# Modélisation et programmation objet : Projet encadré Java

***Carl Esswein***

***V1.4.1-COVID***
*Dernière mise à jour : mai 2021.*

# Table des matières

# Préambule

Ce sujet a fait l'objet d'une adaptation pour l'enseignement partiellement à distance en période de crise sanitaire. La liste des fonctionnalités a notamment été allégée.
Les interactions et échanges liés à ce projet auront lieu principalement sur un canal dédié dans l'**équipe Teams** "Java DI3 2020-21" que vous connaissez déjà.

Les objectifs de ce projet sont plutôt 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 en passant par les tests *etc*.
- Mise à l'épreuve de votre capacité de travail en autonomie (particulièrement en ce moment !).

Le sujet est construit pour être un support de progression pour le plus grand nombre : les débutants en java ainsi que ceux un peu plus expérimentés. Ainsi, la plupart des fonctionnalités seront réalisables sous la forme de plusieurs versions successives. Je vous invite à travailler dans cet esprit et à construire votre solution *crescendo*. Chacun ira *in fine* jusqu'où il pourra. Par ailleurs, des options permettront aux gourmands de se sustenter.

# 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) : typiquement 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 vendredi et on ne prend en compte ni les pauses café ni la pause déjeuner. On se place dans le 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 la direction de l'entreprise, de manière centralisée (un seul déploiement de l'application, sur un poste unique).

L'*entreprise* est constituée de plusieurs *départements*, eux-mêmes contenant des *employés*. Un employé appartient à un département et un seul. Chaque employé est identifié par un *identifiant*, unique, stocké notamment dans sa carte professionnelle pour enregistrer ses *pointages*.

Chaque employé a un planning hebdomadaire fixe défini par une heure d'arrivée et une heure de départ pour chaque jour de la semaine (lundi-vendredi). *Par exemple Enzo travaille toute la semaine de 8h à 16h15 sauf le mardi où il travaille entre 9h45 et 17h.*
Un employé ne devrait théoriquement pas arriver au travail après (ni avant) son heure d'arrivée, et ne devrait pas en partir avant (ni après) son heure de départ. Pourtant, si cela arrive les heures supplémentaires sont comptabilisées (en plus ou en moins) et sont récupérables. Par exemple, si un employé arrivé une demi-heure en avance le mardi matin, il peut partir une demi-heure plus tôt le mardi soir… ou conserver cette demi-heure pour arriver en retard un autre jour. Mais finalement, peu importe : l'application fait simplement le *reporting* des heures de pointages... et le calcul du stock d'heures supplémentaires.

Figure 1 — la pointeuse

## 1.2.Pointeuse

Un module « pointeuse » (*Cf.* Fig. 1) 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 localement sur la pointeuse, puis envoyés de nouveau dès que la connexion redevient disponible.

**On émulera ce module « pointeuse »** en réalisant une 2e application dont les besoins sont exprimés dans le paragraphe ci-dessous.

### Emulateur de pointeuse : fonctionnalités

Il s'agira d'une application monoposte, dotée d'une IHM riche, permettant 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 (*i.e.* l'émulateur) pourra saisir l'identifiant d'un employé (ou sélectionner le nom de l'employé dans une liste déroulante) puis valider l'envoi (*bouton « Check in/out » dans la figure 2*).

Les éventuelles données de l'émulateur de pointeuse seront sérialisés automatiquement à la fermeture de l'application, et désérialisés automatiquement au lancement de celle-ci. On peut par exemple penser aux pointages qui n'ont pas encore pu être envoyés.

### Suggestion d'IHM

Vous pourrez vous inspirer de la maquette d'IHM ci-contre. Attention, elle est partielle : il y manque les éléments de configuration réseau – IP & port – pour communiquer avec l'application principale.

