Travaux pratiques combinés, OMGL et SGBD

Premières séances (7h): défrichage.

1h: Installation et brainstorming

exposé du principe des TP combinés, choix du sujet.
Brainstorming animé par l'enseignant pour lister grossièrement les fonctionnalités attendues. Outil: un google doc tout le monde peut écrire dedans. Un étudiant est désigné pour nettoyer le document et tout retranscrire dans le wiki de la forge.

1h: Maquettage

Chaque étudiant propose une ou plusieurs maquettes à l'aide de l'outil mockingbird.com. L'enseignant passe dans les rangs, relève les points intéressants, difficiles, les problèmes de conception de l'IHM, questionne le processus imaginé par l'étudiant.
Commentaires (positifs) de l'enseignant à voix haute, et conclusion avec tout le monde:
- inspiration de logiciels existants,
- influence du voisinage,
- focus sur différentes fonctionnalités de l'application
- une seule maquette, surchargée, pas de notion de "procédure"

Toutes les maquettes sont exportées en png et ajoutées comme fichiers sur la forge.

2h: Première analyse, les cas d'utilisation.

Par groupe de 2.
L'enseignant comit un projet VP avec un diagramme de cas d'utilisation vide, et assigne à chaque groupe d'étudiant un "thème" de cas d'utilisation:
" en tant que <acteur>, je veux ..."
Chaque groupe d'étudiants
- crée et détaille une demande sur la forge.
- checkout le projet dans VP, et travaille sur le thème de cas d'utilisation assigné.
  -> les acteurs, le système, les CU, si possible des specialize, include, extends...
- focus sur le flow of events d'au moins un cas d'utilisation du thème
  - description d'au moins trois procédures, dont au moins deux procédures d'erreur
  - l'enseignant passe dans les rangs, corrige et discute. On peut facilement mettre à jour un désaccord entre les membres de l'équipe et montrer que les procédures sont imprécises. On impose l'insertion d'images de maquettes dans la colonne "procédure".

Phase de comit et de mise à jour général,
Comportement attendu: on obtient un bordel monstre, tout le monde "nettoie un peu" dans son coin, ça crée des conflits à tout va...
On choisit une bonne méthodologie: un seul étudiant nettoie le diagramme.
Les autres étudiants terminent la description des demandes sur la forge, et les ferment.

Exemple groupe projet mediacenter audio:
- Gérer l'écoute
- Récupérer les paroles
- naviguer
- rechercher/filter
- gérer les playlist

Exemple groupe projet mediacenter video:
- gérer la lecture
- réorganiser la bibliothèque et renommer les fichiers sur le disque
- naviguer dans la bibliothèque de videos
- ajouter des sous-titres

- rechercher/filter
- obtenir info sur le film

2h: mini sprint en petit groupes: faisabilité et preuve de concept

composition et objectifs imposé par l'enseignant.
Objectif: fouiller le web pour évaluer la faisabilité du projet, preuves de concept en java si possible.
Résultat: à la fin de la séance, chaque groupe présente ses trouvailles pendant 5~10 min. L'enseignant peut apporter des précisions si besoin (pour la notion de service web par exemple)

Exemple groupe projet mediacenter audio:
- 2 étud: comment lire un mp3 en java ? ----> javafx
- 2 étud: c'est quoi les Tags des fichiers mp3 ? comment y accéder en java ?
- 2 étud: peut-on récupérer les paroles et les afficher (a-t-on le droit) ?
- 4 étud: c'est quoi MusicBrainz ? qu'est-ce qu'on peut obtenir comme information ? Existe-t-il des solutions alternatives ? Spécialisation imposée dans ce groupe après un certain temps:
  - 2 étud: c'est quoi un service web ? comment ça marche ?
  - 1 étud: c'est quoi un fingerprint ? à quoi ça sert ?
  - 1 étud: peut-on accéder à wikipedia par programmation ?

