

# Modélisation et implémentation d'un Pokédeck

TP 3-4

Programmation objets, web et mobiles en Java  
Licence 3 Professionnelle - Multimédia

Pierre Talbot (ptalbot@hyc.io)  
Université de Pierre et Marie Curie

13 octobre 2014



## 1 Modalité

- Ce TP est à réaliser *seul* avec un gestionnaire de version, votre dépôt sera cloné le 3 novembre au soir.
- Si le TP vous semble trop dur ou trop facile, n'hésitez pas à nous contacter pour adapter le sujet.

Vous pouvez récolter des points bonus :

- +1 point si le code et la documentation sont en anglais.

- +1 point si le rapport est en anglais.
- Spécifiez dans le rapport toutes fonctionnalités supplémentaires non précisés dans l'énoncé si vous pensez que ça mérite des points supplémentaires.

Les points donnés pour le code prennent en compte ces paramètres :

- Clarté du code (bon découpage en petites méthodes, bon nommage des variables, ...).
- Redondance du code (pas de répétition!).
- L'extensibilité (peut-on ajouter facilement une fonctionnalité?).
- La justesse du code.

Finalement quelques notes :

- Ne vous sentez pas obliger d'utiliser le polymorphisme (cours 3) à tout prix, ni l'héritage d'ailleurs. Ce sont des outils qu'il faut garder en tête. Une mauvaise utilisation ou non-justifiée est identique à une absence d'utilisation dans un cas justifié.
- Préférez implémenter moins de fonctionnalités mais bien.
- Faciliter la vie de l'utilisateur, si vous hésitez entre deux choix, préférez celui qui est ergonomique pour l'utilisateur. Si ils sont équivalents, choisissez ce qui est le plus simple à implémenter et relire.

## 2 Préliminaires

### 2.1 Mailing list

Une mailing list est un email auquel vous pouvez vous abonner. À chaque fois qu'un utilisateur envoie un email sur cette liste, tous les abonnés reçoivent ce mail. Pour vos questions concernant ce projet ou le cours en général, vous pouvez utiliser la mailing list mise en place, c'est très facile à utiliser.

- S'inscrire : Envoyer un mail à [java-upmc+subscribe@googlegroups.com](mailto:java-upmc+subscribe@googlegroups.com)
  - Se désinscrire : Envoyer un mail à [java-upmc+unsubscribe@googlegroups.com](mailto:java-upmc+unsubscribe@googlegroups.com)
  - Envoyer un mail : Comme d'habitude mais à l'adresse [java-upmc@googlegroups.com](mailto:java-upmc@googlegroups.com)
  - Répondre : Répondez à l'adresse du groupe.
- N'hésitez pas à vous entraidez sur le groupe.

### 2.2 Gestionnaire de version

Pour ce TP, vous utiliserez un gestionnaire de version. Il existe plusieurs services mais nous vous conseillons [github.com](https://github.com), nous fournissons les explications pour celui-ci. Libre à vous d'utiliser autre chose d'accessible en ligne, mais on ne garantit pas pouvoir vous aider.

1. Créer un compte sur [github.com](https://github.com).
2. Vous pouvez dès lors créer un répertoire. Lors de la création, ajouter une licence à votre projet (MIT ou Apache2 par exemple).

3. Envoyer l'adresse de ce répertoire par mail, tout de suite, à [talbot@ircam.fr](mailto:talbot@ircam.fr), vous serez noté grâce à celui-ci.

