

Documentation for Development

Open Source Applications in Nuclear
Medicine

Version 06/21/2019

Esteban BAICHOO - Titouan QUÉMA

Introduction

Ce document est issu du stage réalisé par des étudiants de l'IUT Paul Sabatier, durant 3 mois, Titouan QUÉMA et Esteban BAICHO.

Ce stage s'inscrit dans une continuité, car c'est la troisième année consécutive que le Dr. Kanoun, via l'Oncopole de Toulouse, développe une application visant à traiter des images de scintigraphie médicale.

Jusqu'à maintenant, tout le travail a été réalisé durant des périodes de stage de 3 mois depuis 3 années.

L'apport majeur qui a été fait dans cette dernière période a été une harmonisation et restructuration générale du code, afin de le rendre plus clair et plus adaptable aux différents besoins, en proposant une implémentation simplifiée et solide des différents examens.

Dans ce qui suit se trouve une simple explication des mécanismes et des choix effectués. Une lecture approfondie du code reste nécessaire en complément de ce document pour comprendre le fonctionnement du projet.

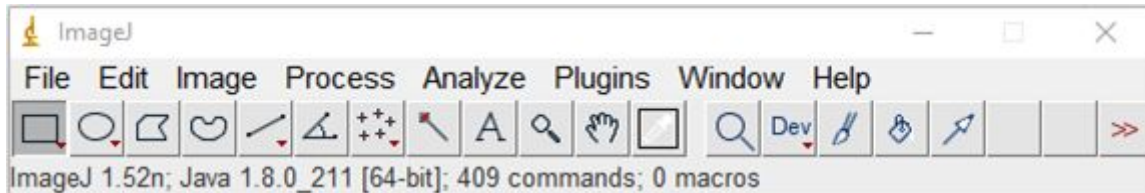
Cette documentation sert à fournir un aperçu bref et concret du fonctionnement de l'application, et à permettre aux futurs développeurs de comprendre le fonctionnement actuel du programme, avant de se familiariser avec le code.

Nous recommandons de lire ce document avant d'entrer dans le code, car ce dernier sera compris par la suite.

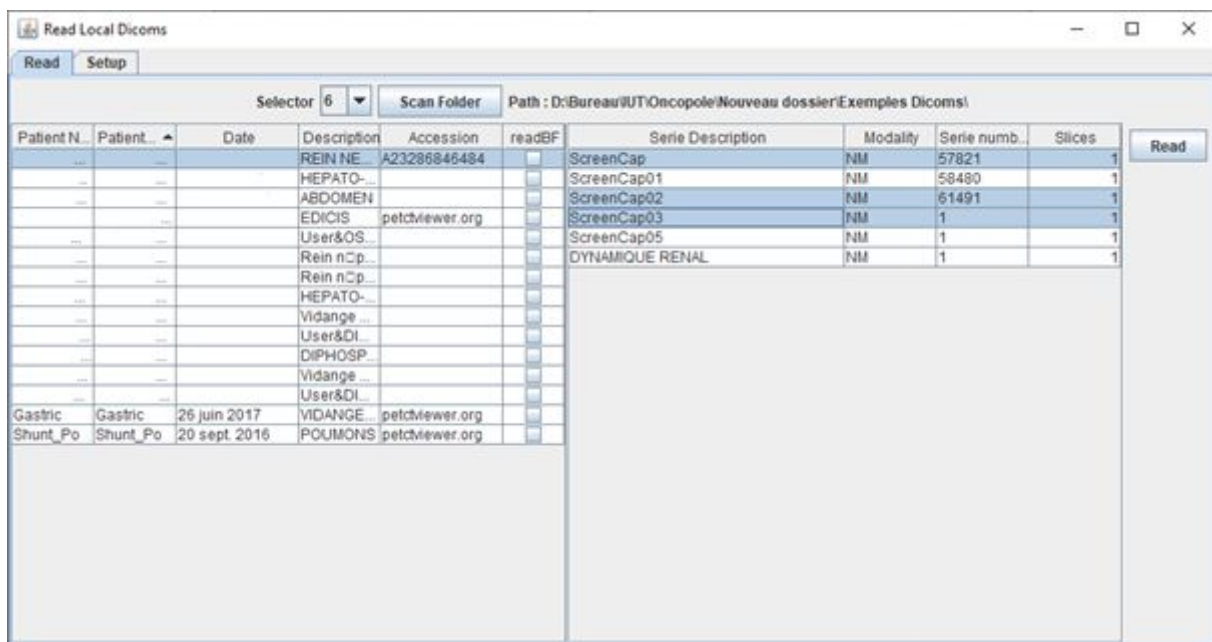
Fonctionnement général d'une application utilisant un workflow

Dans un examen classique, on définit un workflow comme une suite d'instructions que l'utilisateur accomplit une par une, en appuyant sur le bouton « Next ».

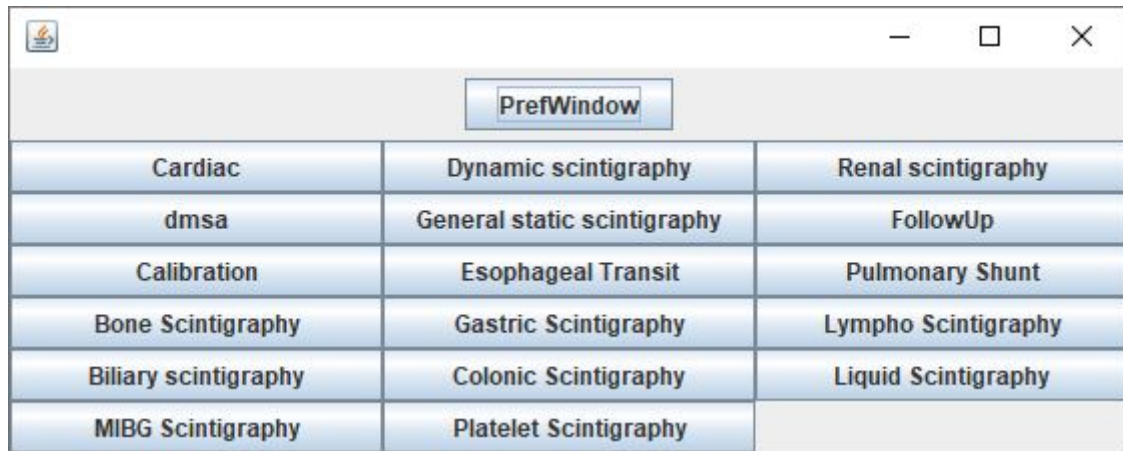
Pour l'utilisateur, le programme commence avec ces 3 fenêtres ouvertes :



Panel utilitaire ImageJ



Fenêtre de sélection d'examen



Fenêtre de rechercher de DICOM

L'utilisateur choisi donc de cliquer sur « Scan Folder », choisi une image ou une série d'image et clic sur « Read ».