Exemple groupe projet mediacenter video:
- 4 étud: comment afficher une vidéo en java, problème de codec, l'enseignant leur impose l'étude de VLCJ au bout d'un certain temps
- 4 étud: imdb, allociné, comment obtenir des informations sur les films et acteurs par programmation ? Spécialisation spontanée dans ce groupe après un certain temps:
  - 2 étud: découverte de l'existance des services web RESTful, formats JSON, XML
  - 1 étud: imdb
  - 1 étud: themoviedbapi
- 3 étud: Comment récupérer automatiquement les sous-titres ? Spécialisation spontanée dans ce groupe après un certain temps:
  - 2 étud: découverte de l'existence des services web XML-RPC, ...
  - 1 étud: mais comment diable peut-on savoir que ce fichier là correspond à ce film là ? compréhension du hachage utilisé par opensubtitles
- 1 étud: peut-on accéder à wikipedia par programmation ?

1h: Choix du modèle de l'application: les entités, ...

Choix ensemble, premier diagramme de classe rapide au tableau
Chaque étudiant/groupe, crée un diagramme d'entité-relation dans VP, niveau "architectural". On s'en sert pour générer automatiquement un diagramme de classes
On peut modifier l'un ou l'autre et utiliser les fonctions de synchronisation.
Sur une entité, on réfléchit aux colonnes: clé primaire ? unicité ? possibilité de nullité ? indexation ?...

Préparation des prochaines séances: travail de(s) enseignant(s)

1. nettoyage du diagramme de cas d'utilisation, sans amélioration, seulement fusionner éventuellement les acteurs, les systèmes, ...
2. création d'un projet java, développement d'une correction du modèle
 - dans le package fr.unice.iutinfo.s3d.{$nom_application}.model
 - c'est un modèle simplifié, il peut manquer des attributs, les étudiants amélioreront ce modèle plus tard si besoin.
3. générer automatiquement un diagramme de classe dans le projet VP et le nettoier.
4. créer le diagramme entité-relation correspondant (simplifié), avec proposition clé primaire, unicité, ...
5. créer une ébauche du package d'interfaces controlleurs:
fr.unice.iutinfo.s3d.{$nom_application}.controller.GestionnaireArtiste
fr.unice.iutinfo.s3d.{$nom_application}.controller.GestionnaireDisque
6. créer une ébauche du package de classes controlleurs
fr.unice.iutinfo.s3d.{$nom_application}.controller.impl.GestionnaireArtisteImpl
fr.unice.iutinfo.s3d.{$nom_application}.controller.impl.GestionnaireDisqueImpl
seulement public class GestionnaireXXImpl implements GestionnaireXX,
 - laisser les étudiants écrire les autres interfaces et remplir les méthodes (ce sera un peu du travail à la chaîne)
7. récupérer les jar (éventuellement javadoc) nécessaires et bien définir le classpath du projet, écrire quelques petites classes exécutables de démonstration de la faisabilité, preuve de concept

commun aux deux projets:
fr.unice.iutinfo.s3d.{$nom_application}.demo.SQLiteDemo
-> méthode qui supprime la table ARTISTE
-> méthode qui crée la table ARTISTE
-> méthode qui insère une ligne dans la table ARTISTE
-> méthode qui lit et affiche les lignes de la table ARTISTE
-> méthode main qui exécute séquentiellement ces méthodes

projet mediaplayer video:
fr.unice.iutinfo.s3d.{$nom_application}.demo.OpenSubtitlesAPIDemo
fr.unice.iutinfo.s3d.{$nom_application}.demo.TheMovieDBDemo
fr.unice.iutinfo.s3d.{$nom_application}.demo.VLCJDemo

projet mediaplayer audio:
fr.unice.iutinfo.s3d.{$nom_application}.demo.JibikiDemo
fr.unice.iutinfo.s3d.{$nom_application}.demo.GwtWikiDemo
fr.unice.iutinfo.s3d.{$nom_application}.demo.MusicBrainzDemo
Attention, cette dernière démo était vide, mais contenait en commentaire des instructions pour la brute de la classe pour développer un wrapper de l'API MusicBrainz (travail quasiment réalisé en 7h, je l'encouragerai à publier son travail sur google code et le valoriser sur le web). J'ai fait ce choix car cette API venait d'être mise à jour et les différentes API java existantes n'étaient pas fonctionnelles.

8. comit le projet sur la forge

A partir de ce point, nous différencions les séances SGBD et les séances OMGL.

OMGL: Séance 2, 4h, Sprint "rush"

Travail préliminaire: installer subversive sur leur eclipe, checkout le projet, exécuter les classes et corriger si besoin (par exemple la classe VLCJ nécessite d'écrire des chemins vers la librairie native, un VLC 32bit n'est compatible qu'avec une JVM 32bit etc...)

