La programmation - 1/2

Je souhaite dans cet article essayer d'expliquer au mieux ce qu'il y a de pertinent dans les cours d'informatique que j'ai suivi l'année dernière sur le langage JAVA. Ces cours sont en anglais, et j'ai toutefois laissé certains termes, expliqués en détail, dans cette langue. J'espère que cet article n'est pas trop technique.

Les cours que j'ai étudié traitent de la programmation en langage Java. Le langage Java est un langage orienté objet. C'est à dire que c'est un langage qui utilise une autre technique et qui contient ce que l'on appelle des "classes" et des "méthodes". Je pense que l'intérêt majeur de cette technique est de permettre de faciliter d'avantage la réutilisation de portions de code : on a dès lors moins de code à écrire. C'est une technique qui permet d'englober dans une structure des fonctions et des procédures auxquelles on donne le nom généraliste de méthodes, et de pouvoir la réutiliser. Dans les langages modulaires cette technique n'existe pas, on peut tout au plus englober dans une procédure d'autres procédures.

Un langage fait pour la réutilisation du code

Le langage Java utilise des stratégies pour rendre les classes indépendantes les unes des autres. Plus les classes sont indépendantes entre-elles (ce que l'on appelle "low-coupling"), plus elles peuvent être réutilisées dans d'autres contextes. Quand on décide des interactions entre classes, on établit des dépendances. Pour créer le moins possible de dépendance et pouvoir effectuer des changements en affectant le moins possibles de classes, il existe notamment une loi, la loi de Demeter.

L'utilisation de certains modules que l'on appelle "interface" permet aussi de rendre les classes d'avantage indépendantes entre-elles dans un "strong-typed language" comme Java. Dans un langage qui n'est pas strong-typed, cette précaution est inutile. Un langage "strong-typed" est un langage où toutes les variables sont déclarées permettant de réduire les erreurs engendrées accidentellement.

Il y a un facteur qui facilite la réutilisation du code, c'est un facteur qui s'appelle "cohesion". En effet plus les méthodes englobées dans une classe ont des points communs (et donc plus il y a un facteur "cohesion" important), plus il est facile de réutiliser la classe.

Les ensembles de classes qui sont réutilisées dans d'autres contexte sont appelés des "Javabeans". On peut créer des "javabeans" en tant que tels ou les dériver d'un programme tout fait.

Faire un programme

Dans le cas d'un programme complexe demander par un client à une équipe de programmeurs, il y a normalement 4 activités dans sa création :

1-Analysis
2-Design
3-Implementation
4-Testing

Auxquelles il faut ajouter 2 autres étapes d'un autre type qu'il faut situer entre l'activité analysis et design :

A-Project management
B-Quality management

La programmation - 2/2

Il est important avant que j'entre dans l'explication de ces différentes étapes, que je précise que dans le langage Java, on utilise des diagrammes pour se représenter les choses avant de passer immédiatement au code. Ces diagrammes sont réalisés en général dans un standard qui s'appelle l'UML, et il y en a 6 variétés. Ils sont réalisés à l'aide d'un autre logiciel (que l'on appelle "CASE tool") qui vient s'ajouter au logiciel de programmation (moi c'est Jbuilder 4 que j'utilise).
Durant la première activité, les programmeurs analyse le "Statement of requirements", un texte que le client leur a préparé afin d'exposer les besoins du programme, et réalisent un "Negotiated statement of requirements", texte basé dessus et qui est plus conforme pour les developpeurs. C'est toujours dès la 1ère activité que la plupart des modèles en UML sont réalisés à partir de ce dernier document, mais durant les activités de 1 à 4, ils seront transformés et peaufinés, on créer au fur et à mesures de nouvelles versions de ces modèles. C'est à partir de l'activité 2, que l'on commence à écrire aussi le code final.
L'activité 4, "Testing" consiste à tester le code.

Durant l'activité "Project Management", on estime à partir de certaines formules, la durée du projet et la taille de l'équipe qui doit le mener à bien. C'est aussi durant cette activité que l'on alloue à chacun des membres, différentes activités.
L'activité "Quality Management" est une activité durant laquelle on établit des points de contrôle à divers moments de la création d'un programme et ainsi on fournit au client un moyen de s'assurer de sa qualité. On mesure les facteurs de qualités, dont certains peuvent être mesurés directement à partir du code ou de diagrammes en UML, et d'autres sont subjectifs. Ce sont des personnes accrédités, externes au projet qui font ce contrôle.

L'interface utilisateur

Dans un programme, en général on sépare l'interface utilisateur du reste du programme (que l'on appelle "Domain model"). On développe l'interface utilisateur en même temps que le "domain model" car ces 2 domaines sont interdépendents. Il y a 2 phases dans la création de l'interface utilisateur :

1. User need analysis.
2. Designing a user interface

Durant la phase 1, on analyse les caractéristiques de l'utilisateur potentiel du programme, des taches qu'il devra effectuer et de l'environnement externe dans lequel sera installé le programme. Cela permet de mieux cibler les caractéristiques de l'interface.
Il faut savoir qu'il y a beaucoup de caractéristiques sur une interface utilisateur (par exemple l'agencement des éléments qu'elle contient, l'apparence de ces éléments, leurs réactions au clic de la souris), et que la conception de cette interface est une science. Il y a 9 principes ("design principles") importants qui ont été identifiés dans la réalisation d'une interface.
Durant la phase 2, on créer l'interface utilisateur à partir des recommandations du "Negociated state of requirement" décrit plus haut. Pendant la création de l'interface, il faut justifier les choix nécessaires en créant un "Design Rational". Dans ce document on explique en quoi les différents aspects de l'interface sont en conformité avec chacun des "design principles".