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

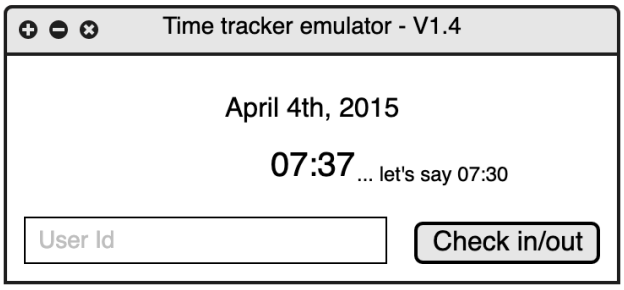


Figure 2 - Suggestion d'IHM (V1.4)

## 1.3.Application centrale

### Liste des fonctionnalités

Vous trouverez ci-dessous la liste des fonctionnalités attendues pour l'application centrale (et par induction, certaines de l'émulateur de pointeuse aussi).

F1. Gestion des **input** de la pointeuse (récupération des données de pointage, transfert via les sockets, enregistrement dans la bonne structure de donnée).
F2. **Sauvegarde** des données par sérialisation (employés, départements, pointages, pointages en attente d'envoi, *etc.*), et restauration par désérialisation au lancement des applis.
F3. **Consultation** des pointages enregistrés (par exemple via une `JTable` avec Swing ou une `TableView` avec JavaFX)
    F3.1. Tous les pointages enregistrés
    F3.2. Pointages de la journée en cours

A des fins de tests (pour vous et pour moi !), et a minima tant que la fonctionnalité F4 n'est pas implémentée en ce qui concerne la création des employés, le client vous demande de **créer une méthode stub**[1] chargée de construire une entreprise virtuelle avec quelques employés virtuels. Nous parlerons de la fonctionnalité F0.

[1] https://en.wikipedia.org/wiki/Method_stub

F4. Gestion des **employés** (CRUD[2]...), y compris gestion des plannings
  F4.1. Création-ajout d'un employé
  F4.2. Visualisation de la liste des employés
  F4.3. Visualisation détaillée d'un employé
  F4.4. Modification, suppression d'un employé
F5. Gestion des **paramètres** (param IP & port de l'application principale pour la pointeuse, *etc.*)

### 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 (F4)
- Check in/out history (F3)
- Paramètres (F5)

Il est fortement conseillé d'utiliser Swing pour la construction de vos IHM.

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

## 1.4.Compléments

Vous trouverez en **annexe 1** des informations de modélisation complémentaires que je ne mets pas ici pour alléger le document. Néanmoins ce sont des éléments importants à prendre en compte.

## 1.5.Fonctionnalités optionnelles

Vous trouverez en **annexe 2** 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 ! Je vous invite à en profiter pour découper le projet en plusieurs « lots », pourquoi pas en partant de la liste des fonctionnalités et en effectuant des regroupements si possible, ou pour des focus techniques (gestion I/O par exemple pour la sérialisation, ou bien encore réseau/sockets TCP). Pensez à envisager plusieurs versions successives quand c'est pertinent.

## 2.2.Evaluation

- L'évaluation de ce projet se fera **en groupe de 3 ou 4 personnes** ; chaque groupe devra rendre sa version personnelle[3] des différents éléments demandés durant le projet.
  Cependant, il est conseillé d'échanger entre vous de façon plus large, dans la mesure du possible, pour les parties d'analyse/modélisation et pour la prise en main de certains aspects techniques (threads, sockets, serialisation, *etc.*).
- **Poser des questions** ne sera jamais pénalisé, au contraire : plus de questions implique une meilleure compréhension des objectifs à atteindre ! Même à distance, je suis disponible pour vous aider/vos débloquer...

---

[2] https://www.ecosia.org/search?q=CRUD
[3] Attention, vos codes seront très certainement scrutés par un logiciel anti-plagiat, du type MOSS (https://theory.stanford.edu/~aiken/moss/)

- Les **options** sont essentiellement là pour satisfaire la gourmandise des plus expérimentés d'entre vous. Mais elles n'ont – comme leur nom l'indique – rien d'obligatoire. Il est tout à fait possible d'avoir 18/20 sans implémenter la moindre option.
- Des **consignes complémentaires** vous seront données ultérieurement concernant l'évaluation du projet. Mais *a priori* il n'y aura qu'un jalon de livraison : la livraison finale.

