jHoover

L'aspirateur de site Web!

Documentation développeur

Sommaire

Description

jHoover est un logiciel de téléchargement de site web écrit en langage Java version 1.5. Il utilise des librairies annexes décrites dans le paragraphe **librairies utilisées**.

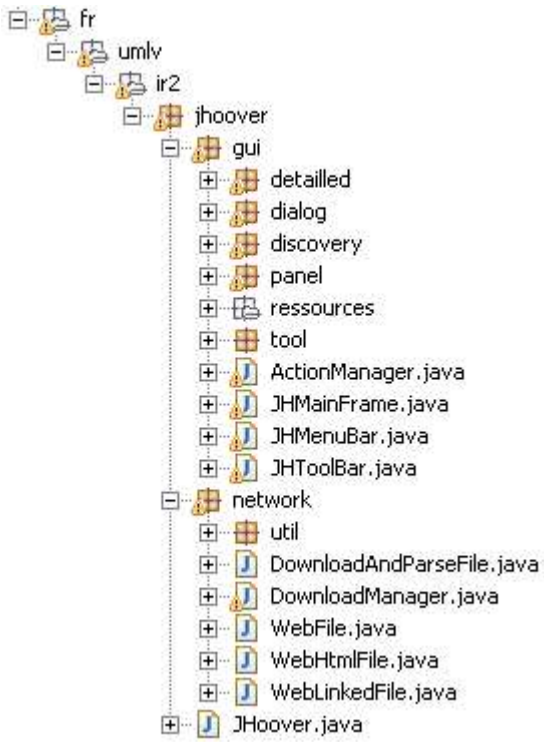
Structure du projet

jHoover se présente sous la forme d'un fichier ZIP nommé **jHoover.zip**. Il a la structure suivante:

- **./bin**: fichiers de lancement de l'application (.bat sous Windows et .sh sous Unix)
- **./docs**: contient les documentation utilisateur, développeur et la javadoc
- **./lib**: contient l'application jHoover et les librairies annexes utilisées
- **./src**: contient les sources du projet
- **./classes**: contient les fichiers .class de l'application
- **build.xml**: fichier de compilation du projet
- **readme.txt**: explique la structure des répertoires, comment compiler et exécuter le projet

Découpage des packages

Voici le découpage du projet jHoover:



- **gui**: ensemble des classes graphiques
    - **detailled**: classes relatives à la vue « fichiers détaillés »
    - **dialog**: ensemble des JDialog
    - **discovery**: classes relatives à la vue « fichiers parcourus »
    - **panel**: ensemble des JPanel
    - **ressources**: images et logos utilisés dans l'application
    - **tool**: contient des constantes et des méthodes statiques pour la partie gui
- **network**: classes métiers de l'application
    - **util**: contient des constantes et des méthodes statiques pour la partie métier

Classes métiers

La classe **JHoover** contient la méthode *main*. Au début, elle lance un *JhooverSplashScreen*. Elle lance ensuite la fenêtre de configuration de l'application *RunConfigDialog* qui nécessite de lancer la fenetre principale *JHMainFrame*. Lorsque l'utilisateur clique sur le bouton de démarrage du téléchargement, la methode *startDownload* de *JHoover* est appelée. Elle va créer le *DownloadManager et lui ajouter* le premier fichier à télécharger.

Le **DownloadManager** est un *Thread*, ce qui permet de le stopper, de lui mettre une pause ou de reprendre le téléchargement. Voici son fonctionnement: on ajoute un fichier dans la liste des fichiers a télécharger. Le *DownloadManager* va alors tester si le fichier respecte l'URL ainsi que le filtre de départ et va ajouter ce fichier dans les modèles. La liste de fichiers à télécharger est parcourue et un thread téléchargement du fichier est lancé (*DownloadAndParseFile*) pour chaque fichier. Il y a un contrôle du nombre de fichier en cours de téléchargement, un pour les fichiers HTML et un pour les fichiers liés. Ce thread permet de télécharger le fichier et de le parser, si c'est un fichier HTML, afin de récupérer les liens contenus dans la page. Ces liens sont ajoutés à la liste des fichiers à télécharger et ainsi de suite, jusqu'à ce que la liste des fichiers à télécharger soit vide, ou que le thread du *DownloadManager* soit stoppé. Le *DownloadManager* est détruit lorsqu'il se termine. A chaque nouveau téléchargement, il est reconstruit.