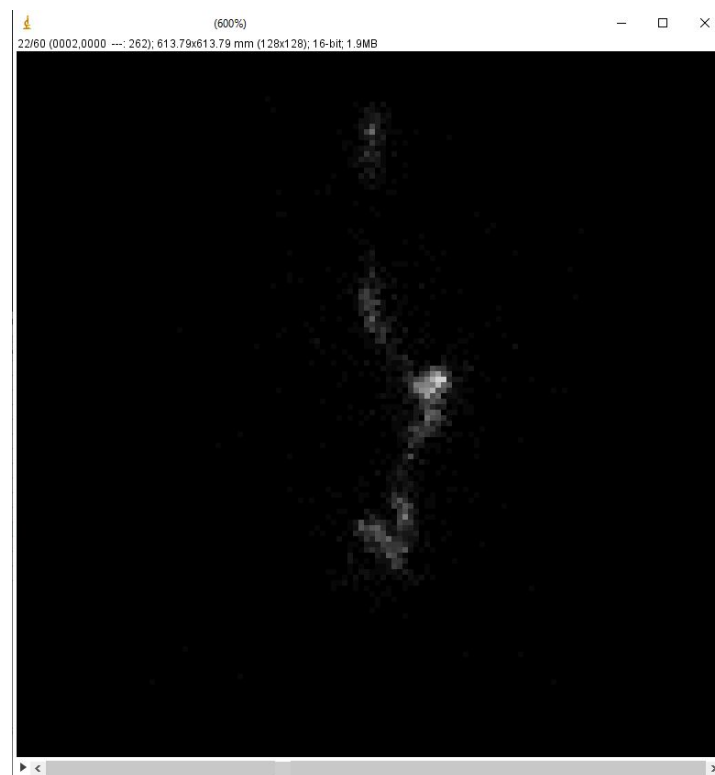
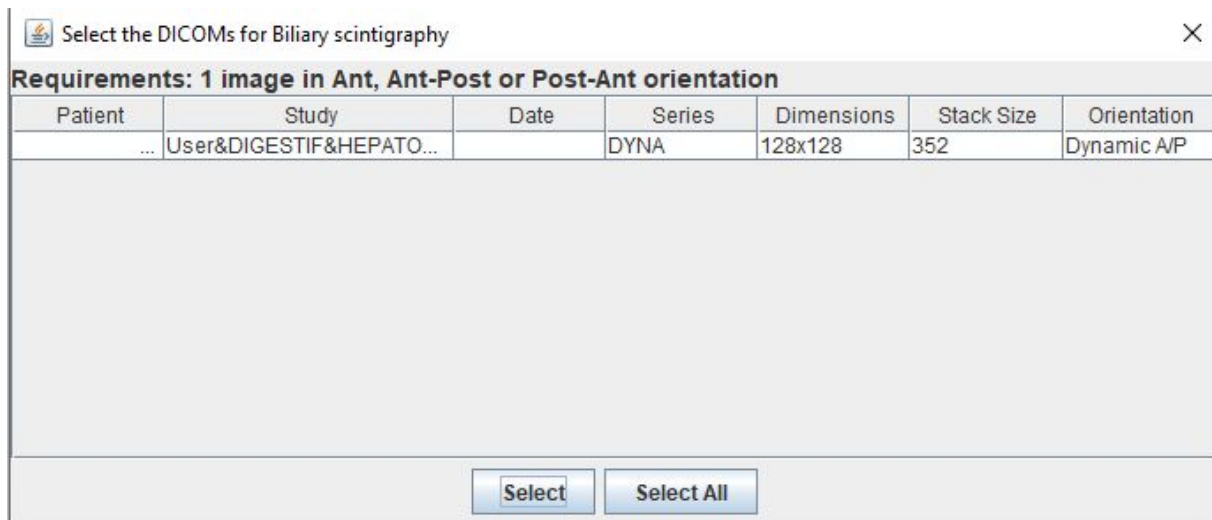


Image DICOM ouverte par ImageJ

Ensuite après avoir ouvert les images, l'utilisateur clique sur l'examen voulu et choisi les images qui sont nécessaires.

