# TP 7 : Sauvegarder le modèle d'usine robotisée

**Dominique Blouin, Télécom Paris, Institut Polytechnique de paris**
dominique.blouin@telecom-paris.fr

Au cours du TP précédent, vous avez modifié votre modèle OO d'usine robotisée pour que les robots puissent se déplacer d'un composant de l'usine à l'autre. Vous avez également implémenté les interfaces MVC fournies par l'interface graphique *canvas viewer* afin que les déplacements de ces robots puissent être automatiquement visualisés.

Au cours de ce TP, vous allez développer la **couche d'accès aux données** (ou de persistance), de votre simulateur afin de pouvoir stocker les données du modèle d'usine robotisée sur un support permanent (disque SSD ou base de données). Ces données pourront ainsi être relues lorsque le simulateur sera redémarré.

Cette couche de persistance s'appuiera sur les différentes classes de Java servant à réaliser les **entrées et sorties de données**, comme l'écriture et la lecture de données dans des fichiers, sous format texte ou binaire.

## Introduction aux entrées-sorties en Java et à la persistance des données

Lire la présentation sur les entrées et sorties en Java disponible ici. Puis lire la présentation sur la persistance des données ici.

## Mettre à jour la bibliothèque *Canvas Viewer*

D'abord télécharger la dernière version de l'interface graphique *Canvas Viewer* qui se trouve ici. Puis remplacer votre version actuelle par cette nouvelle version. Vous constaterez des différences dans les interfaces du modèle de canevas que nous allons expliquer plus loin dans ce document.

## L'interface du contrôleur

L'interface *CanvasViewerController* possède une nouvelle méthode nommée *getPersistenceManager* tel que déclarée dans le code suivant :

```
/**
 * Returns the persistence manager to be used to persist this canvas model into a data
 * store.
 * @return A non {@code null} {@code CanvasPersistenceManager} implementation for the
 * desired data storage kind (file, database, etc.).
 */
CanvasPersistenceManager getPersistenceManager();
```

Cette méthode doit retourner un objet d'une classe qui devra implémenter l'interface *CanvasPersistenceManager*.

## Implémenter l'interface de persistance

Examiner la documentation de l'interface *CanvasPersistenceManager*. Tel que vu dans la présentation sur la persistance des données, cette interface spécifie des signatures de méthodes décrivant la lecture, l'écriture (*persist*) et la suppression d'un modèle de canevas.

Pour ce simulateur, nous allons implémenter une couche de persistance de données s'appuyant sur le **système de fichiers** de l'ordinateur qui exécute le programme. Afin de simplifier l'implémentation de l'interface *CanvasPersistenceManager*, créer une classe qui hérite de la classe abstraite *AbstractCanvasPersistenceManager*. Cette classe est fournie par la bibliothèque *canvas viewer*.

### L'interface *CanvasChooser*

La classe *AbstractCanvasPersistenceManager* contient un attribut de type *CanvasChooser*, et son constructeur prend en paramètre un objet de ce même type. Examiner l'interface *CanvasChooser* et sa documentation. Elle spécifie une méthode *chooseCanvas*, qui retourne une chaîne de caractère identifiant de manière unique un canevas.

Il n'est pas nécessaire de fournir une classe qui implémente l'interface *CanvasChooser*. Puique les modèles seront stockés dans le système de fichiers de l'ordinateur, vous pouvez simplement instancier la classe *FileCanvasChooser*, qui implémente *CanvasChooser*, et qui est fournie par la bibliothèque *canvas viewer*.

La classe *FileCanvasChooser* permet de parcourir le système de fichiers de l'ordinateur et de sélectionner un fichier dont l'extension attendue peut être spécifiée par un paramètre du constructeur de la classe. En effet, lorsqu'un utilisateur de l'interface graphique *Canvas Viewer* sélectionnera le menu *File>>Open Canvas*, l'objet de type *FileCanvasChooser* sera utilisé pour présenter à l'utilisateur une liste de fichiers et de dossiers, qu'il pourra parcourir pour choisir un modèle à visualiser.

### Modéliser un identifiant unique du modèle

Pour stocker vos modèles dans le système de fichier (qui joue ici le rôle de base de données), chaque instance de votre modèle d'usine devra fournir une chaîne de caractères qui servira à l'identifier de manière unique dans le système de fichier. A cette fin, les méthodes *getId* et *setId* ont été ajoutées à l'interface *Canvas*, de sorte que votre classe représentant l'usine robotisée (*Factory*), et implémentant l'interface *Canvas*, devra fournir un attribut servant d'identifiant, ainsi que les accesseurs de cet attribut.

