons » l'exception en cas de fichier corrompu ou non-standard, ou d'un autre format que FITS.

```
public FitsHandler(File file) throws FitsException {
        try {
                Fits srcFile= new Fits(file);
                this.header =new HeaderHandler(srcFile.read()[0].getHeader());

        } catch (FitsException e) {
                //e.printStackTrace();
                throw e;
        }
}
```

Ce constructeur est appelé dans la classe CoreInterface, on l'on « catch » également une exception :

```
public static void loadFile(File file) throws Exception {
		//System.out.println("loadFile()");
		FitsHandler testFitsHandler;
		try {
			testFitsHandler = new FitsHandler(file);
		} catch (Exception e) {
			//e.printStackTrace();
			throw e;
		}
		srcFile = file;
		fitsHandler = testFitsHandler;
	}
```

Dans l'interface, en particulier la classe HeaderTab qui contient le log d'erreur, on a la méthode ci-dessous qui est exécutée à l'ouverture d'un fichier :

```
	public static void init(File file) {
		try {
			CoreInterface.loadFile(file);
			log.append("Loaded :  " + CoreInterface.getFileName() + "." +
		newline);
			fillheader(head);
		} catch (Exception e) {
			displayLoadError(file);
			//e.printStackTrace();
			//throw e;
		}
	}
```

On y attrape bien l'exception, et si elle arrive, notre méthode displayLoadError() écrit un message d'erreur dans le champ prévu à cet effet.

3. Fonctionnement du GUI :

Nous sommes donc reparti sur le même genre d'interface que le précédent RISA. C'etais un souhait important des enseignants. Mais comme nous l'avons dit précédemment, l'objectif principale fut de coder une interface clair et surtout facilement modifiable. Nous faison donc tout « à la main » et n'utilisons pas de logiciel tiers comme NETBEANS.
L'interface est donc architecturé de la maniere suivante :

Classe UserInterface :

Cette classe est la piece maitresse de l'interface. C'est dans cette classe que l'on ajoute tous les élements la constituant. Et c'est dans cette classe que ce trouve le main de RISA.

```
public UserInterface() {

        super(new BorderLayout());

        Border paneEdge = BorderFactory.createEmptyBorder(0,10,10,10);

        JTabbedPane tabbedPane = new JTabbedPane();

        headtab = new HeaderTab();
        JPanel Panel1 = new JPanel();
        Panel1.setBorder(paneEdge);
        Panel1.setLayout(new BoxLayout(Panel1, BoxLayout.Y_AXIS));
        Panel1.add(Box.createRigidArea(new Dimension(2, 10)));
        Panel1.add(headtab.FitsDisplay());
        Panel1.add(headtab.LogDisplay());
        Panel1.add(headtab.HeaderDisplay());
        tabbedPane.addTab("Header", Panel1);

        imagetab = new ImageTab();
        JPanel Panel2 = new JPanel();
        Panel2.add(imagetab.Picture());
        Panel2.add(imagetab.Math());
        tabbedPane.addTab("Pictures", Panel2);

        JPanel Panel3 = new JPanel();
        Panel3.add(others());
        tabbedPane.addTab("Coming Soon", Panel3);

        //Add tabbedPane to this panel.
        add(tabbedPane, BorderLayout.CENTER);
}
```

Les Panel1 Panel2 Panel3 correspondent aux onglets présents dans le l'interface. Pour clarifier le codage du Gui, chaque onglet a sa propre classe, dans laquelle toutes les méthodes auxquelles il fait appelle sont présentes.