Vous pouvez l'installer sur Windows avec cet exécutable <http://msysgit.github.io/>. Il y a quelques commandes principales à connaître :

- Cloner un répertoire : `git clone https://github.com/ptal/nominalworkbench.git`, il est téléchargé et un dossier est créé.
- Ajouter un fichier au projet : `git add Pokedex.java`
- Faire un *commit* : `git commit -a -m "Initial commit"`, l'option "-a" commit tous les changements des fichiers qui ont été ajouté.
- Upload des changements locaux (via commit) sur le serveur distant : `git push`
- Download des changements du serveur sur votre machine locale : `git pull`, à priori seulement utile si vous travaillez sur plusieurs machines ou à plusieurs (pour synchroniser votre pc portable et le pc fixe par exemple).
- Pour un tutoriel ou plus d'infos voir <http://git-scm.com/docs/gittutorial>

Il y a des plugins pour les IDEs qui incorporent cet outil directement. Nous vous conseillons néanmoins de vous familiariser d'abord avec la ligne de commande pour saisir les concepts généraux de git. Ensuite vous devez respecter ces règles :

- Des petits commit, un pour chaque fonctionnalité, résolution de bugs, ...
- Le message dans un commit doit être clair et explicite, généralement il reste assez court quand même, mais si besoin ne pas hésiter à être plus verbeux.
- Le code que vous committez doit être compilable (pas d'erreur de compilation).
- Faites fréquemment des *push*, au moins une fois par jour, au cas où votre pc crash, et pour qu'on puisse suivre votre évolution.
- Vous ne devez pas versionner les binaires (pas de `.class` ou `.jar` par exemple). Tout ce qui est généré n'est pas versionné.
- Soyez cohérent, si vous codez en anglais, les messages des commits seront en anglais. Vous pouvez utiliser le français mais utilisez-le partout, pas de franglais.

### 2.3 Bonus : Architecture du projet

**Bonus.** Ré-organisez votre projet pour utiliser Maven. Voir *Maven by Example* et *Maven — The Complete Reference*, disponible gratuitement en pdf et html.

## 3 Pokédeck

Le but de ce TP est d'implémenter un Pokédeck, c'est-à-dire un logiciel permettant de gérer ses cartes Pokémon, pour le jeu de carte tel que décrit dans ce livret <http://assets19.pokemon.com/assets/cms2/pdf/trading-card-game/rulebook/xy1-rulebook-en.pdf>. Ce fascicule est la description du projet mais il y a trop d'information, à vous de trier ce dont vous avez besoin.

### 3.1 Analyse et modélisation

*Note* : Cette partie est à faire en parallèle avec les autres questions.

Vous devez analyser le domaine de ce problème (les cartes Pokémon) et trouver une représentation informatique vous permettant de le résoudre. Concrètement, vous pouvez commencer par réaliser un diagramme de classe en notant les différentes classes que vous pensez devoir créer, ainsi que leurs attributs et méthodes. Si le problème vous semble trop compliqué, vous devez trouver vous-même un sous-ensemble du problème qui vous semble plus facile d'accès. Par exemple, commencez par faire un menu avec les différentes options et chaque option fait quitter le programme. Vous implémenterez chaque fonctionnalité une par une par la suite.

Dans votre rapport, vous présenterez votre projet de façon à ce qu'une personne tierce puisse comprendre votre architecture et ajouter de nouvelles fonctionnalités facilement. Bien sûr cette personne connaît le Java, donc inutile d'expliquer les détails. Vous pouvez inclure un diagramme de classe ou tout autre diagramme si vous pensez que ça aide et clarifie la présentation (MS-Visio, Bouml ou Dia (2 derniers sont gratuits) permettent de faire des diagrammes). Le but est que le *Design Rational* (pas vraiment d'équivalent français) de votre projet transparaisse, il s'agit de justifier les choix architecturaux de votre projet, et éventuellement de montrer que les autres solutions posent problèmes.

### 3.2 Les bases

Ce logiciel devra nous permettre de faire 2 actions de base :

1. Ajouter la description d'une nouvelle carte.
2. Supprimer une carte.

La représentation d'une carte sera importante ici. Vous pouvez notamment réfléchir à l'utilisation de POJOs<sup>1</sup>, leur utilisation n'est pas obligatoire, utilisez-les si vous pensez que ça rend le code plus clair. Justifier votre choix dans votre analyse (dans le rapport).

### 3.3 Extension

Le client n'est plus content et veut trois nouvelles fonctionnalités :

1. Mettre-à-jour une carte.
2. Consulter sa collection.
3. Rechercher des cartes sur différents critères, deux critères minimum (par exemple numéro de carte ou nom de la carte).