C'est le *DownloadManager* qui va initialiser les paramètres graphiques: il créé le *JTree* vide ainsi que les trois *JTable* vides correspondantes aux onglets « ALL », « HTML » et « FILES ».

La classe **WebFile**, qui représente un fichier à télécharger, est abstraite. Deux classes l'étentent: *WebHtmlFile* et *WebLinkedFile*. Respectivement ces classes représentent un fichier HTML et un fichier lié.

La sauvegarde de la configuration se fait dans la classe **JHooverConfiguration**. Elle utilise la librairie *Jakarta Common* pour récupérer et pour savegarder les paramètres. Le fichier de sauvegarde s'appelle *params.xml*.

Découpage de l'interface graphique

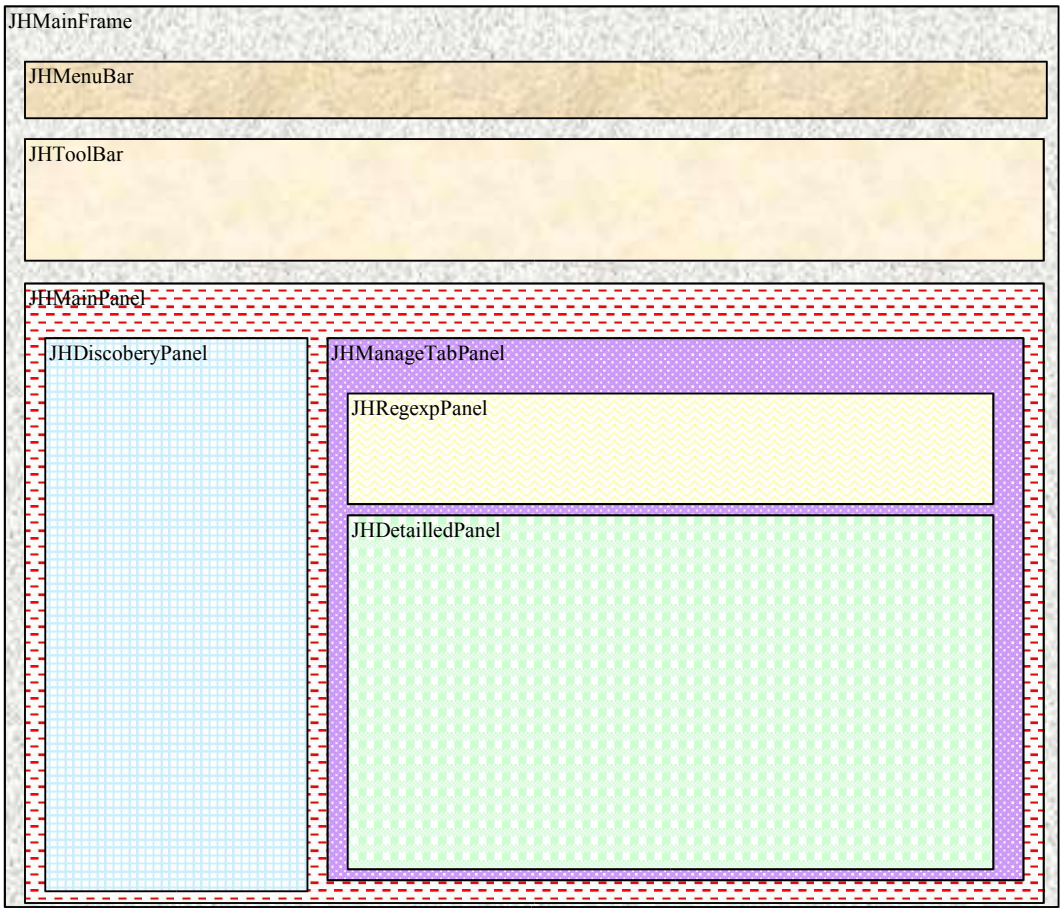
Voici le découpage de l'interface graphique du projet:

Découpage de la JFrame en deux parties avec un **BorderLayout**. Au nord la **JToolBar** et au centre un **JPanel** découpé avec un **JSplitPane** comme cela:

- A gauche un **JScrollPane** et un **JPanel** (contenant un **JTree**)

- A droite un **JPanel** avec un découpage en deux parties selon un **BorderLayout** (Un **JPanel** au nord (zone de création d'onglet dynamique) et un **JTabbedPane** au Centre qui représente la vue détaillée des téléchargements avec les différents onglets). Dans ce **JTabbedPane** on trouvera, pour chaque onglet, une nouvelle **JTable**.

Voici le nommage des composants:



Structure de l'interface graphique

Le projet est découpé en deux modèles distincts, qui forment deux **MVC** (Modèle Vue Controleur):

- **DefaultTreeModel**: ce modèle est utilisé pour le *JTree* de la vue « fichiers découverts ». Ce modèle, passé en paramètre a la construction du *JTree*, est construit avec un *DiscoveryTreeNode*. C'est un noeud qui contient un *WebFile* (unité représentant un fichier) comme donnée. Un *DiscoveryRenderer* est appliqué au *JTree* pour lui donner un bel aspect.

- **DetailledModel**: ce modèle est utilisé pour les différentes *JTable*. Chaque *JTable* est construite avec ces données, filtrées par un adapteur, nommé *DetailledAdapter*. Cet adapteur filtre les données en fonction de l'expression régulière passée en paramètre dans sa construction. La *JTable* necessite deux renderers, un pour le rendu de la barre de progression (*DetailledJProgressBarRenderer*) et un pour le rendu du bouton (*DetailledJProgressBarRenderer*). Le bouton necessite aussi un editor (*DetailledJButtonEditor*) pour associer une action au clic.

Les **boites de dialogues** sont toutes créées à partir d'une classe abstraite nommée *AbstractDialog*, ce qui permet d'avoir les même propriétés pour chaque boite de dialoque sans les redéfinir. La classe *AbstractConfigDialog* étend de l'*AbstractDialog* mais est aussi une classe abstraite: deux boites de dialogues similaires étendent de cette dernière (*RunConfigDialog* et *SetConfigDialog*), ce qui justifie ce choix. Les boites de dialogues sont construites avec la librairie *jgoodies*.

Elles ont les même propriétés:

- non redimensionnable
- centrée par rapport à l'écran
- prenant en paramètres plusieurs *JPanel* afin de les placer convenablement
- effectuer la même action pour tous les boutons *Cancel*

Les **actions** sont, pour la plupart, réunies dans une classe *ActionManager*. Ceci permet de concentrer les actions, d'éviter les redondances et les erreurs.

