Projet d'informatique 2ème année AMER/EUR
Informations diverses

Nicolas Stouls, Patrick Pollet et Bruno Mascret **modifié le 05/05/10**

Table des matières

Ces quelques pages ont pour objectif de vous assister tout au long de votre projet, mais surtout de vous laisser des traces pour d'éventuels besoins que vous auriez durant les années à venir. Elles ne sont qu'un survole des points importants découverts pendant le projet.

Le projet informatique de seconde année d'AMERINSA et d'EURINSA se déroule sur l'intégralité du second semestre. À travers ce projet, nous voulons d'une part vous forcer à manipuler les compétences acquises durant vos deux premiers semestre de programmation et d'autre part vous aider à développer de nouvelles compétences, parmi lesquelles :

- Initiation au **travail collaboratif** à travers un projet de longue durée et en utilisant des outils adaptés au travail réparti et/ou mobile ; Initiation au travail **autonome** où aucun sujet de TD ne guide chacun de vos pas et où l'enseignant n'est pas systématiquement disponible ;
- Initiation à la **conception** d'une application, depuis l'interprétation du cahier des charges jusqu'au choix de la signature de chaque méthode du programme final ;
- Réalisation de programmes destinés à être publiés sur le web, sous la forme d'une **Applette ou d'une application Java Web Start**, avec toutes les contraintes que cela suppose.

Ce projet se déroule sur l'intégralité du second semestre et se décompose en 5 phases, décrites dans le planning disponible sur la page web du cours. Tout au long du projet, l'enseignant fera un suivi des différents groupes. Attention, ce suivi n'est **en aucun cas un soutien à la programmation** java ! L'objectif est de suivre vos raisonnements pour vous aider à ne pas partir dans une mauvaise direction. Voici un rapide rappel des différentes phases du projet :

1. Définition du cahier des charges
2. Apprentissage des outils nécessaires pour le projet
3. Conception du programme
4. Développement du programme et du site web d'hébergement

5. Finalisation du programme

Ces différentes phases sont validées par des rendus permettant d'évaluer la note **du groupe** au projet :

- Rendu du cahier des charges (description de ce que veut faire le groupe)
- Rendu du rapport de conception (description de comment va s'y prendre le groupe)
- Rendu du site web
- Rendu de l'application
- Soutenance orale

Les points 1 et 2 permettent d'obtenir une note de TD, tandis que les rendus 3, 4 et 5 permettent de calculer une note comptant comme un DS.

1. Travail collaboratif : utilisation de subversion (SVN)

Avant même de comprendre comment cela marche et à quoi ça sert, il vous faut obtenir un espace de stockage sur les serveurs :

```
« Afin d'obtenir un espace pour héberger votre projet sur le serveur
SVN du Premier Cycle, envoyer un mail à cipc@insa-lyon.fr en indiquant
le nom du dépôt ainsi que les utilisateurs autorisés en écriture
(login insa). »
```

a) Principe général

L'outil de travail collaboratif SVN se compose d'un serveur, installé sur les serveurs du CIPC, et d'un client, utilisable en ligne de commande sur les terminaux du CIPC. Jdev fournit une interface graphique pour la gestion de cet outil pour la gestion des applications faites sous Jdev.

Le principe de fonctionnement est le suivant : le serveur héberge un *dépôt* qui contient la copie de référence des sources de l'application en cours de développement. Les différents partenaires peuvent accéder à tout moment à cette copie. Il suffit donc à un nouvel arrivant de récupérer cette version pour pouvoir ensuite participer au développement. Une fois ses modifications finies, le partenaire renvoie sur le serveur sa version modifiée. Le serveur mémorise alors cette nouvelle version comme étant celle de référence.

Si quelqu'un d'autre a fait des modifications du même fichier en même temps, alors le serveur sait identifier quelles parties du document ont été modifiées aussi bien sur la version de référence que sur la version client, et peut alors fusionner les deux versions. Cependant, il est nécessaire que les deux partenaires n'aient pas modifié simultanément la même partie d'un document.

Définition : on appel indifféremment *dépôt*, *repository* ou *archive distante*, la copie de référence stockée sur le serveur. De même, la *copie locale* peut être appelée *archive locale* du programme versionné.

Les commandes importantes sont les suivantes :

```
svn checkout repo.   récupération initiale du répertoire distant repo.
svn update     mise à jour du répertoire local vis à vis du distant
svn commit     envoie des modifications locales sur le serveur
svn status     indique les fichiers qui ont été modifiés localement
svn resolved   indique qu'un conflit a été corrigé
```

Points importants :

- Pour accéder au serveur SVN du CIPC (accessible également depuis l'extérieur) il vous est nécessaire de **demander l'ouverture d'un compte** auprès de cipc@insa-lyon.fr en précisant la liste des membres du projet (leur logins) ainsi que le nom du dépôt souhaité.
- Lors de la mise en place d'un dépot SVN, il est possible d'avoir déjà du code source, mais celui-ci ne doit pas être réparti sur plusieurs comptes. La version qui sera mise en place sous SVN par le responsable du groupe **devra nécessairement être récupérée par tous les autres**.