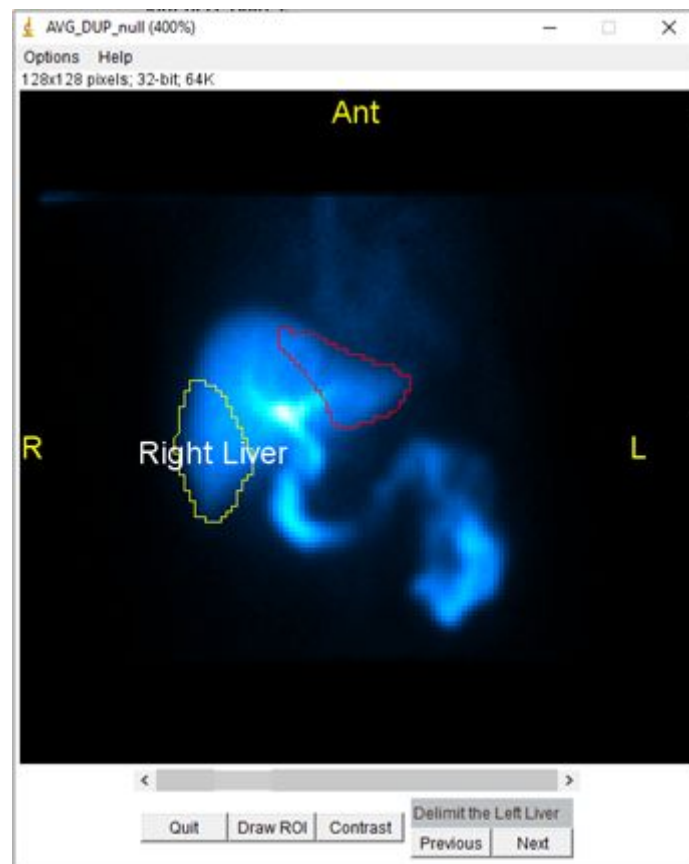


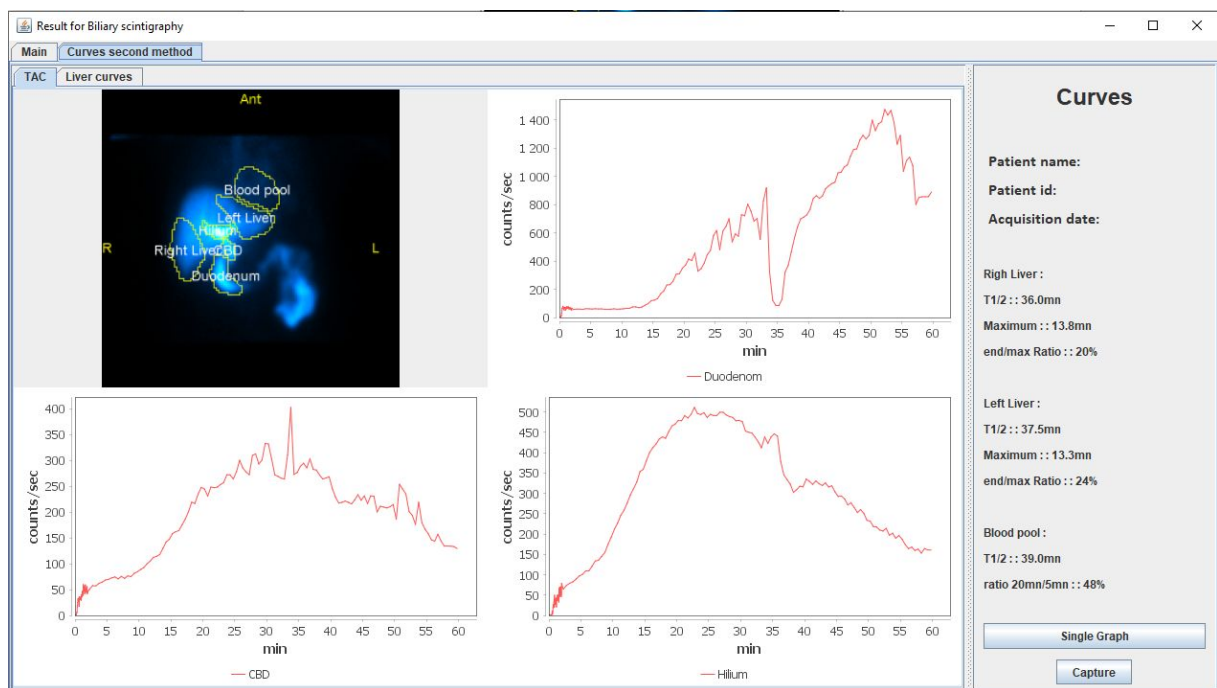
Fenêtre de sélection des DICOM

Cela ouvre une FenApplication, qui permet à l'utilisateur de visionner les images et de dessiner les différentes Region Of Interest (ROI) qui sont nécessaires à l'examen.

Les ROIs déjà dessinées et validées sont en jaunes. La ROI en édition est en rouge. Le texte affiché au-dessus des boutons indique l'instruction en cours que doit réaliser l'utilisateur.

Lorsque toutes les instructions ont été complétées, la FenResults se lance automatiquement, affichant les résultats de l'examen.





Fenêtre de résultats

La fenêtre de résultats est composée de deux éléments : un SidePanel, situé à droite qui reprend les informations du patient (nom, id et date d'acquisition). Des résultats peuvent aussi être affichés, puis un bouton de capture est généralement toujours présent.

Le deuxième élément est le panel de résultats au milieu de l'écran. Ce panel est majoritairement utilisé pour afficher les graphiques et les captures d'écrans prises durant la sélection des ROIs.

Le panel des résultats est composé lui-même de TabResult, qui servent à séparer les résultats présentés à l'utilisateur sous différents onglets. Il est possible d'ajouter autant de TabResult que nécessaire sur la FenResults.

ImageSelection

L'ImageSelection est le successeur de l'ImageOrientation.

ImageOrientation permettait de stocker une ImagePlus - qui est l'objet fourni par ImageJ - et son orientation, déterminée à partir des tags de l'image ou bien donnée par l'utilisateur.

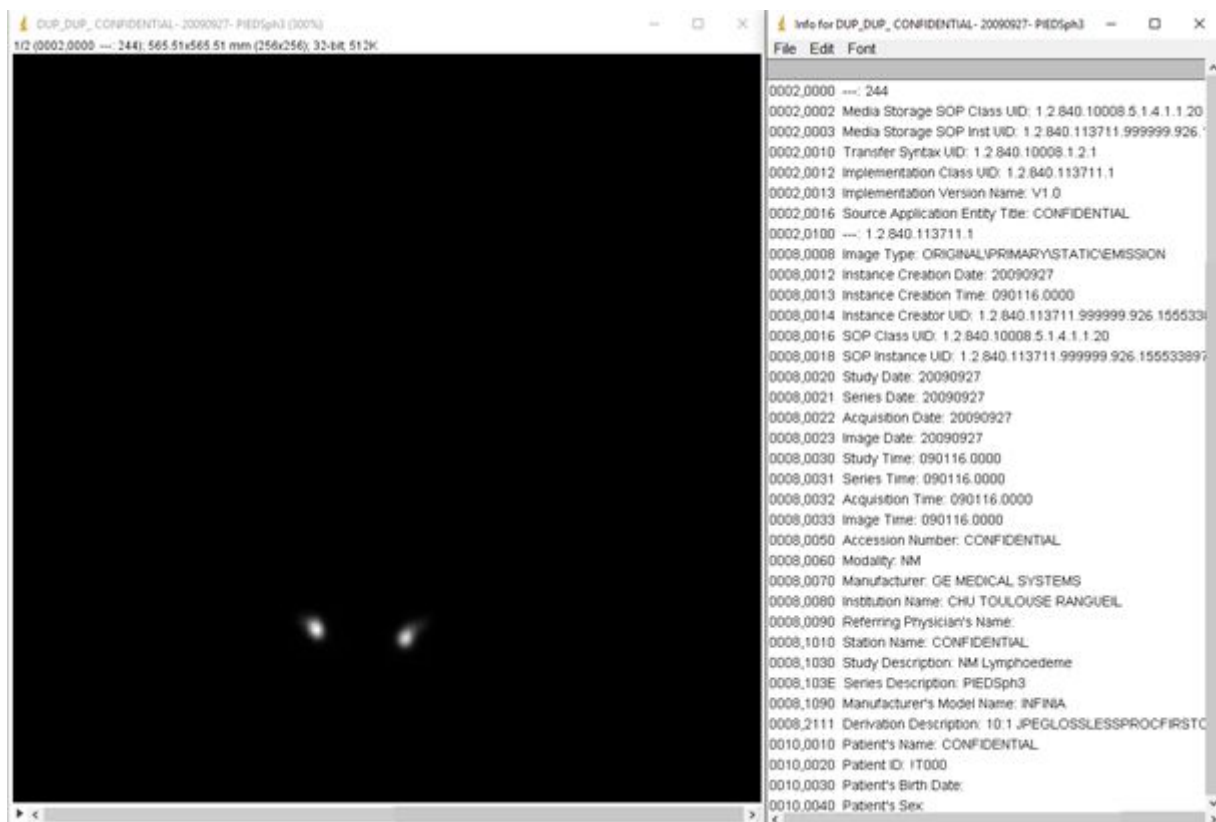
L'ImageSelection englobe encore plus de données, permettant de stocker toute nécessité particulière à un examen, grâce aux colonnes présente dans la FenSelectionDicom.

