INF2 – PROJET DE GÉNIE LOGICIEL 2010-2011

EMMANUEL BRUNO

1. Objectifs et organisation

1.1. **Objectifs.** Ce projet a pour objectif pratique de réaliser un outil simple d'autoformation à l'algorithmique et à la programmation de base. Cet outil est destiné à deux catégories d'utilisateurs : les enseignants qui pourront ajouter des algorithmes (une description et le code source) à la base existante et suivre les progrès des étudiants, les étudiants qui pourront consulter les algorithmes mais aussi tester leur propre solution.

L'objectif pédagogique principal est la mise en pratique des concepts étudiés en cours et appliqués en TP dans le cadre d'un projet complet :

- Analyse du problème et conception d'une solution simple modélisée avec un diagramme de classe UML.
- Implantation de la solution en Java en utilisant :
  - Les concepts de base fournis par java (Interfaces, classes abstraites, héritage, réutilisation, la documentation avec javadoc, ...)
  - Les fonctionnalités des paquetages classiques (interface graphique avec Swing, connexion via JDBC au SGBDR Postgresql, chargement et exécution dynamique de classe, ...).
  - Les extension classiques (les tests unitaires avec Junit, la gestion des logs avec log4j ou le jdk ...)

  Ce projet est composé de deux parties. La première comprend la gestion classique de données en Java (Création et consultation des algorithmes dans un SGBDR via une IHM). La seconde est dédiée à la vérification de l'implantation d'un algorithme proposée par un étudiant par rapport à une solution de référence.

1.2. **Organisation du projet.** Cette section décrit les deux parties du projet. La première est une application directe du cours et la seconde demande un peu plus de recherche personnelle.

2. Gestion des données

La première partie du travail consiste à fournir un outil permettant la gestion des algorithmes (une liste d'algorithmes est disponible ici :`http://zanotti.univ-tln.fr/ALGORITHMIQUE/`).

Un algorithme est décrit par un code à trois champs : le domaine (NUM pour numérique, GRA pour graphe, DON pour traitement de données,...), le type (ITR pour itératif, REC pour recursif), et le niveau de difficulté (de 0 à 3). Pour chaque algorithme on disposera au moins de son nom, d'une description en HTML et d'un pseudo-code. On pourra aussi avoir le programme en langage C correspondant.

Les utilisateurs de cette application sont de deux types : Les enseignants et les étudiants. Les enseignants ajoutent des algorithmes ou les modifient. Les étudiants consultent les algorithmes. La formation des étudiants est découpée en étapes (L1,L2,L3,M1,M2), et chaque étudiant appartient à une promotion (une étape et

Date: décembre 2010.

une année par exemple M1-2010). Pour chaque étape, il existe trois ensembles d'algorithmes : obligatoires, conseillés, facultatifs. Chaque étudiant devra pouvoir indiquer les algorithmes qu'il a étudié, les enseignants pourront consulter l'état d'avancement des étudiants.

- Proposer un diagramme de classe UML qui représente cette application.
- Décider quelle partie sera gérée dans la base de données et quelle partie par du code Java.
- Proposer un schéma relationnel pour la base de données.
- Définir les interfaces, et les classes pour l'application Java.
- Définir des tests unitaire à mettre en place.
- Proposer une organisation pour l'IHM de l'application.

3. Evaluation des réponses

Cette partie se propose de permettre à l'étudiant de tester une implantation de l'algorithme directement dans l'application. Pour cela, on associe à chaque algorithme, en plus du code en langage C, le prototype des fonctions qui composent l'implantation et plusieurs solutions écritent dans un langage simple : java−−.

3.1. **Prototypes Java−−.** La définition d'un algorithme A proposée par un enseignant est complétée par :

(1) une liste de prototypes de fonctions de la forme suivante :

$$< type > nom(< type > nom, ...)$$

où nom est un identificateur et $< type >$ est un type primitif Java ou un tableau. Cela définit donc l'interface IA de l'algorithme A. L'une des fonctions doit être désignée comme le point d'entrée.

(2) une liste de tests composés chacun d'un couple (liste de valeurs correspondants aux paramètres de la fonction d'entrée, résultat attendu de la fonction).

3.2. **Implantations Java−−.** Chaque utilisateur pourra proposer une implantation Java−− d'un algorithme A. Une implantation Java−− d'un algorithme est composée du code java pour les déclarations globales (du texte contenant des déclarations et affectations de variables) et le corps de chacune des fonctions de l'interface IA. Il est donc immédiat de créer le code source d'une classe Java A qui implante l'interface IA. Grâce aux mécanismes d'édition de lien dynamique de Java, il possible de compiler et d'instancier A pendant l'exécution. Finalement, il est possible d'invoquer la méthode déclarée comme le point d'entrée avec les paramètres de test pour vérifier que le résultat est correct.

Département d'Informatique - UFR de Sciences et Techniques - USTV
E-mail address: bruno@univ-tln.fr