Discuter dans votre rapport de comment vous avez étendu votre projet. Avez-vous du changer l'architecture générale car on ne pouvait pas ajouter ces fonctionnalités facilement ? Quelles étaient vos erreurs ? Et qu'avez-vous bien fait ?

---

1. Voir cours 3 et [http://en.wikipedia.org/wiki/Plain\\_Old\\_Java\\_Object](http://en.wikipedia.org/wiki/Plain_Old_Java_Object)

### 3.4 Sauvegarde

Le problème de notre Pokédeck est qu'il ne dispose pas de fonction de sauvegarde, donc l'utilisateur doit chaque fois ré-entrer toutes ses cartes. Pas pratique :(. Nous allons donc permettre à notre utilisateur de sauvegarder ses cartes. Ajoutez une fonctionnalité permettant de sauvegarder les cartes dans un fichier grâce à la *serialization*. Pour cela, utilisez l'interface *Serializable*, voir le tutoriel <http://ydisanto.developpez.com/tutoriels/java/serialisation-binaire/>.

**Bonus.** La *serialization* telle que présentée est binaire, donc le fichier créé n'est pas lisible à l'œil nu et difficilement ré-utilisable par d'autres applications ou échangeable via le réseau. On peut donc utiliser un format texte comme XML ou JSON. Par exemple, le projet suivant utilise un format en XML ([https://github.com/codlab/android\\_pokemon\\_tcg/blob/master/app/src/main/res/xml/e01.xml](https://github.com/codlab/android_pokemon_tcg/blob/master/app/src/main/res/xml/e01.xml)), malheureusement il n'est pas documenté. Vous pouvez essayer de trouver des bases de données XML ou JSON de carte Pokémon et les utiliser dans votre application. Je vous conseille d'utiliser le format JSON (qui est légèrement moins verbeux et plus facile à comprendre), vous pouvez trouver un tutoriel sur l'utilisation d'une librairie JSON ici : <http://www.mkyong.com/java/how-do-convert-java-object-to-from-json-format-gson-api/>. Vous ajouterez cette dépendance à votre projet Maven (bonus).

### 3.5 Découplage interface utilisateur et classes métiers

Vous pouvez découpler la logique de votre code (l'ajout d'une carte dans la liste, les opérations de recherche, ...) de l'interface utilisateur (la console ici). Ceci vous donnera l'intuition nécessaire pour comprendre plus facilement comment on utilise des interfaces graphiques et comment celles-ci interagissent avec le code.

Le but de ce découplage est que les classes métiers, par exemple la classe représentant la collection de cartes Pokémon, n'utilisent plus du tout la console, qu'elles n'affichent rien, ni ne demandent rien. On laissera ce travail aux classes de l'interface utilisateur. Concrètement, vous avez une classe d'interface graphique `Menu` qui affiche le menu (output) et attend de voir ce que l'utilisateur choisit (input), en fonction du choix de l'utilisateur, cette classe peut éventuellement :

- Afficher un sous-menu (par exemple pour simplifier la recherche).
- Appeler une des classes métiers pour demander un service (ajouter une carte par exemple).

On en déduit que ce sont les classes de l'interface utilisateur qui contiennent celles métiers.

**Bonus.** Lorsque l'utilisateur fait un choix dans le menu, vous ne devrez plus tester celui-ci explicitement (via des if ou un switch) mais implicitement, et que l'action correspondant à ce choix soit appelée implicitement. Indice : utiliser la classe `HashMap` où la clé est l'identifiant du menu (ce que l'utilisateur entre pour entrer dans un menu) et la valeur est une classe abstraite `Action` dont hérite chacune des actions correspondantes (Ajouter, Supprimer, ...).

### **3.6 Bonus, jeu Pokémon**

Si vous avez implémenté tous les points précédents (dont les bonus), vous pouvez coder un sous-ensemble des règles du jeu de carte. Expliciter dans le rapport quel sous-ensemble vous implémentez et discuter du design de l'application.