Ces données sont stockées sous forme de HashMap<String, String>, et sont récupérées depuis les colonnes de la FenSelectionDicom.

Pour voir comment éditer les colonnes, voir [Columns](#).

ImagePlus

La librairie *ImageJ* permet d'ouvrir les images DICOM (.dcm) sélectionnées et de les interpréter en Java, car une image DICOM est composée d'un *header* (qui contient des informations relatives à l'image, à ses moyens d'acquisition, au traceur utilisé, au patient, et d'autres données), ainsi que la matrice image représentant les pixels.



ImagePlus – Image (à gauche) et header (à droite)

ImageState

Un objet ImageState permet de définir l'état dans lequel une image doit être affichée. Cet objet permet donc de définir des propriétés sur l'image.

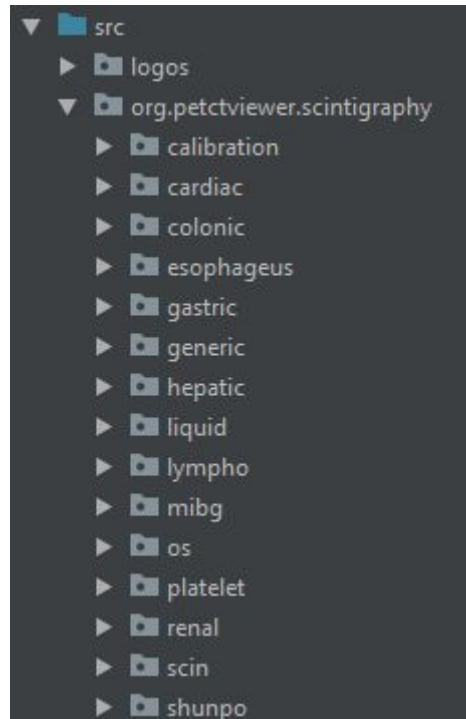
Les propriétés pouvant être définies sont :

- index de l'image à afficher : cet index correspond à la position de l'image dans celles qui se trouvent dans le modèle. Quelques constantes sont définies pour certains cas spéciaux.
- slice de l'image : lorsqu'une image possède plusieurs slices, il est nécessaire de définir celle à afficher.
- latéralisation : indique où se situe la droite et la gauche de l'image par rapport à l'écran. Par exemple, une image avec une latéralisation Right-Left indique que la droite du patient se trouve à gauche de l'écran et que la gauche du patient se trouve à la droite de l'écran.
- orientation de l'image : l'orientation de l'image peut être en Antérieur ou en Postérieur

Toutes ces informations sont utilisées (notamment par la méthode `prepareImage()` du contrôleur) pour afficher l'ImageSelection à l'écran.

Structure of the Project

The project is composed of packages. Each application is wrapped inside a package and relies on the MVC pattern.



The only exception is the `scin` package which gathers all of the abstract classes along with all of those used by several applications.

MVC Pattern

All of the applications in the project are designed with the MVC architecture. Abstract classes were made to group most of the features requested by the programs. Abstract classes are located in the `scin` package.

Here is the location of each class matching an object of the pattern:

- Model → `scin > model > ModelWorkflow`
- View → `scin > gui > FenApplicationWorkflow`
- Controller → `scin > controller > ControllerWorkflow`

Example of Creating an Application

Here are some quick steps to create a simple application.

First, you need to create a new package under `scintigraphy`. Name it “test”.

Scintigraphy Class

The first class to create is the `TestScintigraphy` which will be used as an entry point for our application.

It is necessary to override some abstract methods of this class. We will explain each of these methods:

- `super(String)` : When calling the parent constructor, the name of the program is passed in. This name will be used when displaying results, for instance.
- `start(List<ImageSelection>)` : This method launches the program. Generally, the view and the controller are created here.
- `getColumns()` : Gives a list of columns to display to the user while selecting the images for this program (in the `FenSelectionDicom`). Since our program is very simple, the default columns will be enough. To do that, you just have to call the `Column.getDefaultColumn()` method located in the `FenSelectionDicom` class.
- `prepareImages(List<ImageSelection>)` : This is the most important method. It ensures that all images provided by the user complies the necessities of the program. To do that, you should duplicate the images and then close all of the images given in input. You should also check that the number of images provided is correct, and that each image orientation is accepted. If needed, the images can be ordered (with the comparators provided in the `scin` package).
- `instructions()` : In this method, you provides a simple text to the user for him to know what kind of image is requested by the application.

View Class

Once this entry point is made, the view of the application needs to be created. In the `start` method of the `scintigraphy`, you can use the provided method `setFenApplication(FenApplication)` by passing it a new instance of the `FenApplication` class. Since we do not want to change the design of the view, we don't have to create a inherited class of the view, and we can simply create a new instance.

Controller Class

Then, we have to pass a controller to the view. The controller should inherit from the `ControllerWorkflow`. You also need to pass the list of the opened images to the controller, since it will be from it responsibility to create the model later on.

The controller has the responsibility to intercept any user input. But most of that job is handled by the `ControllerWorkflow`. In fact, almost everything is managed by this class.

The only thing you need to define, is the workflow itself. You need to implement the `generateInstructions()` method, in order to create the workflows of this program.