### 2.3.Encadrement

Ce projet encadré n'est ni un TP ni un projet en temps libre. Avec la nécessité de l'enseignement à distance, votre autonomie sera soumise à rude épreuve. Pour autant, je reste disponible pour répondre à toutes les questions que vous vous poserez (à la fois en terme de compréhension du besoin qu'en terme de modélisation ou sur des points techniques java).
Je vous invite à privilégier le canal Teams pour vos questions qui pourraient intéresser vos camarades. Ce sera également la zone de partage de documents annexes que je posterai ultérieurement pour le projet.

## 3. Consignes générales et conseils

- Il est **très fortement conseillé** de mettre le code en anglais (noms de types, variables *etc.*).
- Le ***pattern MVC***, ou une de ses variantes, est **obligatoire** pour la structuration de vos modèles objet.
- La rédaction de commentaires (pertinents) et la génération de la javadoc sont **obligatoires**.
- Il est, évidemment, **recommandé** de faire des sauvegardes régulières et de gérer un **versionning** mimimum.
- Pour une meilleure maintenabilité, tâchez de réduire au maximum l'usage dispersé des chiffres et des chaînes de caractères dans vos codes. Utilisez plutôt, autant que possible, des constantes de classe.
- Evitez à tout prix (of course!) l'utilisation de chemins de fichiers en absolu et utilisez systématiquement les **séparateurs génériques** (tels `System.lineSeparator()` ou `File.separator`). Et si ça peut achever de vous convaincre, sachez que le correcteur évaluera votre travail sous MacOS...
- **Posez des questions !!**

## Conclusion

Des documents annexes viendront compléter cette présentation générale du projet. Si vous décelez des incohérences, merci de m'en informer.

Bon courage, bons projets !

# Annexes

## 4. Annexe 1 : Compléments

En complément du paragraphe 1 qui décrit le sujet (voir plus haut), vous trouverez ci-dessous des éléments de modélisation complémentaire. Bien qu'ils soient en annexe, ce sont des éléments importants, non optionnels, à prendre en compte dans votre modélisation. J'en profite pour éclairer – j'espère – quelques points.

- Pour être clair : on attend de vous la livraison de deux applications : **l'application principale** avec notamment la visualisation des pointages, et l'**émulateur de pointeuse**. Des informations transitent de l'émulateur vers l'application principale à chaque nouveau pointage.
- Les notions d'heure d'arrivée et d'heure de départ (dans les plannings) ne servent que de référence pour le calcul du stock d'heures supplémentaires. En aucun cas un employé peut être "empêché" de pointer s'il part en avance par exemple, ou s'il a cumulé trop de retard... L'application fait simplement du monitoring, charge ensuite au management/aux RH de gérer.
- Les heures d'arrivée et de départ des plannings des employés ne sont pas contraintes, à une exception près : elles doivent toutes les deux être dans la même journée calendaire (Par exemple un début de journée à 23h et une fin de journée à 7h le lendemain n'est pas possible. En revanche on accepte un début de journée à 17h et une fin de journée à 23h45).
- *Toutes* les heures manipulées sont arrondies au quart d'heure près.
- La *sérialisation*, qui vous est suggérée pour la sauvegarde des données, n'est pas une technique qui serait idéale, dans la pratique, pour la sauvegarde des données. Mais dans le cadre de ce projet, elle présente l'avantage de la simplicité et vous permettra de dégager du temps pour d'autres tâches.
- La gestion des **identifiants** des employés devrait idéalement se faire via des UUID[4]. Cependant, suite aux simplifications apportées au sujet, je vous invite à utiliser un simple entier incrémental (1, 2, 3, ...) ; l'objectif est de faciliter l'utilisation de l'émulateur de pointeuse.

## 5. Annexe 2 : 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. 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).

2. Permettre de **sérialiser** aussi les paramètres dans un fichier de configuration.

3. Dans F3, permettre une vue partielle des pointages (filtrée par personne, par département et/ou par date/période)

4. Mettre en évidence (dans F3, et éventuellement F4) des **incidents de pointage**
   4.1. Pointage d'arrivée en retard de plus de X minutes
   4.2. Pointage de départ en avance de plus de X minutes

5. Gérer plusieurs pointeuses.

[4] https://docs.oracle.com/javase/8/docs/api/java/util/UUID.html