Modélisation et programmation objet : Projet encadré Java

Carl Esswein

Dernière mise à jour : mars 2016.

Table des matières

Les objectifs de ce projet sont nombreux :

- Mise en œuvre sur un cas concret de la modélisation orientée objet (UML),
- Mise en œuvre pratique des concepts de programmation orientée objet et du langage java,
- Approfondissement de l'usage de l'API standard Java SE (IHM, gestion des dates et du temps, écriture de sockets, gestion du multithread, entrée/sortie fichier, sérialisation, *etc.*)
- Mise en œuvre sur un cas concret du *pattern* MVC.
- Travail sur un projet (presque) complet : de la conception à la livraison.
- Mise à l'épreuve de votre capacité de travail en autonomie.

Le sujet est construit pour être un support de progression pour le plus grand nombre : les étudiants débutants en java ainsi que ceux un peu plus expérimentés. Ainsi, beaucoup de fonctionnalités seront décrites sous la forme de plusieurs versions successives. Je vous invite à prendre connaissance de toutes ces version puis à construire votre solution *crescendo*. Chacun ira jusqu'où il pourra. Par ailleurs, des options permettront aux gourmands de se sustenter.
Dans tous les cas, les séances de travail encadré ne seront pas suffisantes pour vous permettre de réaliser totalement le projet. C'est normal. Il est recommandé un minimm de 1h de travail personnel pour chaque séance encadrée de 2h.

1. Le sujet

1.1. Présentation

Nous allons construire petit à petit une **« Application de suivi des pointages d'horaires des employés d'une entreprise »**. Tous les employés de l'entreprise utilisent une pointeuse avec leur carte professionnelle (*Cf.* Fig. 1) : tous les matins pour signaler au système leur arrivée sur leur lieu de travail, et tous les soirs pour indiquer que leur journée de travail est terminée. L'entreprise n'est ouverte que du lundi au verndredi et on ne prend en compte ni les pauses café-clope ni la pause déjeuner. On se place dans les cas simplifié où il n'y a ni congés ni jours feriés.

L'application à concevoir et réaliser sera une application monoposte, dotée d'une IHM riche, permettant de suivre et superviser l'état des pointages des employés de l'entreprise. Elle a pour vocation à être utilisée par les managers et la direction de l'entreprise mais de manière centralisée (un seul déploiement de l'application, sur un poste unique).

L'entreprise est constituée de plusieurs *départements*. Les *employés* de l'entreprise appartiennent à un[1] département et un seul. Chaque département est supervisé par un[1] *manager*. Chaque employé est identifié par un *identifiant*, unique, stocké notamment dans sa carte professionnelle pour enregistrer ses *pointages*.

Chaque employé a une heure d'arrivée standard (unique, pour tous les jours de travail) et une heure de départ standard (unique, pour tous les jours de travail). Du lundi au vendredi, il ne doit pas arriver après son heure d'arrivée standard, et ne doit pas partir avant son heure de départ standard. Mais il y a une exception car les heures supplémentaires sont comptabilisées et sont récupérables. Par exemple, si un employé arrivé une demi-heure en avance le matin, il peut partir une demi-heure plus tôt le soir… ou conserver cette demi-heure pour arriver en retard ou partir plus tôt un autre jour.

Tous les managers appartiennent, en plus de leur département de rattachement (le département qu'ils dirigent), à un département « Management », dirigé par le *Boss*. Le Boss n'est pas un employé. Il ne pointe pas ses horaires. Les managers sont des employés et pointent comme tous les autres.

1.2. Pointeuse

Un module « pointeuse » remonte à l'application les pointages des employés (début de journée de travail, fin de journée de travail), chaque **pointage** contenant les informations suivantes : date (30/03/2015), heure du pointage (arrondie au ¼ d'heure le plus proche), identifiant de l'employé.
Chaque pointage est remonté en temps réel à l'application centrale par l'utilisation de sockets TCP. S'il est impossible de se connecter à celle-ci, les pointages sont stockés sur la pointeuse, puis envoyés de nouveau dès que la connexion redevient disponible.