Lors de la sauvegarde du modèle via l'interface graphique *Canvas Viewer*, l'objet de type *FileCanvasChooser* sera également utilisé afin que l'utilisateur puisse choisir un nom de fichier existant dans le système de fichier, ou encore saisir un nouveau nom de fichier. Cet identifiant récupéré de l'objet *FileCanvasChooser* sera ensuite valorisé dans le modèle d'usine par l'appel de la méthode *setId* de votre classe. Il sera constitué du nom du fichier et de son chemin dans l'arborescence des dossiers.

### Rendre les classes du modèle sérialisables

Afin de simplifier la représentation de votre modèle dans un fichier, vous allez utiliser la *sérialisation des données*, telle que vue dans la présentation du cours sur les entrées et sorties de données.

A cette fin, toute classe de votre modèle devant être sauvegardée devra implémenter l'interface *Serializable*. Modifiez donc les classes de votre modèle pour leur faire implémenter cette interface. Remarquez que cette interface est une interface de **marquage**, qui ne déclare donc aucune méthode et ne sert qu'à marquer des classes.

Il arrive cependant que certaines des données des classes ne nécessitent pas d'être mémorisées d'une exécution à l'autre du programme. C'est notamment le cas des **observateurs** du modèle. Il est en effet préférable de ne pas stocker ces objets, d'autant plus qu'ils consistent en des instances des classes d'interface graphique Java, qui ne sont pas toujours sérialisables.

Utiliser donc le mot clé *transient* pour déclarer que l'attribut des observateurs ne doit pas être sérialisé. Ne pas oublier de modifier l'instanciation de l'attribut pour utiliser une instanciation **paresseuse**, étant donné que lors de la désérialisation de l'objet, aucun constructeur de classe ne sera appelé, tel que vu dans la présentation sur les entrées-sorties.

Faites de même pour tous les autres attributs de vos classes du modèle qui ne doivent pas être sérialisés.

### Implémenter les méthodes de persistance

La classe abstraite *AbstractCanvasPersistenceManager* ne fournie pas de corps pour les méthodes *read*, *persist* et *delete* de l'interface *CanvasPersistenceManager*. A vous de les implémenter dans votre classe de persistance en utilisant les flux de sérialisation d'objets *ObjectInputStream* et *ObjectOutputStream* tels qu'introduits dans la présentation sur les entrées-sorties.

En ce qui concerne la méthode *delete*, une manière simple de faire consiste à instancier un objet de la classe *java.io.File* et d'appeler la méthode *delete* sur cet objet.

## Tester la persistance du modèle

Afin de tester la persistance de votre modèle, lacer le simulateur tel que vous l'avez fait au TP précédent. Utiliser les sous-menus *Open Canvas* et *Save Canvas* du menu *File* de l'interface *Canvas Viewer* et vérifier que vous pouvez bien sauvegarder le modèle sur disque, puis le réafficher correctement dans l'interface graphique.

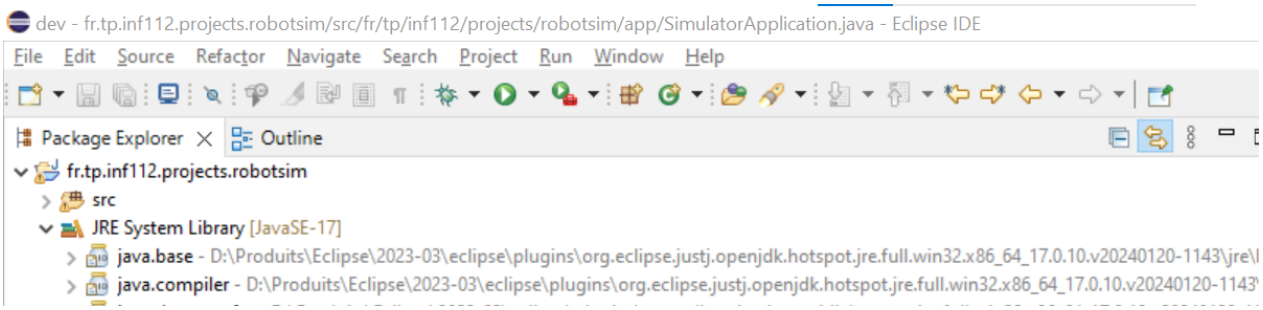
## Mettre en place le logging de l'application

Nous avons vu en classe qu'il est préférable d'utiliser une bibliothèque de logging pour afficher les traces d'exécution du programme, plutôt que d'utiliser directement la sortie console du système, et ce afin de pouvoir stocker les traces dans d'autres supports tels que des fichiers par exemple. Dans cette partie de l'exercice, vous allez mettre en place un logging à la fois à la console et dans un fichier log en utilisant et en configurant le logger du JDK.