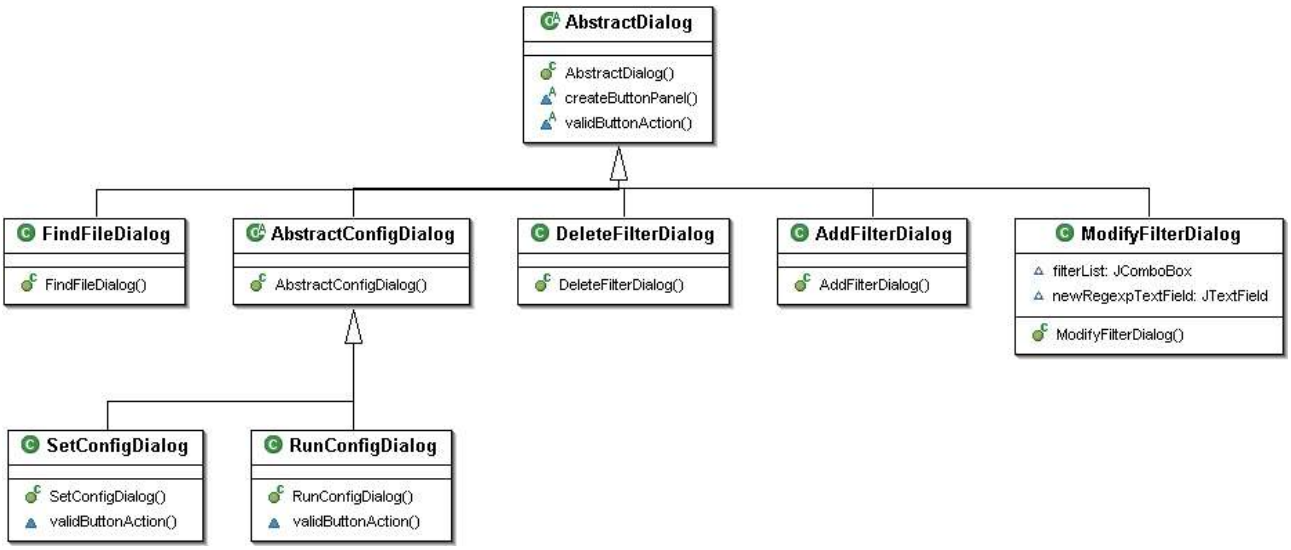
Les **labels et les icônes** sont réunis dans un package tool afin de simplifier tout changement graphique au niveau des icônes et des labels, dans les classes *Labels* et *Icons*.

Dans le projet on trouve un certain nombre de méthode **getInstance()**, qui vont retourner un objet unique (**singleton**) de la classe:

- **JhooverConfiguration**: gestion de la savegarde de la configuration XML
- **JHMainFrame**: fenetre principale du programme
- **JHDetailledPanel**: panel de la « vue détaillée »
- **JHoover**: lancement de l'application
- **DetailledModel**: model de la « vue détaillée »
- **DownloadManager**: gestion du téléchargement

Diagrames UML

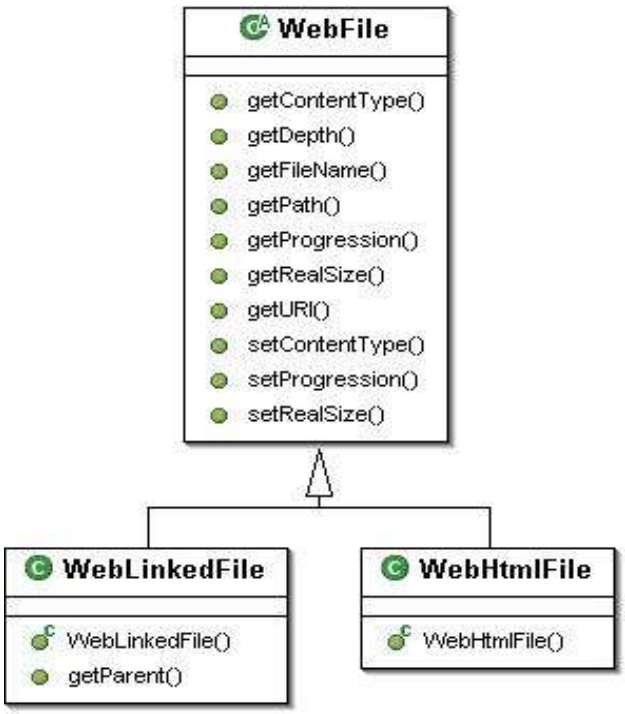
Voici quelques diagrames UML du projet jHoover:



Ce schéma montre l'héritage des boites de dialogues. Nous voyons qu'il y a une classe Abstraite principale. Toutes les autres boites de dialogues héritent de celle-ci. Elles doivent alors redéfinir les méthodes *validButtonAction()* (action associée au bouton OK) et *createButtonPanel()* (change suivant le nombre de boutons ajoutés).