- Tout **commit devrait être précédé d'un update** pour limiter le risque de collisions qui vont faire échouer ce commit.
- Tout **commit doit être accompagné d'un message** permettant de relire ensuite l'historique du SVN.

b) Mise en place du SVN

Cette mise en place n'est à faire qu'**une seule fois**, par le responsable du projet. Une fois le svn mis en place, seul le dernier point est à faire sur chacun des postes de travail utilisés**.** Nous appellerons mon*SVN* le nom de votre dépôt SVN, *ApplicationDuProjet* le nom de l'application jdev contenant votre projet et *monProjet* le nom du projet jdev.

- **Créez** un projet Jdev de type « java application project », **puis quittez Jdev.**
- Allez dans le dossier de l'application contenant le projet créé *(dossier contenant le fichier .jws)*

```
cd ~/jdeveloper/mywork/ApplicationDuProjet
```

- Importez le projet créer dans le dépôt SVN *(avec un commentaire « init »)*

```
svn import monProjet http://cipcgw/svn/monSVN/monProjet –m "init"
```

- Sauvegardez le dossier du projet créé par jdev :

```
mv monProjet old_monProjet
```

- Récupérez une copie versionnée de votre projet :

```
svn checkout htt://cipcgw.insa-lyon.fr/svn/monSVN/monProjet
```

Si nécessaire, ouvrez le fichier projet (.jpr) depuis Jdev pour accéder au projet.

c) Travail sur une application versionnée

Lorsque l'on travail de manière collaborative sur un programme, il faut toujours garder à l'esprit que si plusieurs personnes modifient la même partie d'un fichier, alors le gestionnaire de version **ne saura pas** faire la fusion des documents. Il est donc nécessaire de suivre quelques bonnes pratiques simples :

- Il est nécessaire de se **répartir les tâches** en attribuant à chacun des participants une liste de fichiers qu'il peut modifier. Ne travaillant pas sur les mêmes fichiers, les conflits seront très rares.
- Chaque jour, **avant** de commencer à travailler, il faut toujours **commencer par vérifier s'il n'y aurait pas une version plus récente du projet** disponible sur le serveur.
- Avant d'arrêter de travailler, je m'assure toujours que la nouvelle version de mon programme **compile** et marche. S'il y a des restrictions d'utilisation alors je les signale par un commentaire. Ce n'est qu'à ce moment là que je peux envoyer cette nouvelle version sur le serveur SVN.

c.1) Ajout des nouveaux fichiers dans l'archive

Si de nouveaux fichiers ont été créés dans le dossier src ou l'un de ses sous-dossiers, alors il est nécessaire de préciser que l'on veut les ajouter dans les fichiers à synchroniser avec le serveur. Depuis le dossier src, la commande est la suivante :

```
cd jdeveloper/ApplicationDuProjet/nomProjet/src
svn add lesFichiersAAjouter
```

c.2) Récupération des modifications distantes

Chaque jour, **avant** de commencer à travailler, il faut toujours **commencer par vérifier s'il n'y aurait pas une version plus récente du projet** disponible sur le serveur. On commence donc toujours une session de travail par la commande :

```
svn update
```

c.3) Envoie des modifications locales

Après avoir modifié des documents et avoir vérifié que le programme compile et marche, alors les modifications peuvent être soumises sur le serveur avec la commande :

```
svn commit –m ''Ceci est un message''
```

Lors d'un commit, pensez à **toujours mettre un message** expliquant ce que vous avez effectué comme modification.

c.4) Résolution de conflits

Lors d'un commit, si les modifications entre en conflit avec d'autres modifications effectuées en parallèles, alors la commande commit le précise en proposant plusieurs possibilités. Choisissez l'option 'p' pour que le fichier en conflit soit annoté. Ces annotations sont indiquées entre chevrons sous la forme <<<<<mine ===== .r23 >>>>> et doivent être résolues par l'utilisateur. Une fois le conflit résolu, il faut le signaler au client SVN par la commande :

```
svn resolved monFichierEnConflit
```

c.5) Vérification de l'état de la copie locale

Afin de savoir l'état des fichiers locaux (conflit, modifié et non commité, non ajouté au dépôt, etc.) il est possible d'utiliser la commande :

```
svn status
```

d) Documentations externes

- Description du SVN disponible sur le CIPCNET :
  http://cipcnet.insa-lyon.fr/aide/cipchowto/subversion/
- Description des bonnes pratiques pour l'utilisation de SVN :
  http://cipcnet.insa-lyon.fr/aide/cipchowto/le-guide-du-programmeur-subversif/
- Livre de référence de SVN gratuitement disponible en ligne :
  http://svnbook.red-bean.com/

2. Rappel de quelques notions de conception

Concevoir un système consiste à comprendre ce que doit faire le système et comment agencer les différentes fonctionnalités pour simplifier son développement. Nous vous proposons d'utiliser plus particulièrement 4 outils cette année ; sachez qu'il en existe de nombreux autres, notamment en UML.