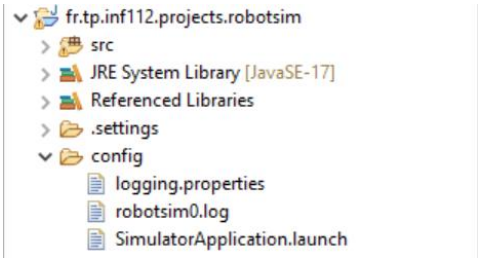
### Configurer le logger

Tel que vu en classe, le logging peut être configuré en code Java, ou via un fichier de configuration. Pour toute installation d'une JRE, un fichier de configuration par défaut est fourni dans le répertoire d'installation de la JRE. Vous allez copier ce fichier dans un répertoire du projet de votre simulateur, puis le modifier pour adapter la configuration du logger à votre besoin.

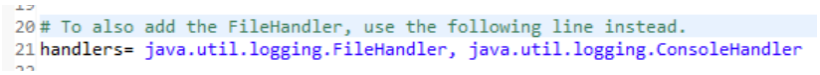
Tel qu'illustré par la capture d'écran suivante, ouvrir le répertoire de bibliothèque *JRE System Library* de votre projet de simulateur, afin de connaître le répertoire d'installation de la JRE dans le système de fichier de votre ordinateur. A partir de ce répertoire, naviguer vers un sous-répertoire nommé *conf* et copier le fichier *logging.properties*.



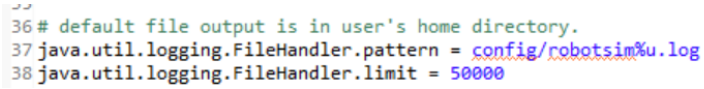

Créer un répertoire nommé *config* dans votre projet et y déposer le fichier *logging.properties* précédemment récupéré tel qu'illustré par la capture d'écran suivante :



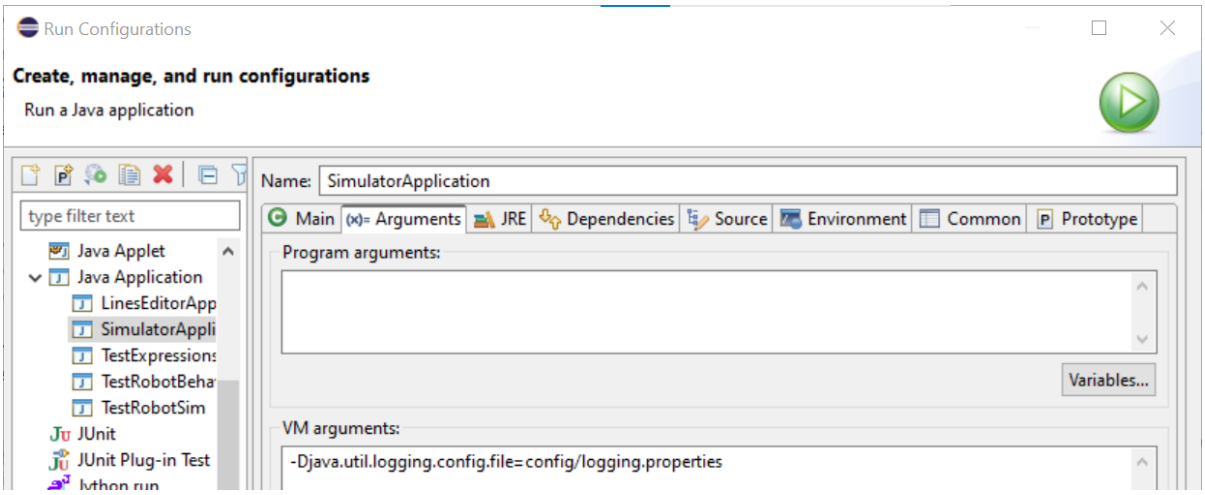

Ouvrir le fichier *logging.properties* dans Eclipse et modifier le fichier pour que deux handlers soient utilisés ; un premier handler qui écrira à la console, et un second qui écrira dans un fichier, tel qu'illustré par la capture d'écran suivante.