Classe HeaderTab :

Cette classe contient donc toutes les méthodes permettant un affichage du premier onglet (un exemple de méthode ci-dessous).
Ces méthodes sont accompagnés de leur propre actionListener. Nous avons fait ce choix afin de ne pas se retrouver avec un classe actionListener très lourde dans laquelle nous ne savons pas a qu'elle partie du code le `e.getSource()` fait référence.

```
protected JPanel FitsDisplay() {
            //Create Label
            JLabel label1 = new JLabel("Open FITS files:");

            //Create a file chooser
            fc = new JFileChooser();

            //Create the open Button.
            openButton = new JButton("Open");
            openButton.addActionListener(this);

            //Create eraser Button
            erase = new JButton ("Erase log");
            erase.addActionListener(this);

            //Create Button Panel
            JPanel buttonPanel = new JPanel();
            buttonPanel.add(label1);
            buttonPanel.add(openButton);
            buttonPanel.add(erase);
            return buttonPanel;
            }
```

Dans cette méthode nous procédons de la manière suivant. On crée les éléments swing que l'on a besoin (Jlabel Jbutton JfileChooser…), ensuite on leur ajoute une action si il y en a besoin et enfin on crée le Jpanel auquel on ajoute toutes les éléments précédents.
Toutes les méthodes sont codées de cette manière de facon à faciliter la compréhension et le débuggage.

Classe ImageTab :

La classe est programmée de la même manière que la classe précédente

```
public class ImageTab extends JPanel implements ActionListener {
```

Elle comporte ces propres méthodes, son propre actionListener.

B. L'évolution du code :

Code source

Au niveau du code source, nous nous sommes appliqué à respecter les conventions Java recommandées par Sun. Tout au long du développement, nous avons veillé à écrire proprement, à bien nommer les variables, les méthodes et les classes, et à bien commenter l'entièreté du code source.

De cette façon, l'accès au code source devrait être relativement simple, et la compréhension assez rapide, ainsi, les prochains développeurs pourront maintenir le code et implémenter de nouvelles fonctionnalités plus facilement.

Dans le futur, on pourrait penser à une ré-organisation de la partie « core », avec une décomposition de la CoreInterface actuelle, en classes pour regroupant par types de transformations.

Wiki

Au tout début du développement nous avons mis en place une wiki hébergé sur un serveur personnel, dans le but de laisser une trace de notre travail, et des informations pour les futurs développeurs, renseignant sur les direction envisagées, l'orientation générale de l'application et les fonctionnalités à prévoir ou à améliorer.

Ce wiki pourra éventuellement être hébergé sur un serveur de la faculté, pour assurer sa longévité. Quoi qu'il arrive, il restera accessible le plus longtemps possible à son emplacement actuel.

L'interface

Pour faire évoluer l'interface, rien de plus simple. Pour ajouter un onglet, l'utilisateur futur doit créer une classe sur le même modèle que HeaderTab ou ImageTab.

```
public class ImageTab extends JPanel implements ActionListener {
```

Dans cette classe il devra créer des méthodes qui retourne un Jpanel :

```
protected JPanel others() {
        JPanel pane = new JPanel();

        return pane;
}
```

Il pourra aussi ajouter son propre actionListener :

```
public void actionPerformed(ActionEvent e) throws NullPointerException{
```

Pour finir et ajouter cet onglet au logiciel, il devra ajouter dans la classe UserInterface :

```
JPanel Panel3 = new JPanel();
Panel3.add(others());
tabbedPane.addTab("Coming Soon", Panel3);
```

L'interface étant réfléchi pour être évolutive, l'utilisateur pourra ajouter le nombre d'onglets et de fonctions qu'il desire sans restriction, mais il devra respecter les règles de programmation pour chaque méthode afin de conserver cette facilité.

Conclusion :

Le projet RISA a été, pour nous, intéressant mais difficile et nous a forcé à nous remettre en question sur notre approche au travail en collaboration et à la conception de logiciel. En effet, le travaille en équipe nous a obligé à nous adapter aux autres membres du groupe et à développer des solutions logistiques type SNV afin de travailler dans les meilleurs conditions possibles. Il nous a néanmoins permis d'apprendre énormément de choses sur la programmation objet, les librairies, les exceptions, la création d'interface et la corrélation entre code et ce qui nous apparait a l'écran tout en améliorant notre connaissance globale du Java. Pour finalement mettre en pratique nos apprentissages récents, et les concrétiser à travers ce projet et le logiciel RISA.