For instance, let say we want the user to draw the stomach and then take a capture of it.

Here is a sample code for that:

```
this.workflow = new Workflow[1];
this.workflow[0] = new Workflow(this,
this.getModel().getImageSelection()[0]);
ImageState stateAnt = new ImageState(Orientation.ANT, 1, ImageState.LAT_RL,
ImageState.ID_WORKFLOW);
this.workflow[0].addInstruction(new DrawRoiInstruction("Stomach", stateAnt);
this.workflow[0].addInstruction(new ScreenShotInstruction(this.captures,
this.vue, 0, 0, 0));
this.workflow[0].addInstruction(new EndInstruction());
```

And that's it! Everything else is handled by the `ControllerWorkflow`.

Workflow

Le workflow est basiquement une liste d'instructions, attachées à une image.

Il n'y a normalement aucune raison de modifier cette classe car elle n'est utilisée que par le `ControllerWorkflow` pour gérer le flux d'instructions.

Instruction

Une instruction est une **Interface**, composée de méthodes qui seront appelées par le `ControllerWorkflow`.

Certaines classes ont été créées implémentant cette interface pour des opérations courantes (telles que la création d'une nouvelle ROI, la prise d'une capture d'écran...). Toutes ces instructions déjà créées sont présentes dans les sous-package de : **scin > instructions**.

Pour savoir si l'arrivée se fait après un appui sur le bouton « Next » ou sur le bouton « Previous », il suffit de placer son code dans les bonnes méthodes. En effet, les méthodes :

```
void prepareAsNext();
void afterNext(ControllerWorkflow);
```

sont appelées lors d'un clic sur "Next", et les méthodes :

```
void prepareAsPrevious();
void afterPrevious(ControllerWorkflow);
```

sont appelées lors d'un clic sur "Previous".

Comme mentionné précédemment, c'est le `ControllerWorkflow` qui s'occupe d'appeler ces méthodes au bon moment.

Il faut bien noter que ces méthodes permettent de gérer l'arrivée sur une instruction, à différents moments. Par exemple, `prepareAsNext()` sera appelée avant la sauvegarde

potentielle de la ROI courante, et `afterNext(ControllerWorkflow)` sera appelée après la sauvegarde.

Il est à noter qu'il n'est pour le moment pas possible pour une instruction de savoir lorsqu'elle n'est plus l'instruction courante.

Cela peut être un point d'amélioration pour les autres stages.

La gestion des ROI se fait si `saveRoi()` renvoie vrai, et c'est le `ControllerWorkflow` qui gère ensuite la sauvegarde des ROI.

Le `Controller` garde en interne un `indexRoi` pour connaître le nombre de ROI sauvegardées, et savoir lesquelles afficher lors des clics sur « Previous » et « Next », ainsi que pour associer aux instructions qui sauvegardent des ROIs, l'index de la ROI correspondante dans le `RoiManager`.

ControllerWorkflow

Le ControllerWorkflow est ce qui gère l'interaction avec l'utilisateur, mais également entre la FenApplication et le Modèle.

Les fonctions `start()` et `generateInstructions()` doivent obligatoirement être appelées manuellement à la fin du constructeur d'un ControllerWorkflow.

Il gère aussi la fonction `prepareImage()` qui affiche la bonne image pour l'utilisateur, et place les Overlay en fonction de l'ImageState associée à l'Instruction courante.

Cette partie gère également l'update de la Scrollbar des workflow si l'instruction de la Scrollbar n'est pas la même que l'instruction courante au moment de l'appuie sur le bouton « Next » ou « Previous ».

Gère également l'action à faire lors du clic sur le CaptureButton.

Dans un souci de compréhension dans le cadre d'une première lecture avant le stage, je vais représenter simplement les implémentation de `clickNext()` et `clickPrevious()`.

Algorithme simplifié de `clickNext()`

On récupère l'instruction courante

Si elle n'est pas `cancelled`

 Si l'Instruction courante sauvegarde une ROI

 On sauvegarde la ROI

 On affecte l'index de la ROI à l'instruction courante

 Si la ROI doit être affichée

 On affiche la ROI

 On incrémente `l_indexRoi` (index des ROIs)

 Si l'instruction courante est génératrice d'instruction

 On génère une nouvelle instruction génératrice

 On ajoute cette nouvelle instruction génératrice à la liste des instructions

`super.clickNext()` ;

Si l'instruction courante est la dernière du workflow actuel

 On passe au workflow suivant

On passe à l'instruction suivante

On appelle `prepareAsNext()` ; de l'instruction suivante

On vérifie si il faut griser le bouton « Previous » ou pas

 Si c'est la dernière instruction du Controller

 On appelle la fonction `end()` du `ControllerWorkflow`

 Si l'instruction suivante attend une action de l'utilisateur

 On affiche les instruction de l'instruction suivante

 On prépare l'image associée

 On regarde si il y a une ROI à afficher pour cette instruction suivante¹

 On appelle `nextInstruction.afterNext(this)` ; de l'instruction suivante

Sinon si l'instruction suivante n'attend pas d'action de l'utilisateur

 On appelle `nextInstruction.afterNext(this)` ; de l'instruction suivante

 On simule un appui sur le bouton « Next »

Sinon si l'instruction courante est `cancelled`

 Si l'instruction courante n'attend pas d'action de l'utilisateur

 On simule l'appui sur le bouton « Previous »²

Algorithme simplifié de `clickPrevious()`

`super.clickPrevious()`

On récupère l'instruction courante

On récupère l'instruction précédente

On vérifie si on doit griser le bouton précédent

Si le Controller était fini et que l'instruction précédente est une instruction génératrice

On active l'instruction génératrice, pour lui permettre de générer de nouveau

Si l'instruction précédente est `null`

On passe au Workflow précédent

On récupère l'instruction courante du workflow précédent

On appelle de `prepareAsPrevious()` de cette instruction

Si l'instruction précédent attend une action de l'utilisateur

On affiche les instructions de l'instruction

On actualise l'Overlay

On prépare l'image

Si l'instruction précédente doit sauvegarder une ROI

On décrémente `l'indexRoi` (index des ROIs)

On affiche sur l'Overlay toutes les ROIs à afficher

Si l'instruction précédente doit sauvegarder une ROI

On permet d'éditer la ROI de l'instruction précédente

On appelle `afterPrevious(ControllerWorkflow)` de l'instruction précédente

Sinon si l'instruction précédente n'attend pas d'action de l'utilisateur

Si l'instruction précédente doit sauvegarder une ROI mais ne doit pas l'afficher

On décrémente `l'indexRoi` (index des ROIs)

On appelle `afterPrevious(ControllerWorkflow)` de l'instruction précédente

On simule un appuie sur le bouton « Previous »

Librairies

Les différentes librairies regroupent les méthodes statiques pouvant être appelées dans chaque examens.