(*à faire*: écrire un petit doc pour décrire la procédure et les différentes erreurs que l'on peut rencontrer)

3h. Sprint en binôme

Par groupe de deux.
Objectif: peu importe leur niveau, peu importe leur travail, maintenir les étudiants sous pression pendant 3h.
L'enseignant assigne à chaque groupe un objectif à atteindre:
- on doit pouvoir justifier que l'objectif est fondamental pour l'une ou l'autre des fonctionnalités importantes de l'application finale.
- on doit adapter la difficulté de l'objectif au niveau estimé du groupe
- on doit anticiper quelques ponts évidents entre les travaux des groupes pour la 4e heure.

Les étudiants doivent créer la demande sur la forge, et sprinter pour atteindre cet objectif.
- si ils demande comment remplir la demande, ce qu'ils devraient mettre dans chaque champs: on leur dit: "on s'en fout, on rush ! on fera ça plus tard. dépêchez vous de coder !
- On s'en fout complètement que le code soit moche, mal conçu, ou couvre tous les cas.
- Travail en profondeur d'abord. Si l'objectif est atteint avant les 3h, l'enseignant leur impose une extension, amélioration, ...

Exemple projet mediaplayer video
- une fenêtre qui lit la video avec vlcj, puis on ajoute , puis on ajoute un contrôle, on le rend foncitonnel, puis on ajoute un nouveau contrôle etc. (play, pause avec jbutton, volume avec jslider, mute avec jtogglebutton, contrôle et visualisation de la position dans le fichier mp3 avec jslider...)
- une fenêtre qui lit la video, et on ajoute le contrôle play/pause avec la barre d'espace.
- récupération des informations d'une vidéo via themoviedb: le titre, le chemin, la popularité et le nom du réalisateur d'un film
- visualisation et contrôle de la position dans la vidéo avec un jslider
- une JFrame avec une option de menu qui ouvre un JFileChooser grâce auquel on peut choisir un répertoire. On recherche alors toutes les video du repertoire et des sous-repertoires,
- ajout d'un menu pour telecharger automatiquement les sous-titres français de la vidéo que l'on lit, et les ajouter à VLCJ. Ensuite, creation d'une fenêtre pour permettre de choisir la langue des sous-titres à télécharger

Exemple projet mediaplayer audio
- une fenêtre qui lit l'audio avec javafx, puis on ajoute un contrôle, on le rend foncitonnel, puis on ajoute un nouveau contrôle etc. (play, pause avec jbutton, volume avec jslider, mute avec jtogglebutton, contrôle et visualisation de la position dans le fichier mp3 avec jslider...)
- une JFame avec une option de menu qui ouvre un JFileChooser grâce auquel on peut choisir un répertoire.
- à partir du chemin d'un répertoire, lister tous les fichier mp3 d'un dossier (et de ses sous-dossiers) / pour chaque fichier mp3, récupérer les tags id3v2 et instancier des objets du modèle.
- à partir d'objets du modèle, affichage d'un tableau avec les colonnes titre, duree, popularité, artiste, album. Faire en sorte que ce tableau se trie lorsque qu'on clique sur un entête de colonne
- creation d'un wrapper pour l'API web MusicBrainz
- affichage d'un article wikipedia dans une JFrame, avec un slider pour défiler / ouverture d'une page internet dans le navigateur par défaut / à partir du nom de l'artiste, recherche de l'URL de la page wikipedia qui décrit cet artiste.

Dernière heure

chaque groupe a 2min pour s'adresser à la classe entière et expliquer ce qu'il a réussi à faire.
Ensuite, l'enseignant leur demande quels ponts évidents ils imaginent.

L'enseignant peut aider à trouver les ponts. Les étudiants créent les demandes, et essaient de faire le plus possible d'intégration avant la fin de l'heure. Une demande et une classe par fusion des travaux des groupes.

SGBD: Séance 2, 3h

1. Les étudiants dupliquent les packages ~.model en ~.<nom>.model, idem pour controller et controller.impl Attention aux imports, qui sont souvent foireux.
2. Ils doivent d'abord compléter la déclaration des interfaces, du package controller
3. Ils doivent ensuite s'inspirer de la demo JDBC SQLite pour les implémentations concrètes.