Les classes *SetConfigDialog* et *RunConfigDialog* héritent de la classe *AbstractConfigDialog*, puisque seule la fonction sur le bouton OK est changée.

Le diagrame UML suivant montre encore un héritage:



Afin de différencier un fichier html et un fichier lié, j'ai créé une classe *WebFile* qui contient les méthodes et attributs communs. Par héritage, les classes *WebHtmlFile* et *WebLinkedFile* les possèdent. La classe *WebLinkedFile* possède aussi un attribut parent qui indique son fichier html parent.

Compilation, Execution et JavaDoc

jHoover est doté d'un fichier **build.xml**, qui permet de compiler l'application, de créer le fichier *jHoover.jar*, d'effacer les fichiers générés et de créer la *javaDoc* du projet.

Voici les *targets* possibles:

- **compile**: compilation de l'application (seulement les .class)
- **release**: création d'un fichier jar executable (défaut, dépend de compile)
- **clean**: suppression des fichiers générés
- **javadoc**: génération de la javadoc

Le lancement de l'application s'effectue par les fichiers script situés dans le répertoire *bin*. Sous Windows, on utilise le fichier *jHoover.bat* et sous Unix, le fichier *jHoover.sh*.

Librairies utilisées

Les librairies suivantes ont été utilisées:

- htmlparser.jar
- commons-configuration-1.1.jar
- commons-lang-2.1.jar
- commons-collections-3.1.jar
- commons-logging.jar
- forms-1.0.5.jar
- jakarta-regexp-1.3.jar

La librairie **HtmlParser** permet de parser facilement le html.

La librairie **JGoodies** permet de créer rapidement et facilement des boites de dialogues.

La librairie **Jakarta Regexp** permet d'utiliser facilement les expressions régulières.

La librairie **Jakarta Common** permet d'utiliser la sauvegarde des paramètres de configuration XML.

Bugs connus

Voici une liste des bugs connus dans jHoover:

- **Exception**

  lorsque l'application est en pause et que l'on veut arreter le téléchargement. Ceci est dû à une faute de programmation: le bouton resume lance un *Thread.interupt()* pour relancer le Thread en pause *Thread.wait()*. Il en est de même lorsque l'on lance un stop.

- **Probleme de Refresh**

  lorsque l'utilisateur appuie sur le bouton cancel d'un fichier en cours de téléchargement, il manque un refresh.

- **Pas de téléchargement**

  lorsque la page d'accueil du site n'est pas *index.html*, le téléchargement ne fonctionne pas; il faut alors entrer la page de démarrage dans l'adresse du site, dans la fenêtre de configuration.

- **Téléchargement incomplet**

  Mauvaise utilisisation de la librairie *HtmlParser* dans la classe *DownloadAndParseFile*. Ce qui fait, par exemple, que les feuilles de style ne sont pas téléchargées.

Améliorations possibles

Voici les options prévues mais non implémentées:

- **Find a file** (cf. doc utilisateur)
- **Colors** (cf. doc utilisateur)
- **Help** (cf. doc utilisateur)
- **About** (cf. doc utilisateur)

Voici les options demandées mais non implémentées:

- Rafraichissement du *JTree* en fonction de l'onglet sélectionné
- Sélection dans la *JTable* lors d'un clic sur un object du *JTree*
- Clique droit sur un fichier pour voir les propriétés (cf. *Find a file*)

Amélioration possible sur l'existant:

- faire un seul modèle pour le *JTree* et pour les *JTable* avec des *adapteurs*
- Changer les *ArrayList<>* du *DownloadManager* en *HashMap* pour associer un état au WebFile
- Sauvegarde des extentions (musique, video, ...) dans un fichier XML