- Diagrammes de cas d'utilisation
- Descriptions de l'interface homme-machine
- Diagrammes de classes
- Diagrammes de Gantt

Chacun de ces outils a un rôle précis que nous rappelons. Pour leur mise en œuvre détaillée, nous vous renvoyons aux cours et exemples disponibles sur CIPCnet.

a) Méthodologie

Avant toute chose, il faut identifier clairement ce que votre logiciel va proposer comme fonctionnalités, et surtout quel est votre **sujet d'étude**. Pensez également à intégrer dès le début les **contraintes** de votre projet (langage, accessibilité, temps, ressources humaines, etc.). Prévoyez vos **cas d'études** composant votre sujet d'étude.

Une fois ce travail réalisé, vous pourrez proposez quelques **cas d'utilisation** pertinents, puis tenter un premier **diagramme de classes**. La **maquette de l'interface graphique** peut être réalisée avant, après ou en même temps que le diagramme de classes. Très souvent, **il est préférable de la réaliser après,** afin de ne pas privilégier l'aspect graphique de l'application au détriment de l'organisation.

Le diagramme de GANTT est un outil de gestion de projet, il doit être réalisé en parallèle du processus de modélisation qui l'intègre, et surtout mis à jour régulièrement !

D'une manière générale, **tous les diagrammes vont et doivent être maintenus du début jusqu'à la fin du projet.**

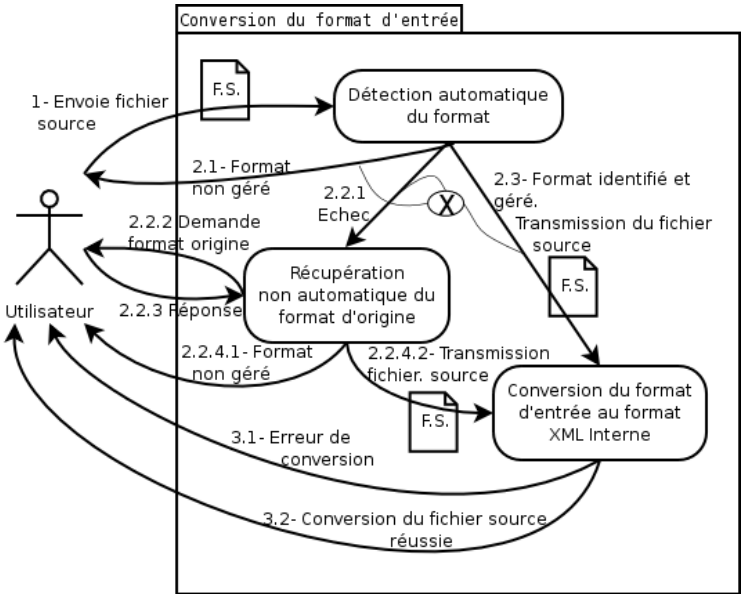
Dans la dernière partie de cette section, nous vous proposons différents outils pour la réalisation et le rendu des différents diagrammes.

b) Diagrammes de cas d'utilisation

Ce type de diagramme, souvent très simple en apparence, liste l'ensemble des fonctionnalités principale du projet et explicite les interactions entre les utilisateurs et le système. Il permet de mettre en évidence, pour chaque action, qui sont les déclencheurs du traitement et comment celui-ci sera organisé.

Les commentaires sur chaque action permettent de décrire plus précisément ce que celle-ci doit faire, et/ou ce dont elle requiert comme ressources (fichiers, bases de donnée, type de données). Les données produites ou transformées en sortie doivent également apparaître.

Ce diagramme est très utile pour faire le lien avec les descriptions de l'interface graphique prévue, comme décrit dans la section suivante. De plus, c'est un outil de communication qui a l'avantage de ne pas nécessiter de connaissances approfondies en informatique pour être lu.



Exemple de cas d'utilisation: conversion d'un fichier

Suivant le besoin, un cas d'utilisation peut être plus ou moins détaillé. Il convient donc de définir un niveau de granularité pertinent (ni trop simple, ni trop complexe). Un cas d'utilisation s'exprime en général graphiquement, mais peut aussi être textuel.

c) Descriptions de l'interface homme-machine

Afin de mettre au clair au plus tôt ses idées, il est important de réaliser, durant la phase de conception, des esquisses des différentes fenêtres que pourra rencontrer l'utilisateur. Il est particulièrement intéressant d'associer ces esquisses aux actions proposées dans le diagramme de cas d'utilisation. Cette réflexion permet de gagner beaucoup de temps au moment de la réalisation de l'interface, car elle permet de raisonner en dehors des contraintes de Java.