Nous avons donc conçu ce logiciel de façon à ce qu'il soit le plus simple à modifier dans le futur, avec des classes nombreuses chacune ayant des tâches bien précises. Ainsi, nous obtenons un code source plutôt optimisé et assez propre, qui permet une lecture simple et une compréhension rapide du programme.

Annexes :

A. Les Fichiers FITS :

1. Généralités

FITS ou Flexible Image Transport System est le format de fichiers le plus communément utilisé en astronomie. Le format de fichiers FITS est souvent utilisé pour sauvegarder aussi d'autres données, comme le spectre, des listes de photons, des cubes de données, et bien d'autres choses encore. Une fichier FITS peut contenir plusieurs extensions, et chacune de celles-ci peut contenir des données. Par exemple, il est possible de sauvegarder dans le même fichier FITS des images à la fois dans le domaine des rayons X et dans celui de l'infrarouge.

À la différence de beaucoup de formats d'image, le format FITS est conçu spécifiquement pour des données scientifiques et par conséquent inclut beaucoup de dispositifs pour décrire l'information photométrique et spatiale de calibrage, ainsi que les metadata d'origine d'image.

FITS ne possède pas encore de logiciels standards génériques. Les utilisateurs doivent développer leurs propres logiciels pour lire et visualiser leurs données.
C'est un format de données standard approuvé par NOST (NASA/Science Office of Standards and Technology). Il existe un service support (le FITS Support Office Software) responsable de la documentation du standard, qui participe à l'évolution de ce dernier.

Son domaine d'application est essentiellement l'astronomie et l'astrophysique. Dans ce cadre
là, un groupe de travail, le IAUFWG (International Astronomical Union FITS Working Group) s'est formé auquel sont soumises toutes propositions d'évolution du format. Celui-ci décide de refuser ou d'accepter l'évolution en prenant comme règle première, dans le cas de l'acceptation, d'assurer toujours une compatibilité avec les versions antécédentes.

2. Composition du format Fits

Un dispositif important du format FITS est que les metadatas d'image sont stockés dans un en-tête lisible par un humain, au format ASCII, de sorte qu'un utilisateur intéressé puisse examiner les en-têtes d'un fichier de provenance inconnue. Chaque fichier FITS se compose d'un ou plusieurs en-têtes contenant des "**card" ASCII (80 caractères de longueur constante)** qui portent des paires de "**keyword / value**" (mots clés / valeurs), intercalées entre les blocs de données. Les paires de mots clés / valeurs fournissent des informations telles que la taille, l'origine, les coordonnées, le format de données binaire, les commentaires en format libre, l'historique des données, et toute autre chose souhaitée par le rédacteur ; tandis que beaucoup de mots-clés sont réservés pour l'usage interne, la norme permet l'utilisation arbitraire du reste.

B. Lettre de Mission :

RISA

Introduction- Historique

RISA *Il existe des logiciels professionnels (MIDAS, IRAF, …) pour la réduction des données astronomiques recueillies aux foyers des télescopes ou de leurs instrumentations auxiliaires. Ils sont d'un apprentissage très lourd et masquent souvent la physique de la réduction de données. C'est pour pallier ces deux inconvénients qu'avait été réalisé à l'UM2, avec Christian Sam, stagiaire d'IUT informatique, en 1995 le logiciel IRIS (Initiation à la Réduction d'Images et de Spectres) qui tournait sous DOS (puis sous Windows sous sa forme Iris-fv) dans les salles de TP du CR2I puis de l'UFR. Il utilisait le format professionnel FITS, les algorithmes professionnels précis mais, pédagogique, son interface posait des questions physiques à l'utilisateur. Depuis 2007 RISA reprend et amplifie cette facilité.*

