ANTOGNELLI Pauline
FARCHAK Marouane
TESTART Léandre
SZWARCBART Anna

Java Programming
PréIng 2
05.12.2024

RAPPORT JAVA TEXTE

I. Organisation de l'équipe

A. Méthodes

Pour le projet, étant peu disponibles pour travailler en groupe en dehors des heures allouées au projet, nous avons décidé de nous répartir le travail.
Suite à la conception de l'UML, nous avions une idée de ce que chaque classe était censée faire. Ainsi, nous avons scindé le développement par classe : chaque membre de l'équipe avait pour objectif de construire 1 à 2 classe qui lui était attribuée.

Afin de nous partager le code, et ayant peu de personnes compétentes en Git dans l'équipe, nous avons décidé à la majorité de nous partager les classes une fois terminée par la personne qui s'en occupe.

Aussi, nous communiquions via un groupe whatsapp afin d'échanger au fur et à mesure de l'avancée du projet de nos idées et visions ainsi que sur les difficultés liées au code. Nous avons réalisé que cette méthode n'est pas optimale mais à défaut de mieux nous avons tout de même pu nous tenir au fait de l'avancée de chacun.

B. Problèmes et limites rencontrés

La répartition du travail a rapidement eu des limites. Le fait que chacun implémente sa classe à sa façon, nous à causé plusieurs problèmes :

- Chaque classe est implémentée différemment, avec un structure différente
- Les libertés prisent sur une classe pouvait rendre caduc des méthodes d'autres classes
- La programmation objet n'ayant pas été acquises par l'ensemble des membres, certaines classes produites étaient à revoir dans leur ensemble

Ces problèmes ont mené à une revue intégrale des classes du projet par le membre ayant le plus d'expérience en orientée objet, et ce dans un délai fortement limité.
Ainsi, une cohérence globale a pu être assurée, modulo de possibles erreurs ou oublis.
Cette solution n'était évidemment pas optimale, mais nous a permis d'atteindre ce rendu.

Le problème complémentaire nous ayant mené à devoir agir dans de courts délais, était notre manque de communication. Certes nous maintenions un groupe whatsapp entre nous, mais nous n'avons pas pu voir ce que les autres faisaient, et plus encore comment ils réfléchissaient au développement du projet.
En faisant des points réguliers où chacun présentait son code, nous aurions pu voir plus rapidement les points de vues de chacun et ainsi, limiter les risques rencontrés.

Aussi, après réflexion, nous n'aurions pas dû répartir notre charge de travail par classes à développer, mais plutôt en fonctionnalités distinctes à apporter.
Par exemple, la réalisation d'un dépôt dans une poubelle est indépendante de la création d'un contrat de commerce.

Enfin, afin de faciliter notre développement en équipe et de faire monter en compétence chacun, nous aurions dû travailler sur un git collaboratif, en faisant du Pair Programming.

II. Evolution de l'UML

En commençant à coder le projet, nous nous sommes rapidement rendu compte que l'UML que nous avions préparé n'était finalement pas adapté à notre vision de développement du projet.

Sur le papier, les classes, leurs attributs et méthodes imaginés au début du projet fonctionnaient entre elles. Mais au moment de traduire l'UML en Java, nous avons remarqué que certains des attributs n'étaient finalement pas utiles et que d'autres dont nous avions besoin manquaient.

Par exemple, pour la classe "Ménage" nous avions décidé en UML d'inclure un "idUser", un "nbPoints" et un "bonAchat", sauf que cela n'était pas adapté à notre vision.
Le premier attribut ("idUser"), n'était pas nécessaire car nous avons décidé qu'il était plus adapté que la classe "Ménage" ait un "username" et un "password" afin qu'il puisse se connecter et créer un compte dans l'application.
Le second ("nbPoints") aurait pu apparaître : nous avons préféré maintenir une unique source de vérité de point de façon à ce que cela soit lié au ménage : notre fonction "ajouterDepot" dans la classe "Poubelle". Cela fait plus sens, étant donné qu'un nombre de point est donné (ou retiré) à nos "Ménages" pour chaque dépôt.
Le troisième ("BonAchat") n'était pas adapté car nous n'avions pas considéré qu'un "Ménage" pouvait échanger plusieurs bons d'achats contre des points, tout en les stockant. Nous avons donc mis en place un référentiel des "BonAchats" dans "Application", car cela faisait plus sens pour nous de les centraliser à l'endroit où l'utilisateur peut récupérer l'information, plutôt que d'avoir une liste indépendante à chaque ménage. Nous avons également créé une classe "BonAchat" pour répondre à l'ensemble de nos besoins.

III. Explication de l'implémentation du projet

A. Classes supprimées

Les classes suivantes étaient modélisé dans notre UML, et ont été supprimé de notre implémentation Java :

- Les classes "Historiques" : "HistoriqueBonsAchats", "HistoriquePoints" et "HistoriqueDepots"
- Les classes "Poubelles de couleurs" : "PoubelleVerte", "PoubelleJaune" et "PoubelleBleue"
- La classe "Commerce"

Les classes "Historiques" tels que nous les imaginions sont finalement récupérables autrement de par notre modélisation:

- "HistoriqueBonsAchats" : via la liste "BonsAchats" de la classe "Application",
- "HistoriquePoints" : via la liste des "Dépôts" de chacune des "Poubelles" d'un "CentreDeTri" (historique des points gagnés par "Dépôt")
- "HistoriqueDepots" : également à travers la liste des "Dépôts" de chacune des "Poubelles" d'un "CentreDeTri"

Pour les "Poubelles de couleurs", nous avons préféré créer un type "TypeDechet" à la place de classes héritées : vu que nos poubelles sont intelligentes, nous avons pensés qu'il était plus adapté de se dire qu'elle ne peut recevoir que certains types de déchets, qu'elles trient automatiquement.
Pour la classe "Commerce", nous n'avions finalement besoin que du nom du "Commerce" pour les "bonAchats" et pour les "Contrats". N'ayant pas besoin d'information supplémentaire, nous n'avons gardé que le nom, à travers le type "String".