Le plus simple est tout simplement de réaliser des maquettes (dessins) de l'interface souhaitée, en précisant ce que déclenchent les différents éléments.

ATTENTION : il ne faut pas non plus que ce soit l'interface graphique seule qui conditionne l'organisation, notamment le diagramme de classes !

d) Diagrammes de classes

Ce type de diagramme a pour but d'aider les développeurs à prévoir les grandes phases de développement et faciliter la répartition des tâches. Les trois aspects à résoudre avec ce diagramme sont :

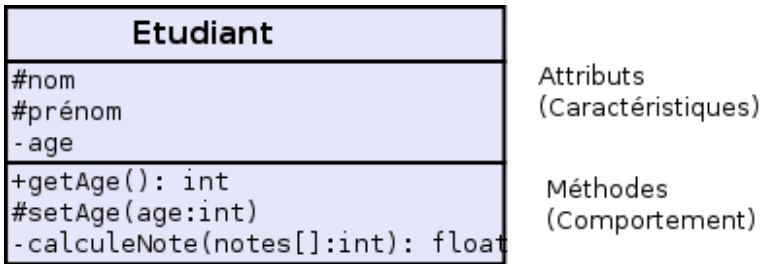
- quelles entités/classes faut il caractériser ?
- de quels comportements (méthode) et caractéristiques (attributs) a besoin chaque classe ?
- quels sont les liens entre les classes (héritage, implémentation, cardinalité, messages) ?

Une classe est un modèle en elle-même. Elle doit donc être réalisé en ayant en point de mire les caractéristiques de tout modèle :

1. la **représentativité** : la classe fonctionne pour tout cas de votre sujet d'étude doit ;
2. la **généricité** : lorsqu'un nouveau cas se présente, la classe fonctionne toujours ;
3. la **finalité** : la classe a été réalisée dans un certain but, il faut le préciser (par exemple, avec de la javadoc ou des annotations) ;
4. la **subjectivité** : la classe incarne le point de vue de son modélisateur, qui peut s'avérer limité, ou carrément ne plus convenir en cas de ***changement de paradigme*** ou d'approche !
5. son **langage de modélisation** est UML, pas vaguement UML ;-)...

Dans un premier temps, il ne faut surtout pas modéliser les aspects graphiques de l'application ! Ce n'est qu'une fois les fonctionnalités prises en compte qu'on se posera la question de comment les représenter sur le diagramme de classe. Chacune des fenêtres prévues dans les esquisses doit être associée explicitement à une ou plusieurs classes décrites et chaque action prévue dans le diagramme de cas d'utilisation doit également être réalisable par les services prévus.

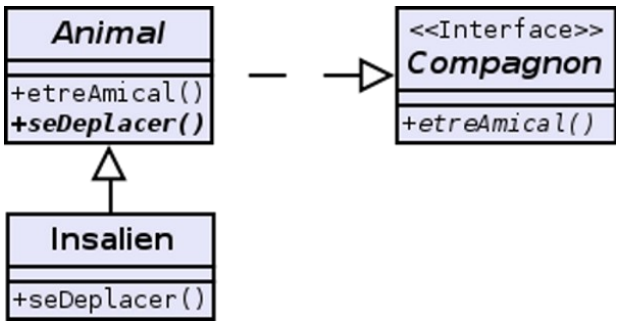
d.1) Représentation d'une classe :



Notez :

- que les visibilité s'expriment avec + (public), - (privé) et # (protected) ;
- que les types sont tous donnés ;
- que les signatures des méthodes sont complètes ;
- que le nom de la classe est explicite (préférer "PanneauDessin" plutôt que Jpanel368).

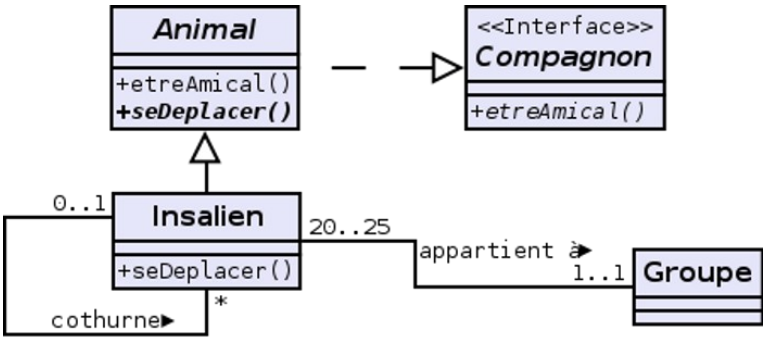
d.2) Héritage et implémentation



Une méthode abstraite est en ***italique gras*** ;

Une méthode à implémenter en *italique*.

d.3) Liens de cardinalité



Notez la possibilité d'un lien récursif (sur Insalien).