Problématique

RISA (**R**éduction d'**I**mages et de **S**pectres **A**strophysiques) est un logiciel créé par Vannak Kol, Alexandre-Florent Gonthier et Julien Horte lors de leur stage de M1 Phy-Info en 2007. Le cahier des charges était de reproduire les fonctionnalités de base de l'ancien « IRIS-fv » mais avec les nouvelles possibilités que permettaient le fenêtrage et l'indépendance du système d'exploitation et de rajouter de nouvelles fonctionalités notamment en photométrie. Le travail a été complété en 2008 par Hervé Dauberton, Jenny Bravo et Julien Roupie avec encore l'aide d'Alexandre-Florent Gonthier. En 2009 c'est Mounir Bensadia, Serge-Henri Cachia et Boualam Hasnoun qui ont poursuivi ce travail pour corriger quelques bugs et développer la partie « photométrie » (mesures sur images). En 2010 Donia Aissa, Nicolas Glaizette et Karim Bammou ont repris un certain nombre de dysfonctionnements dans le traitement des données et de leur affichage, d'amélioration de l'interactivité et développé l'aide en ligne. RISA, en versions d'essai sert depuis 2007 pour la réduction des observations en M1 (actuelle UE « Astrophysique Observationnelle », 15 étudiants en 2009-2010). Il reste encore quelques bugs dans le traitement de certains fichiers et des « incommodités ». Par ailleurs la partie photométrie n'a pas été testée en vraie grandeur avec les étudiants (coupole Est en cours de rénovation).

Finalité

L'objet du stage 2011 est d'abord de faire atteindre à RISA un niveau professionnel qui nous permettra de le diffuser de manière publique. Le souhait est de faire de RISA un logiciel pédagogique au service de l'enseignement de l'astrophysique au sein de différentes universités. Le travail demandé porte en grande partie sur une refonte de l'architecture actuelle du logiciel. La nouvelle architecture devra offrir la modularité, la maintenabilité et la flexibilité attendues pour tout outil d'analyse pouvant être partagé par l'ensemble d'une communauté. Ainsi, RISA ne fonctionne bien aujourd'hui qu'au sein de l'environnement Windows alors qu'il a été développé avec le langage Java pour justement faciliter sa portabilité (Linux, MacOS). Ensuite, l'organisation actuelle du code ne permet pas de le faire évoluer de manière souple et ainsi rend difficile la correction des problèmes persistants ou encore la prise en charge de nouvelles fonctionnalités. Il faut donc retravailler les notions du développement orienté objet pour répondre à ces limites. Il est attendu d'exploiter différentes librairies Java dédiées comme la librairie de la Nasa nom.tam.fits (http://heasarc.gsfc.nasa.gov/docs/heasarc/fits/java/v0.9/) ou encore la librairie d'ImageJ (http://imagej.nih.gov/ij/) pour rendre efficace l'écriture du code. Le code devra également s'affranchir d'erreurs d'exécution. Dans ce sens, les mécanismes de gestion d'exception seront révisés et testés sur des exemples concrets. Le modèle conceptuel précisant les classes Java à développer sera fourni par les encadrants.

Une partie d'exploitation physique est incluse. Elle consistera par exemple en la réalisation d'une étude photométrique d'étoiles d'un amas avec RISA de données acquises au télescope.

Encadrants astronomes : Jean-Pierre Cordoni <Jean-Pierre.Cordoni@graal.univ-montp2.fr> et Henri Reboul <Henri.Reboul@univ-montp2.fr>
Encadrant Informaticien : Isabelle Mougenot <Isabelle.Mougenot@lirmm.fr>

Bibliographies :

INTERFACE GRAPHIQUE EN JAVA, API SWING :
Institut National Paris-Grignon

JAVA
Claude Delannoy

JAVADOC POUR JAVA
Sun

THE JAVA TUTORIALS
http://download.oracle.com/javase/tutorial/index.html
Oracle

THE SUN CERTIFIED JAVA DEVELOPER EXAM WITH J2SE5, SECOND EDITION
Paul Sanghera, Ph.D.