Ensuite, tel qu'illustra par la capture d'écran suivante, spécifier le chemin du fichier log pour qu'il soit créé dans le répertoire *config* du projet du simulateur, puis sauvegarder le fichier de configuration du logger.

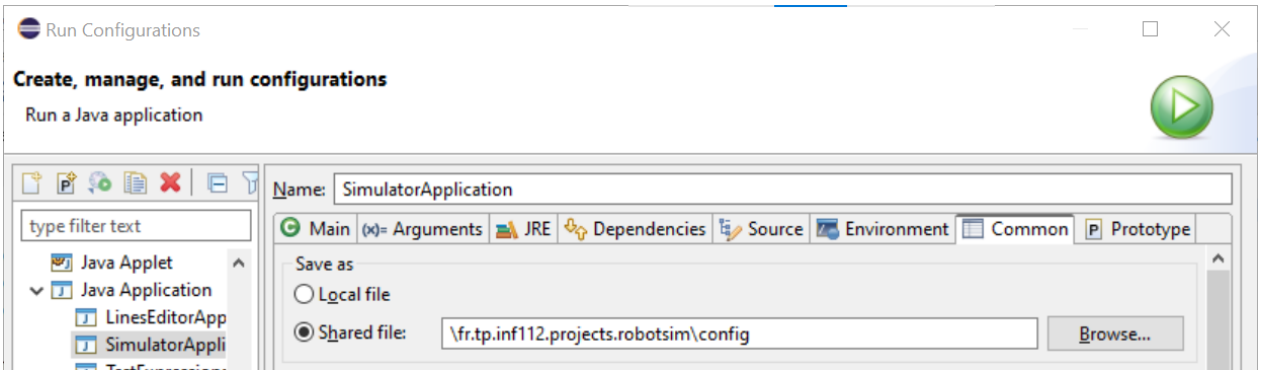



Puis spécifier l'endroit du fichier de configuration de logging à utiliser par votre simulateur. Cela peut se faire en spécifiant la valeur de la propriété de la *java.util.logging.config.file* telle qu'illustrée par la capture d'écran suivante :




Finalement, sauvegarder également votre configuration de lancement du simulateur dans votre projet. Cela facilitera l'évaluation de votre projet par les enseignants, qui pourront utiliser directement votre configuration de lancement pour exécuter votre simulateur et vérifier son bon fonctionnement.

Cela se fait en sélectionnant l'onglet nommé *Common*, puis en spécifiant le répertoire *config* dans le champ de texte de l'option *Shared file* tel qu'illustré par la capture d'écran suivante :




## Définir un objet de type *Logger* pour afficher les messages

Dans la classe de l'application de votre simulateur (*SimulatorApplication*), créer une variable pour le logger telle de celle de la ligne suivante :

```
private static final Logger LOGGER = Logger.getLogger(Main.class.getName());
```

Puis, afficher deux messages d'information notifiant le démarrage de votre simulateur, de deux niveaux de détail différents : INFO et CONFIG, tel que donné dans le code suivant :

```
LOGGER.info("Starting the robot simulator...");

LOGGER.config("With parameters " + Arrays.toString(args) + ".");
```

## Vérifier le bon fonctionnement du logger

Exécuter votre simulateur et observer les messages qui s'affichent à la console. Dans l'explorateur de projet de Eclipse, rafraîchir le dossier *config*. Le fichier log généré par le logger devrait apparaître. Ouvrez-le et vérifier les messages qui y sont écrits. Est-ce que tous les messages s'affichent bien dans la console et dans le fichier log ? Si non, pourquoi ?

Examiner le contenu du fichier *logging.properties* et modifier le niveau de logging au niveau CONFIG et vérifier que les deux messages s'affichent bel et bien dans le fichier log.

Puis spécifier un niveau de logging plus fin (FINE par exemple) et relancer le simulateur. Est-ce que les messages de logging du simulateur s'affichent toujours ? Vous constaterez que plusieurs messages en provenance de l'interface graphique s'affichent maintenant à ce niveau de détail. Si les messages du simulateur ne s'affichent plus, examiner la valeur de la propriété *java.util.logging.FileHandler.limit* et rechercher sa documentation pour corriger le problème.

Noter que le fichier log est écrit au format xml, alors que les données sont écrites dans un autre format dans la console. Comment expliquez-vous cette différence ?