Vous trouverez également des liens de cardinalité plus contraignants du point de vue du modèle : les liens de composition et les liens d'aggrégation.

La composition indique que l'objet lié est une partie de l'objet cible (des élèves composent. un groupe, mais un groupe peut n'avoir aucun élève à un instant t).

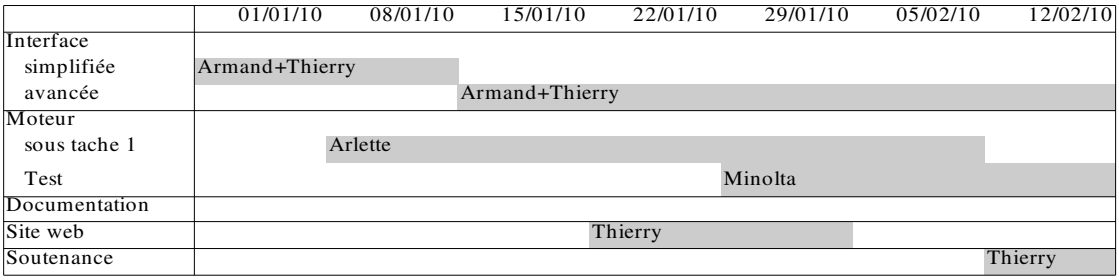
L'agrégation est plus forte : elle indique que l'objet cible ne peut pas exister si l'objet lié n'existe pas : par exemple, une voiture ne peut pas exister en tant que voiture si elle n'a pas un moteur.

Il n'est pas possible en java de coder directement une association de cardinalité. Il vous faudra donc choisir pour chaque lien dans quel sens vous souhaitez l'interpréter :

- soit de A vers B : la classe B aura donc un attribut de type A (ou une liste de type A) : Groupe a un attribut List<Eleve> ;
- soit de A vers B : Groupe ne connait pas ses élèves mais tout élève contient un attribut Groupe (c'est pas la meilleur solution dans cet exemple comme vous le voyez !)
- A et B se contiennent tous les deux : le groupe connait ses élèves et les élèves connaissent leur groupe. **Attention avec cette dernière possibilité** :la plupart du temps, c'est une erreur de conception ! De plus, il vous faudra choisir dans quel ordre créer les objets : A a besoin de B pour exister, mais B à besoin de A pour exister. Il faut donc créer A sans B, puis créer B avec A, et enfin mettre à jour A avec B.

e) Diagrammes de Gantt

Une fois que les différentes parties du programme final sont prévues *(liste des actions, interface utilisateur et liste des classes à développer)*, il faut rapporter cette conception au problème des ressources. Étant donné le travail à faire, le nombre de développeurs et la durée du projet, alors dans quel ordre faut il faire les choses et combien de temps peut on se permettre d'y passer ? Faut il prévoir de faire des « bouchons » ? Le diagramme de Gantt est un support aussi bien pour ce type de réflexion que pour en figer le résultat sous la forme d'un calendrier de développement.



Exemple de diagramme de Gantt

f) Outils pour la conception

Suivant les diagrammes, différents outils peuvent être utilisés pour vous aider à concevoir vos applications. Notre objectif n'est pas de vous apprendre à réaliser ces diagrammes de manière

professionnelle, mais uniquement de vous donner des outils vous permettant de simplifier/accélérer votre conception.

f.1) Papier+crayon

La plupart des diagrammes sont plus simples à faire, au moins dans une première version, sur papier. Alors faites le ! Ceci étant dit, il n'existe pas d'outil agréable et souple permettant de concevoir une **interface graphique**. Pour le rendu du rapport de conception, nous vous conseillons donc de joindre une (photo-)copie de ces dessins pour le rendu de vote conception d'interface graphique, sans rien faire à l'ordinateur. Il sera toujours temps plus tard d'immerger vos créations dans le carcan de Java et Jdev.

f.2) Jdev

Cet outil vous permet de faire facilement des **diagrammes de cas d'utilisation**, ainsi que des **diagrammes de classe**. Pour y avoir accès, il faut que vous ayez le rôle « default role ». Cela ne se peut se faire qu'au lancement de jdev (Si le choix ne vous est pas proposé au lancement alors : properties -> role -> always prompt at startup).

Pour créer un diagramme de cas d'utilisation, créez un nouveau composant de type « use case diagrams » dans votre projet.