Objectif 1: on souhaite dans chaque gestionnaire:
une méthode pour créer la table
une méthode pour supprimer la table
une méthode pour lister tous les objets
une méthode pour trouver un objet à partir de son identifiant
une méthode pour mettre à jour un objet
une méthode pour supprimer un objet

Objectif 2 (pas atteint du tout en 3h) on souhaite avoir une classe executable ~.<nom>.Main, qui permette:
de supprimer toute la base de données
de créer toute la base de données
de peupler la base de données avec un jeu d'objets de test.

A savoir !!! dans le diagramme entité-relation, clique droit n'importe où -> utilities... -> generate SQL

J'ai pris mon temps avec ceux qui ont des lacunes
Pour ceux qui avançaient un peu plus vite, j'ai proposé de:
- réflechir à comment factoriser le code, en particulier l'intérêt:
  - d'une interface générique ~.<nom>.controller.Gestionnaire<T,ID extends Serializable>
  - d'une classe abstraite ~.<nom>.controller.impl.GestionnaireImpl<T,ID extends Serializable>
- générer la javadoc

OMGL: Séance 3, 4h, Grand nettoyage

1h: présentation de la solution des gestionnaires avec généricité, et pourquoi on a quand même besoin des gestionnaires ? parce que ils peuvent avoir des méthodes plus compliquées que les quatres méthodes de base.
exemple pour la pagination: List<Film> findByName(String query, int firstResult, int nbResult)
exemple avec jointures: List<Film> findFilmsTheyPlayedTogether(Actor actor1, Actor actor2)
Choix de quelques méthodes dont on a besoin, selon les sprints qui ont été fait. On ne les implémente pas

3h: grand nettoyage du résultat des sprints, choix d'une architecture du projet, les packages, les vues, les controlleurs,
écriture des tests unitaires

... à suivre

SGBD: Séance 3, 3h, Méthodes complexes

Implémentation des méthodes complexes de gestion des données choisies, plus d'autres.
validation des résultats dans un instanciation de la classe tableau

OMGL: Séance 4, 4h, Dernier sprint

idem séance 2, avec de nouveaux objectifs à définir. (par exemple, utilisation des images pour la navigation : pochettes d'album / poster de films)

SGBD: Séance 4, 3h, Chargement aggressif et cascades

Objectif: les faire coder des trucs qui vont générer des erreurs, que ce soit évident pour qu'ils comprennent pourquoi ça fait des erreurs, réfléchir et coder des solutions
- la persistance d'une ligne avec null dans une colonne not nullable
 -> on doit persister l'objet référencé avant l'objet
 -> problème des cycles
- les attributs collection dans les bases de données
 -> doit-on imposer le chargement d'un attribut liste de manière aggressive ? comment faire autrement ?

Colle pour les furieux: proposer une solution générique et simple pour préciser lorsqu'il doit y avoir cascade. (utilisation d'annotations, début de réflexion sur l'implémentation de JPA)

OMGL: Séance 5, 4h, séance notation de groupe.

Séance libre. J'observe et je note le fonctionnement du groupe, l'investissement de chacun, les discussions. Je peux donner quelques conseils ou corriger quelques bugs bien entendu.

objectif 1:
parfaire l'application, améliorer ce qu'on a déjà et ne surtout pas chercher à avoir plus de fonctionnalités

objectif 2:
à partir du diagramme des cas d'utilisation:
- vérifier que le workflow est correct pour l'application, l'améliorer si besoin,
- vérifier que les tests fonctionnels passent tous, que les résultats attendus sont corrects.
- proposer chacun au moins deux tests fonctionnels qui représentent des améliorations possibles de l'application (correction de bugs par exemple...).

SGBD: Séance 5, 3h, Solutions, Hibernate, JPA

Solution des exercices précédents avec l'utilisation d'annotation
présentation de Hibernate et JPA

Ouverture sur les développements bonus

Il n'y a pas de projet à rendre pour ce cours.
Par contre, tout développement ultérieur peut faire l'objet d'une évaluation très rapide, et peut apporter des bonus sur la note finale dans une limite de 3pt bonus.
La date limite est: une semaine pile poil avant l'examen écrit
Pour ne pas être ignoré, chaque développement doit faire l'objet:
- d'une demande dans la forge, bien documentée,
- d'un comit du VP, avec un commentaire précis de ce qui a changé,
- d'un comit du projet java, avec un commentaire précis de ce qui a changé.