Elles ont été factorisées ici car leur appel peut être commun à plusieurs exemples.

Library_Capture_CSV

- Récupérer les différents tags de l'images (header)
- Récupérer les différentes informations du patient (Patient ID, Patient Name, Date, Accession Number)
- Faire des montages d'ImagePlus
- Créer un ImageStack à partir d'un tableau d'ImagePlus
- Prendre la capture d'une image

Library_Dicom

- Récupérer les Date Acquisition
- Connaître et traiter les Orientations d'une ImagePlus (depuis les tags)
- Trier les ImagePlus (Dynamique et Statique) par leur Orientation
- Trouver l'isotope associé à l'examen
- Normaliser les images en coups/secondes

Library_Gui

- Gestion des Overlay
- Gestion des ROIs

Library_JFreeChart

Library_Quantif

- Arrondir, calculer une moyenne géométrique
- Récupérer les coups, moyennes de coups, nombre de pixels
- Calculer un nombre de coup : corrigé du background, corrigé de la décroissance radioactive.
- Fourni les méthodes de convolution et de déconvolution

Cette librairie gère aussi les Isotopes.

FenSelectionDicom & Scintigraphie

FenSelectionDicom

FenSelectionDicom est la fenêtre qui permet de sélectionner les DICOM à ouvrir.

Scintigraphie

ImagePreparator

ImagePreparator est une interface qui permet de détacher la FenSelectionDicom de la Scintigraphie.

Précédemment, la FenSelectionDicom était lancée et lançait ensuite, après la sélection d'image, les différentes fonctions de Scintigraphie.

Ce fonctionnement a été revu car il ne permettait pas de simplement sélectionner des images, sans lancer d'examen, d'où l'apparition de ImagePreparator, pour rajouter un niveau d'abstraction.

Columns

Les colonnes de la FenSelectionDicom sont gérées au niveau de l'ImagePreparator. En pratique, étant donné que chaque Scintigraphie implémente ImagePreparator, elles sont dans le programmes gérées dans les Scintigraphie.

```
@Override
public Column[] getColumns() {

    // Orientation column
    String[] orientationValues = { Orientation.ANT_POST.toString(), Orientation.POST_ANT.toString() };
    Column orientation = new Column(Column.ORIENTATION.getName(), orientationValues);

    // Organ column
    String[] typesValues = { FULL_BODY_IMAGE, ONLY_THORAX_IMAGE };
    this.imageTypeColumn = new Column(COLUMN_TYPE_TITLE, typesValues);

    // Choose columns to display
    return new Column[] { Column.PATIENT, Column.STUDY, Column.DATE, Column.SERIES, Column.DIMENSIONS,
        Column.STACK_SIZE, orientation, this.imageTypeColumn };
}
```

Exemple de colonnes

Il faut redéfinir la méthode, et retourner un tableau de Column. Cet exemple nous montre que les colonnes classiques sont : { **PATIENT**, **STUDY**, **DATE**, **SERIES**, **DIMENSIONS**, **STACK_SIZE**, **ORIENTATION** }

Cependant, pour le besoin de l'examen, les orientations ont été redéfinies pour ne laisser de valide que ***ANT_POST*** et ***POST_ANT***.

Une colonne supplémentaire a été rajoutée, pour spécifier à quel type chaque image appartient (***this***.imageTypeColumn).

ReversedChronologicalAcquisitionComparator

C'est juste ce qui permet de trier les ImageSelection par date.

Gui

FenApplication

La fenêtre application est une fenêtre générique sur laquelle tous les examens se basent pour interagir avec l'utilisateur afin de réaliser les différentes étapes en vue de générer les résultats.

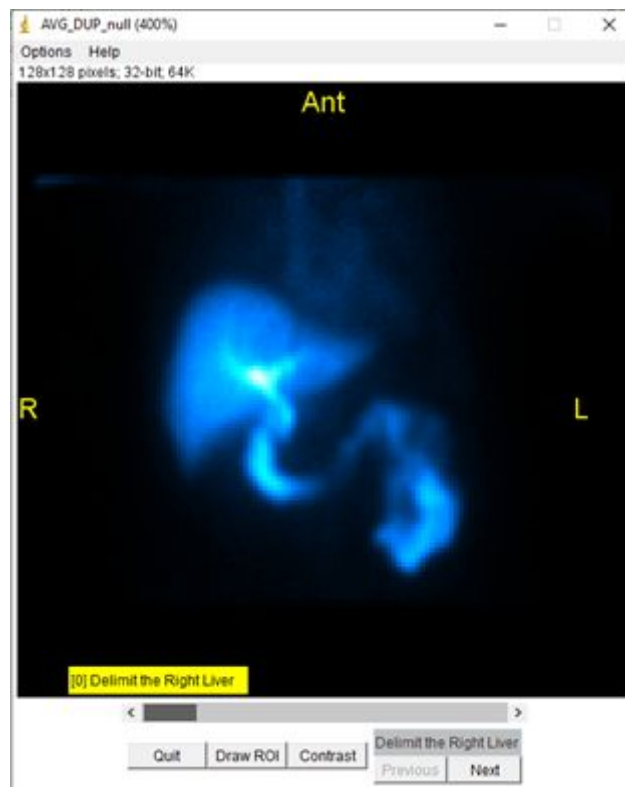
Elle a été initialement conçue pour exécuter des examens nécessitant des workflow, mais a été refactorisé pour être basiquement une fenêtre avec les boutons « Next » et « Previous », et un Slider permettant de visionner les différentes images à afficher.

Un fait important à ne pas oublier avec la FenApplication est qu'elle est un élément AWT, et non Swing.

Cette obligation vient du fait que pour accéder à une ImagePlus en lien avec ImageJ (et donc avec les fonctions de saisie de Roi, d'affichage, etc ...), il faut que la FenApplication hérite de StackWindow, qui est un élément AWT, car l'ImagePlus s'ouvre dans une Frame et non dans un Panel. Il en va de même pour la FenApplicationWorkflow, qui hérite de FenApplication.