Pour créer un diagramme de classes, créez un nouveau composant de type « java class diagrams » dans votre projet. Les éléments à poser peuvent ne pas être disponible immédiatement dans la palette de composants. Dans ce cas, veillez à bien choisir les composants de type « java class » en haut du panneau d'outils.

f.3) Tableur

Le tableur est l'outil idéal pour réaliser un **diagramme de Gantt**. Open Office tout comme Excel vous proposent des outils pour réaliser de tels diagrammes de manière très professionnelle. Cependant, cela peut nécessiter un temps trop long. Nous vous déconseillons donc d'utiliser ces fonctionnalités. Utilisez simplement le tableur comme un tableau dont on peut colorer/détourer/hacher certaines cases.

3. Applette ou Java Web Start ?

Les deux approches permettent de réaliser un programme Java hébergé par un site web et s'exécutant sur une navigateur Web. La différence entre les deux est très petite :

- une Applette est un programme dont la classe principal hérite de Japplet. L'avantage est que le programme s'exécute dans la page Web, au milieu du texte et dans une mise ne page HTML/CSS.
- un Java Web Start est un programme dont la classe principale hérite de Jframe. Pour l'exécuter, l'utilisateur devra cliquer sur un lien HTML. Le programme s'exécutera alors dans une fenêtre séparée.

Pour vous guidé dans la réalisation de ces programmes, notez la présence de vidéos dans le thème 9 de la page du cours. Parmi elles, vous en trouverez plusieurs vous décrivant les principes de développement et déploiement d'une Applette telles que :

- Ma première Applette Java avec jDeveloper 20'
- Debogger son Applette avec la console Java des navigateurs
- Les Applettes Java : http://cipcnet.insa-lyon.fr/moodle.195/file.php/2/transparents/Applettejava.pdf

4. Déploiement du projet sur un site web

- Une application java s'exécute dans une machine virtuelle. Cependant, dans le cadre du projet, nous voulons que le programme développé soit hébergé sur un site web. Hors, la machine virtuelle utilisée par le navigateur web qui est plus restreinte (en terme de librairies) et contraignante (pour des raisons de sécurité). Pour réduire ces contraintes et faciliter le déploiement de l'application, nous demandons que celle-ci soit rendue sous la forme d'une archive jar signée. Les sections suivantes vous aiderons dans cette démarche.

a) Créer un jar

Il est possible de concaténer tous les fichiers nécessaires à l'exécution d'un programme java dans un unique fichier exécutable appelé **archive Jar**. Sous Jdev, la création d'une telle archive tient en 2 étapes. Il faut d'abord créer un profil de déploiement, où l'on configure les paramètres de la création de l'archive voulue (Illustrations 1 à 4), puis il suffit d'actionner le profil de déploiement pour que jdev génère le fichier jar à l'emplacement voulu (Illustration 5).