Figure 1 — la pointeuse

[1] *« Zéro » possible de façon transitoire à l'initialisation, avant d'avoir remplie la structure de données de façon cohérente.*

On émulera ce module en réalisant une 2e application dont les besoins sont exprimés dans le pragraphe ci-dessous.

Emulateur de pointeuse : fonctionnalités

Il s'agira, à terme, d'une application monoposte, dotée d'une IHM riche, permettant de sélectionner un employé et de simuler un pointage sur le lecteur de cartes. Pour cela, l'application affichera la date et l'heure courante, ainsi que l'heure arrondie au quart d'heure le plus proche. L'utilisateur de l'application pourra sélectionner un employé (via nom-prénom, *e.g.* dans une liste déroulante) puis valider l'envoi.

Les informations relatives aux employés seront chargées par le réseau (via des sockets TCP) depuis l'application centrale. Idéalement, ces données seront synchronisées automatiquement. Dans un premier temps, la synchronisation peut être initiée par l'utilisateur.

Les données de l'émulateur de pointeuse seront sérialisés automatiquement à la fermeture de l'application, et désérialisés automatiquement au lancement de celle-ci.

Suggestion d'IHM

Vous pourrez vous inspirer de la maquette d'IHM ci-contre.

Il s'agit d'une simple suggestion. Vous pouvez faire des propositions alternatives dès lors qu'elles répondent aux besoins fonctionnels exprimés ci-dessus.

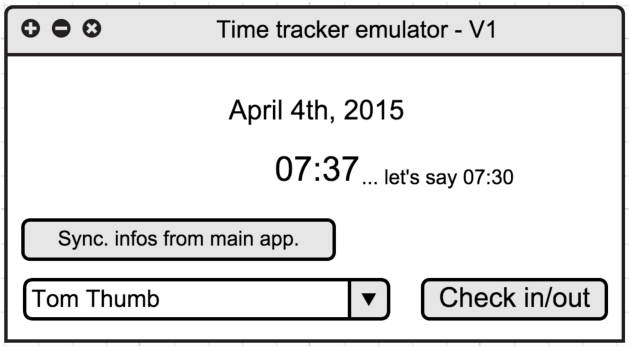

Figure 2 - Suggestion d'IHM (V1)

1.3.Application centrale

Aperçu des fonctionnalités

Vous trouverez ci-dessous un **aperçu** de la liste des fonctionnalités attendues pour l'application centrale. La liste définitive vous sera diffusée en temps utile, dans un document spécifique. Cet aperçu donne cependant une vision assez large des éléments attendus.

Fonctionnalités principales

1. Gestion des **input** de la pointeuse (récupération des données de pointage, via les sockets).
2. **Sauvegarde** des données par sérialisation (employés, départements, pointages, *etc.*)
3. **Consultation** des pointages enregistrés
   - 3.1. Tous (par exemple via une *JTable*)
   - 3.2. Etat en cours des présences
   - 3.3. Vues aggrégées par personne, par département et/ou par date/période (=filtrage)
4. Synchronisation des infos concernant les employés avec la pointeuse (id, nom, prénom).

Fonctionnalités secondaires

5. Gestion des **départements**
   - 5.1. Visualisation de la liste des départements
   - 5.2. Ajout, modification, suppression
6. Gestion des **employés**
   - 6.1. Visualisation de la liste des employés
   - 6.2. Ajout, modification, suppression
   - 6.3. Import CSV
   - 6.4. Export CSV
7. **Alertes** par email

   Chaque manager peut-être informé par email des « incidents » de pointage des employés de son département :
   - 7.1. Tout retard de plus de 30 minutes (absences aussi, donc)
   - 7.2. Tout départ anticipé de plus de 30 minutes
8. Gestion des **paramètres** (param IP & port module pointeuse, …, fréquences alertes, etc.)

9. **Reporting** par email

Chaque manager peut recevoir (9.1) par email un bilan des recueils de pointages de son département. De même, le Boss peut recevoir (9.2) par email un bilan des recueils de pointage pour l'entreprise.