FenApplicationWorkflow

La fenêtre application workflow est là pour encapsuler les différentes nécessités des workflows, en changeant notamment le Slider, qui permet désormais de se déplacer dans les Instructions.



Overlay

L'overlay est ce sur quoi vont être dessinées les informations, telles que les ROIs.

Si une Roi doit être dessinée, il faut bien faire attention à ce que sa position soit égale au numéro de slice voulue.

Une Roi stock la slice sur laquelle elle doit être dessinée, et n'ira pas sur l'Overlay d'une autre slice.

Si la position de la Roi vaut 0, elle se dessinera sur toutes les slice en même temps.

FenResults

La FenResult est une JFrame générale qui permet d'uniformiser le code et l'interface.

Tous les résultats sont présentés grâce à une FenResult., qui les affiche grâce à un JTabbedPane.

Ces résultats sont mis dans des TabResult.

SidePanel

ResultContent

CaptureButton

Le capture button est un élément courant dans les TabResult de la FenResult.

Il permet de déclencher la sauvegarde de l'ensemble des données des programmes, à savoir le nouveau DICOM tiré de l'ImagePlus, le CSV tiré du modèle, les ROIs et les Workflows tiré du RoiManager et ControllerWorkflow, et la capture de la fenêtre sur laquelle a été appuyé le bouton de capture.

Les éléments à afficher ou à masquer au moment de la capture, ou les informations additionnelles utiles à la sauvegarde sont gérés dans chaque TabResult.

Cet élément CaptureButton a été créé pour stocker aisément la TabResult à laquelle il est associé.

ContrastSlider

Le ContrastSlider est l'élément qui va afficher dans le ResultContent l'image transmise en paramètre, et qui pas permettre de jouer sur son contraste grâce aux valeurs du Slider.

Le ContrastSlider crée une image dupliquée de celle passée en paramètre, l'ouvre pour avoir accès au Canvas¹ et cache cette copie ouverte.

A chaque modification du Slider, le contraste est modifié sur la copie cachée et une capture est prise de cette copie, afin de remplacer l'image du ResultContent par la capture.

DocumentationDialog

Cette fenêtre de documentation est affichée lorsque l'utilisateur clique sur le menu Help dans la FenApplication. Cette fenêtre permet de lister les références utilisées pour le modèle de chaque étude.

Pour ajouter cette fenêtre à un nouvel examen, il suffit d'en créer une nouvelle instance dans la classe de Scintigraphy de l'étude créée, puis de l'ajouter à la FenApplication dans la méthode `start()` de la Scintigraphy en faisant appel à `setDocumentation(DocumentationDialog)` de la classe FenApplication.

DynamicImage

C'est un JPanel permettant d'afficher une ImagePlus tout en adaptant sa taille lors du redimensionnement de la fenêtre.

Model

ModelWorkflow

Le ModelWorkflow sert à faciliter les échanges de données avec le contrôleur.

Généralement, le modèle va proposer des constantes publiques de Result, pour chaque résultat qu'il peut fournir. Lorsque le contrôleur désire obtenir un résultat du modèle, il fait appel à la méthode `getResult(ResultRequest)` en passant en paramètre une requête d'un résultat. Une requête permet de faire transiter des données nécessaires au modèle pour retourner le résultat (par exemple le fit à utiliser pour extrapoler les résultats, ou bien l'unité dans laquelle le résultat doit être retourné, etc).

Data

Cette structure de donnée permet de stocker différentes valeurs calculées pour une image. Ainsi, une Data est associée à une image, et contient une liste de régions. Ces régions correspondent généralement aux ROIs dessinées par l'utilisateur. Les régions peuvent être soit sur la face Antérieure, soit sur la face Postérieure.

Nota bene : Si l'orientation Ant/Post n'a pas de sens pour un examen, il est toujours possible de n'utiliser que l'Ant (ou que la Post).

Cette structure devrait normalement être remplie au fur et à mesure que les données sont insérées dans le modèle (mais elle peut tout aussi bien être remplie lors de l'appel à `calculateResults()`).

Region

Une région est généralement une ROI dessinée par l'utilisateur. Les Region sont utilisées par la classe Data pour représenter ses données.

Chaque région stocke des données comme le nombre de coups présents, le pourcentage ou encore le nombre de pixels... Chaque donnée est représentée par un entier. Une liste de valeurs déjà réservées est présente dans la classe Data. Pour utiliser des données personnalisées, il suffit d'utiliser un entier supérieur à 1000.

Result

Un Result est un résultat que fournit un modèle. Chaque modèle propose une liste de résultats publics qu'il est nécessaire d'utiliser dans les requêtes.

ResultRequest

Une requête justement, est ce qui est utilisé pour demander un résultat au modèle. Chaque requête possède le Result à laquelle elle est associée, ainsi qu'un jeu de données dont le modèle a besoin pour traiter cette requête.

On peut notamment retrouver l'unité demandée en retour ou encore le fit à utiliser si la valeur doit être extrapolée.

ResultValue

Le ResultValue est la réponse du modèle à une requête.

La réponse est retournée en fonction du Result passé dans la requête.

Unit

Unité utilisée pour afficher les résultats. Différentes unités sont disponibles, et les conversions entre les unités sont gérées par cette classe.

JFreeChart

Selector

Les Selector sont des barres verticales qui permettent de sélectionner une valeur en Y et d'afficher en bas de la bar verticale la valeur en X

YSelector

Le YSelector est un dérivé du Selector et permet d'avoir une barre horizontale. Selon ce qui a été demandé, elle permet d'afficher le $X_{1/2}$ à partir du dernier point sur la courbe qui touche la barre Y.

Cela sert notamment à afficher les $T_{1/2}$

Fit

Un Fit est une translation des valeurs données par les acquisitions sur un modèle mathématique.

Deux fits sont actuellement disponibles : linéaire et exponentiel.

Le fit permet d'extrapoler des valeurs à partir de coefficients déterminés par la courbe obtenue à partir des données des images.

Save & Load

La partie `SaveAndLoad` permet de sauvegarder les différentes parties du programme, à savoir le CVS tiré du modèle, le DICOM suite à l'examen, une image *.png* de la `TabResult` sur laquelle le `CaptureButton` a été appuyé, et les ROIs et les Workflows de cet examen.

Je ne détaillerai ici que la partie réalisée pendant ce stage, à savoir la sauvegarde et le chargement des ROIs et des Workflows.