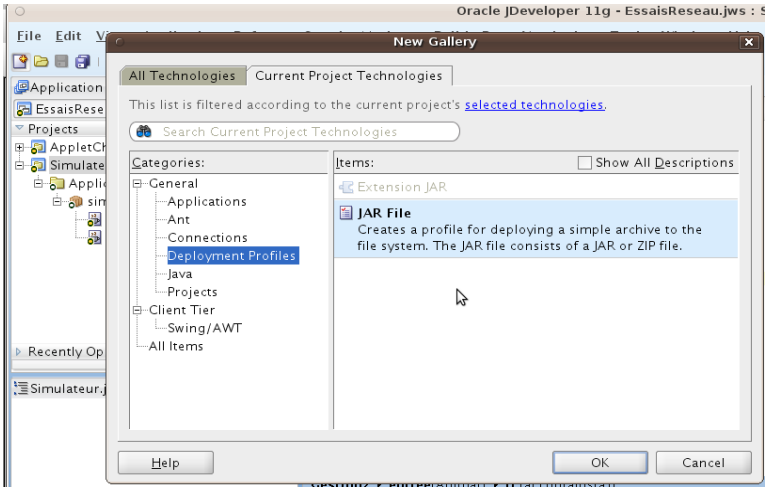


Illustration 1: Créer un nouveau profile Jar

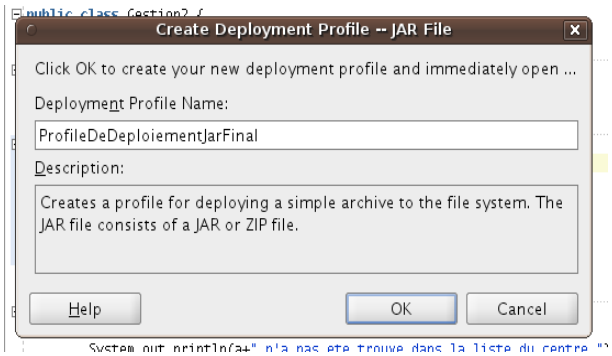


Illustration 2: Choix d'un nom de profile

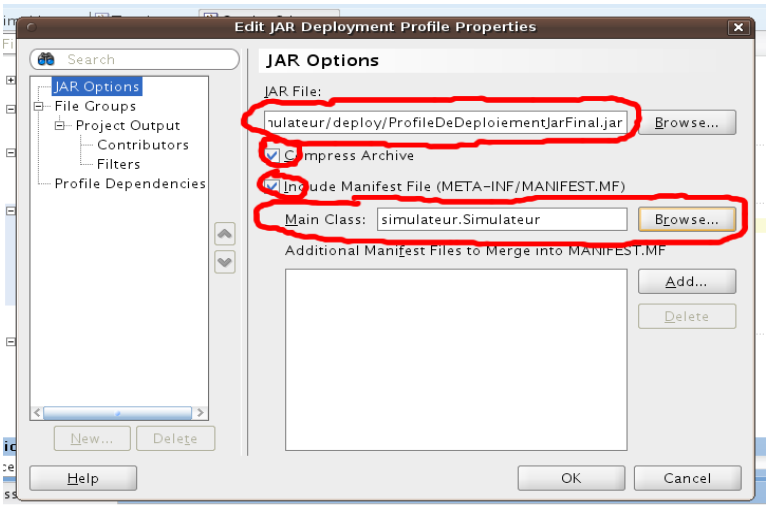


Illustration 3: Choix des options définissant ce profile

Les points importants sont : (i) nom et adresse du fichier à générer, (ii) compresser l'archive générée, (iii) inclure un manifest à l'archive et (iv) préciser le nom de la classe principale (héritant d'Applette).