9.1. Mensuel, pour chq dept : aggrégé du dept + aggrégé par pers + détail complet
9.2. Mensuel, aggrégé pout tte l'entreprise + aggrégés par dept

IHM – suggestions

Pour réaliser cette application, vous avez toute liberté pour proposer des IHM pertinentes. Pour autant, il paraît nécessaire d'avoir au moins les « zones » suivantes (gérées par exemple sous forme d'onglets) :

- Staff management (F6[2])
- Department management (F5)
- Global check in/out history (F3.1)
- How is your day? (F3.2)

Faites donc des propositions… et faites-les valider !

(N'oubliez pas qu'il convient de structurer vos codes en mettant en place le pattern MVC !)

1.4. Fonctionnalités optionnelles

Vous trouverez en annexe 1 une liste de fonctionnalités optionnelles, sans ordre de priorité ni de difficulté, que les plus efficaces d'entre vous peuvent souhaiter intégrer à leur projet. Il s'agit d'options qui ne sauraient remplacer le manque de fonctionnalités principales ou secondaires.

2. Organisation

2.1. Structuration

Le projet est un petit peu trop ample pour que vous commenciez à coder tout de suite (euphémisme…) ! Les premières séances seront grandement dédiées à l'analyse et la conception. Ne négligez pas cette première phase du projet.

Le projet est à découper en plusieurs lots. La liste des lots vous parviendra ultérieurement. Chaque lot pourra donner lieu à 0, 1 ou plusieurs livrables qui seront évalués.

2.2. Evaluation

Il est conseillé de travailler en petits groupes pour les parties d'analyse/modélisation et pour la prise en main de certains aspects techniques (threads, sockets, serialisation, *etc.*). Mais l'evaluation de ce projet est individuelle : chacun devra rendre individuellement les différents éléments demandés durant le projet.
Poser des questions ne sera jamais pénalisé (pas pris en compte dans l'évaluation), au contraire : plus de questions implique une meilleure compréhension des objectifs à atteindre.

2.3. Encadrement

Ce projet encadré n'est ni un TP ni un projet en temps libre. Vous serez très guidés durant les premières séances, et de moins en moins au fur et à mesure de l'avancée des projets ; c'est normal. Les premières séances étant dédiées à de la modélisation, les petites variations entre vos différents modèles à ce stade auront des conséquences de plus en plus grandes au fil de la construction de votre solution.

[2] Fait référence au numéro de fonctionnalité.

3. Consignes générales et conseils

- Le ***pattern MVC*** est **obligatoire** pour la structuration de vos modèles objet.
- La rédaction de commentaires (pertinents) et la génération de la javadoc sont **obligatoires**.
- Il est **conseillé** de mettre le code en anglais (noms de types, variables *etc.*).
- Il est, évidemment, **recommandé** de faire des sauvegardes régulières et de gérer un versionning mimimum.
- Regrouper les constantes (numériques, String etc.) dans des VRAIES constantes au début de vos classes. >> Réduire au max l'usage des chiffres et des «"» dans vos codes. Ca permet une meilleure maintenabilité.
- Posez des questions !!

Conclusion

Des documents annexes viendront compléter cette présentation générale du projet. Si vous décelez des incohérences, merci de m'en informer. Mais en cas de doute, c'est l'information la plus récente qui fait foi.

Bon courage, bons projets,

Annexes

4. Annexe 1 : options

Voici une liste – non exhaustive – de fonctionnalités optionnelles, sans ordre de priorité ni de difficulté, que les plus efficaces d'entre vous peuvent souhaiter intégrer à leur projet.

1. Dans les différentes vues, mettre en évidence les données qui sont **temporaorement incohérente** (par exemple afficher d'une autre couleur un employé sans affectation, ou un département sans manager…).

2. Créer un module de **régularisation manuelle** des incohérences (par exemple Paul Martin a oublié de pointer hier soir à 18h00.). Faire en sorte de pouvoir ajouter manuellement (via l'IHM de l'application principale) des pointages « artificiels ». Mettre en place la possibilité d'en ajouter plus massivement (file import).

3. Faire en sorte que chaque employé puisse avoir des heures de début et de fin de journée standards différentes selon le jour de la semaine.

4. Gérer les congés.

5. Permettre de **sérialiser** les paramètres dans un fichier de configuration.

6. Pouvoir gérer plusieurs pointeuses.