Sauvegarde

La sauvegarde se fait dans un fichier *.zip*, en parcourant toutes les ROIs du `RoiManager` et en les enregistrant, puis en sauvegardant le `ControllerWorkflow`.

Sur des examens permettant de lancer d'autres examens, le procédé choisi a été de n'avoir qu'un seul modèle et de sauvegarder les différents `ControllerWorkflow` avec à l'intérieur de chaque *.zip* l'entièreté des ROIs du `RoiManager`.

Ensuite est sauvegardé le `ControllerWorkflow` au format `Json`.

Ceci est fait pour chaque `ControllerWorkflow`.

Le format `Json` a été choisi parce que c'est un standard, et qu'il est simple d'utilisation sous tous les langages.

Chargement

Le chargement du Json en classe de fait de manière automatique grâce à la librairie de Google : Gson.

Pour que cela soit fait, il a fallu créer des classes cohérentes avec les données exportées.

Ces classes sont :

- `WorkflowsFromGson` : Stock la List des `WorkflowFromGson` et permet de faire des opérations dessus, et stock également les informations de l'examen et du patient.
- `WorkflowFromGson` : Représente un `Workflow`, qui stock la List des `InstructionFromGson`
- `InstructionFromGson` : Représente une `Instruction`. Stock le type d'instruction, l'index de la Roi, le nom de la Roi, et le nom de sauvegarde de la Roi dans le fichier .zip

Le chargement se fait ensuite séquentiellement, en parcourant à la fois les `InstructionFromGson` et les `Instructions` du `ControllerWorkflow` dans lequel on charge la sauvegarde.

Toutes les `Instruction` du `ControllerWorkflow` qui ne sauvegardent pas de Roi sont ignorées.

Un cas spécifique est géré : celui de l'examen Cardiac.

En effet, dans cet examen, il peut y avoir des `Workflow` ne contenant pas d'instruction qui sauvegardent des Roi, car une instruction préliminaire demande à l'utilisateur si il veut sauvegarder des ROIs. Si l'utilisateur répond non, alors aucune instruction n'est générée, et si il répond oui, des instructions sont générées.

Une gestion de ce cas est faite :

Je parcours toutes mes instructions

 Si je suis dans un examen Cardiac

 Si je suis sur une instruction préliminaire

 Si je n'ai pas d'autres instruction que la préliminaire dans mon workflow

 Si les `InstructionFromGson` me disent de générer une instruction

 Je génère l'instruction

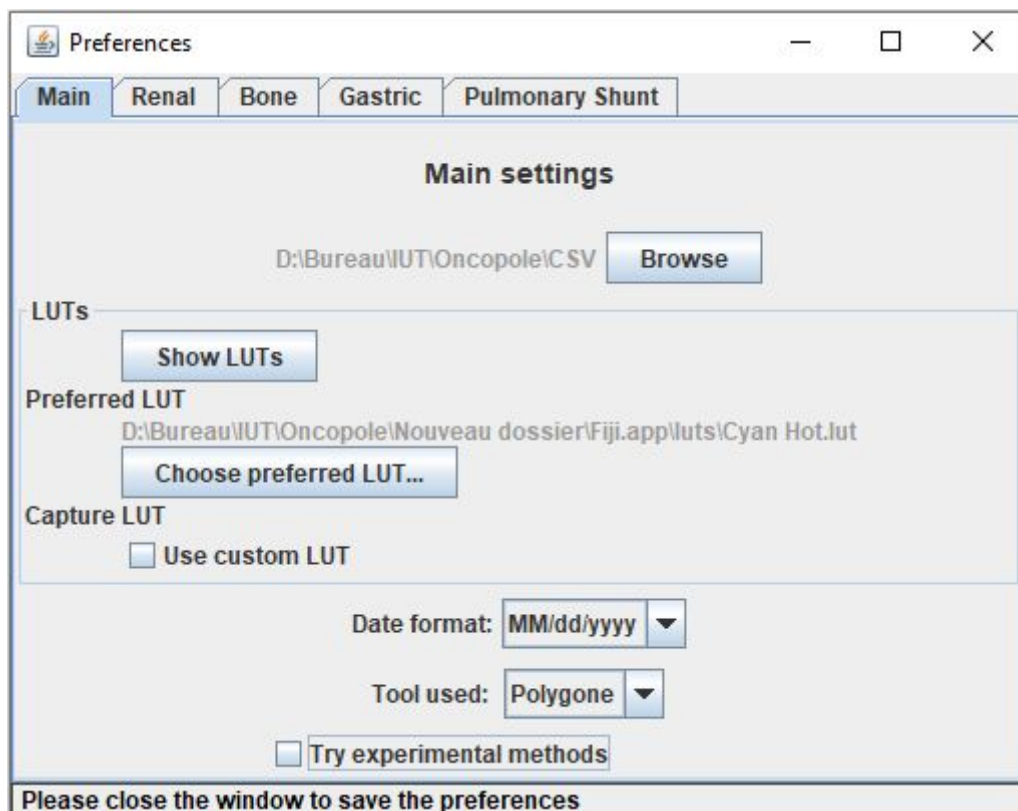
Une batterie de test par rapport au patient, à la date de l'examen, à l'ID du patient et à son accession number est faite avant de charger les ROIs et le `ControllerWorkflow`, demandant si l'utilisateur veut continuer en cas de différences entre les informations actuelles et les informations sauvegardées.

TabPreference

La TabPreference est un panel accessible en dehors de tout examen, et qui permet de gérer certaines préférences générales, ou liées plus spécifiquement à chaque examen.

Le choix a été fait de sauvegarder les préférences lors de la fermeture de la JFrame uniquement, afin d'éviter toute écriture inutile sur le disque.

Un message explicite le précise lorsqu'une préférence est changée.



Ces TabPref sont des JPanel qui sont ajoutés à la PrefWindow.

Chacune des Tab gère ses propres préférences par l'appel de la fonction `Prefs.get(PREF_SAVE_DIRECTORY, "Save Directory")` ;

Par exemple ici pour le répertoire de sauvegarde des données de la TabMain.

Le fichier sauvegardant les préférences est un fichier texte qui est géré par ImageJ.

Bonnes pratiques

Ci-dessous une liste des bonnes pratiques à adopter :

- Toujours privilégier l'utilisation de constantes (mot-clef `final`) définies dans la classe au lieu de constantes magiques.

Exemple : remplacer `afficherImage(0)` par `afficherImage(IMAGE_REIN)`

- Ajouter la JavaDoc
-