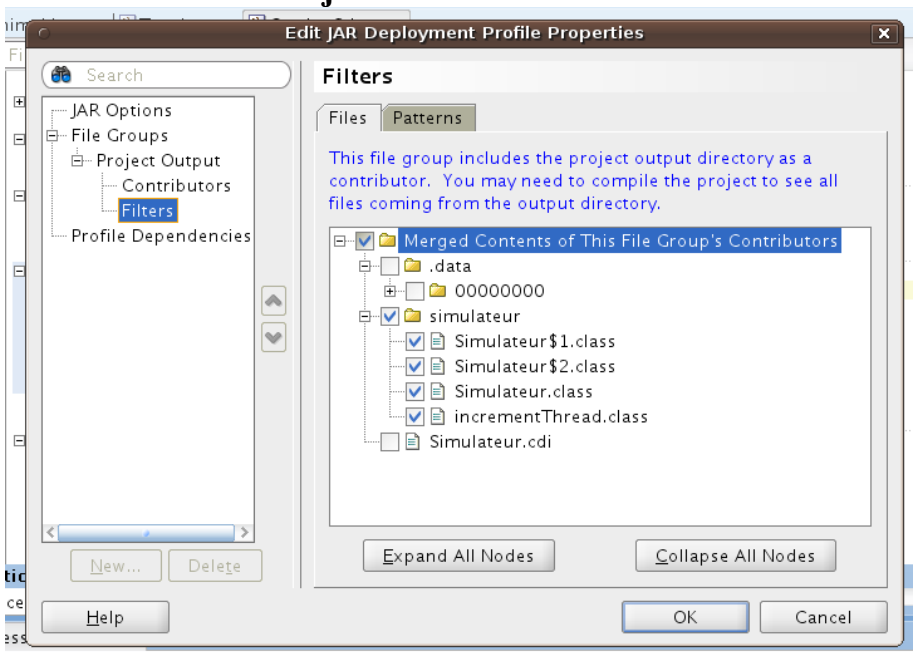


Illustration 4: Choix des fichiers compilés à inclure dans l'archive

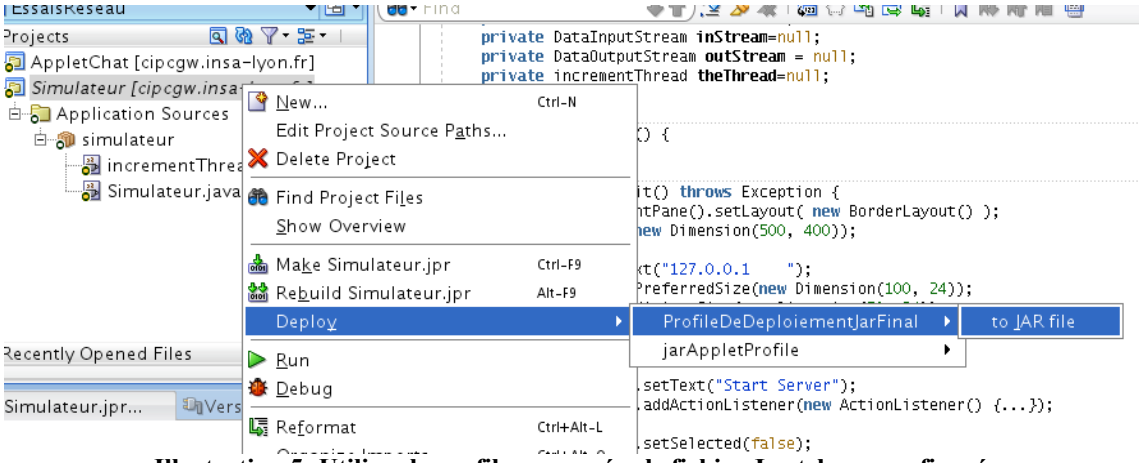


Illustration 5: Utiliser le profile pour créer le fichier Jar tel que configuré

b) Signer un jar

Pour exécuter une archive Jar avec des droits étendus (accès au réseau ou aux fichiers depuis le mode Applette), il est nécessaire de signer numériquement l'archive. Pour signer un Jar, il est donc nécessaire d'avoir une signature personnelle. Celle-ci est ensuite utilisée pour signer l'archive avant que l'Applette ne soit utilisable.

b.1) Création d'une signature numérique personnelle

En ligne de commande, tapez la commande suivante, en donnant un nom utilisable à votre clef (sans espace ni caractères spéciaux). Répondez aux questions posées et validez.

```
keytool -genkey -alias <nomDeVotreClef>
```

Cette génération n'est **à faire qu'une seule fois**. Lorsque votre clef aura été créée, vous pourrez y faire référence en utilisant le nom que vous lui avez donné.

Classiquement, cette clef est stockée dans le fichier *~./keystore*.

ATTENTION : Mettez un mot de passe robuste à votre clef car si vous vous la faite voler, alors un pirate pourrait signer des programmes malveillants avec votre identité.

b.2) Signature d'une archive

Pour signer une archive Jar, utilisez la commande suivante, en précisant le nom de l'Applette à signer et le nom d'identification de la clef à utiliser pour signer.

```
jarsigner -verbose monApplette.jar nomDeVotreClef
```

Une fois le mot de passe saisi, l'Applette est signée. Cette manipulation est a répéter avant chaque déploiement sur site web si l'Applette a été recompiler depuis sa dernière signature.

b.3) Utilisation d'une Applette signée

Lors de l'utilisation d'une Applette signée de cette manière, un message d'erreur peut apparaître pour signaler que l'organisme de certification n'est pas reconnu. C'est normal, car vous n'avez pas payé d'organisme externe pour générer votre signature. Mais vous pouvez forcer le lancement de votre Applette et celle-ci aura toutes les autorisations nécessaires pour effectuer des accès fichier ou des accès réseau.

c) Déployer une Applette sur un site Web

Le déploiement d'une Applette encapsulée dans une archive JAR se fait très facilement par une balise HTML :

```
<APPLET    CODE="<accès à la classe de type Applette>"
           ARCHIVE="<Archives JAR (l'Applette et ses dépendances)>"
           HEIGHT="400"
           WIDTH="500" >
     Texte, image ou code HMTL qui ne sera affiché que si le
     navigateur ne supporte pas la balise .
</APPLET>
```

Cette balise prend en paramètre la liste des archives JAR contenant le code nécessaire à l'exécution du programme, ainsi que le point d'entrée du programme. Les dimensions sont celles qui seront octroyées à l'Applette. Ce ne sont pas nécessairement celles prévues lors de son développement.

IMPORTANT : si l'archive JAR ne se trouve pas dans le même dossier que la la page HTML, alors le chemin d'accès doit être précisé dans l'attribut ARCHIVE.

Exemple de balise HTML permettant d'afficher l'Applette « Test.java », présente dans le package « programmes » et qui est archivée dans « maPremièreApplette.jar » :

```
<APPLET    CODE="programmes.Test"
           ARCHIVE="maPremièreApplette.jar"
           HEIGHT="400"
           WIDTH="500" ALIGN="bottom">
     This browser does not support Applettes.
</APPLET>
```

d) Solution alternative : application de type Java Web Start

L'avantage d'une application JWS sur une Applette est qu'elle peut être créée à partir d'une application graphique standard créée avec jDev. Le même code peut donc s'exécuter en mode 'local' aussi bien qu'en mode 'Internet'. À la différence d'une Applette, une application JWS ne s'intègre pas dans une page web, mais ouvre une nouvelle fenêtre pour se lancer. En revanche, les restrictions de ce type d'application sont les mêmes que celles des Applettes. En particulier, il est nécessaire de signer l'application pour que celle-ci puisse avoir accès au réseau et aux fichiers locaux.

Une démonstration vidéo est disponible sur la page du cours.