B. Classe Contrat, Depot et BonAchat

Nos classes "Contrat", "Depot", et "BonAchat" n'ont pas d'intelligence : elles nous permettent de regrouper plusieurs informations de façon logique, afin de les retrouver plus facilement dans nos autres classes.

C. Classe Menage

La classe "Ménage" quant à elle porte peu d'intelligence. Nous avons implémentés 2 fonctions "equals" pour faciliter le développement de nos méthodes "CreerCompte" et "Login" dans notre classe "Application".

D. Classe Poubelle

La classe "Poubelle" contient trois fonctions : "estPleine", "ajouterDepot" et "vider".

La fonction "estPleine" permet à la poubelle de prévenir le "CentreDeTri" auquel elle est reliée lorsqu'elle a atteint plus de 90% de sa capacité maximale. Ainsi, elle est ajoutée à la liste de poubelles pleines qui doivent être vidées lors de la collecte du "CentreDeTri" (déclenchement lors de l'ajout d'un "Dépôt").

La fonction "vider" consiste tout simplement à remettre la capacité de la poubelle à 0 : elle ne contient plus de déchets.

La fonction “ajouterDepot” est la plus complexe de cette classe mais aussi la plus importante dans la réalisation du projet.
Cette fonction permet à un utilisateur d’effectuer un “Dépôt” dans une poubelle intelligente en s’identifiant. Ensuite, elle fait gagner ou perdre des points au “Ménage” en fonction de son “Depôt”.
Cette fonction prend en entrée un “Ménage” pour simuler l’identification à la poubelle, le poids du “Dépôt” ainsi que le type de déchets déposés (“TypeDechet”).
La poubelle intelligente vérifie tout d’abord qu’elle a la capacité de recevoir un “Dépôt” du poids indiqué (si ce n’est pas le cas elle renvoie une erreur à l’utilisateur), puis elle calcule le nombre de points à attribuer. En cas d’un bon “Dépôt”, l’utilisateur gagne des points, en cas de mauvais “Dépôt” il en perd. On calcule le gain ou la perte de points en fonction du poids du “Dépôt” et du type de déchets attendu.

E. Classe CentreDeTri

La classe “CentreDeTri” contient plus de fonctions; celles relatives à la gestion des poubelles, la gestion des ménages, la gestion des contrats et la gestion des statistiques.
Pour la gestion des poubelles nous avons :

- “ajouterPoubellePleine” qui ajoute une poubelle à la liste des poubelles pleines lorsqu’elle notifie le “CentreDeTri” qu’elle l’est.
- “ajouterPoubelle” pour créer une nouvelle poubelle avec le type de déchets qu’elle peut recevoir et son emplacement. On lie la “Poubelle” à son “CentreDeTri”.
- “retirerPoubelle” qui simule le retrait d’une poubelle de son emplacement si elle est par exemple défectueuse ou n’a plus lieu d’exister
- “collecterDéchet” qui vide toutes les poubelles faisant parties de la liste des poubelles pleines, simulant la collecte des déchets.

Pour la gestion des ménages, nous avons :

- “getListeMénages” qui renvoie la liste de tous les “Ménages” inscrits dans notre “CentreDeTri”. Cela nous sert dans d’autres classes, notamment dans “Poubelle” pour vérifier que le “Ménage” identifié lors du “Dépôt” existe.
- “afficherListeDepotsMenage” qui permet de renvoyer la liste de tous les “Dépôts” d’un “Ménage”. Cette fonction va chercher tous les “Dépôts” à partir du référentiel stocké dans le “CentreDeTri”.

Pour la gestion des “Contrats”, nous avons trois fonctions simples qui nous aident à modifier les états des “Contrats” dans la classe “Application” : si le “Contrat” est passé, il n’est plus à mettre dans la liste des “Contrats” en cours, mais dans les “Contrats” passés. Ces fonctions sont :

- “ajouterContratEnCours”
- “ajouterContratPasses”
- “supprimerContratEnCours”

F. Classe Application

Nous avons essayé de créer notre classe “Application” avec l’idée de créer une interface utilisateur de nos “Ménages” pour interagir avec les données récupérées grâce aux “Poubelles” et au “CentreDeTri”.

Nous avons tout d'abord des fonctions "creerCompte" et "login" qui permettent respectivement à l'utilisateur de créer son compte auprès du centre de tri et de se connecter à son compte utilisateur grâce à son nom d'utilisateur ("username") et son mot de passe (password"). Nous avons ajouté à "creerCompte" une fonctionnalité qui empêche l'utilisateur d'utiliser un nom d'utilisateur déjà inscrit. Aussi, lors du "login", nous vérifions que l'utilisateur existe déjà dans la liste des "Ménages" du "CentreDeTri", grâce à son nom d'utilisateur et son mot de passe. Aussi, dans la continuité, nous avons une fonction "logout" qui permet à l'utilisateur de se déconnecter.

Nous avons ensuite plusieurs fonctions destinées aux actions que l'utilisateur ("Ménage") peut effectuer avec son compte :

- "consulterPointsTotal" pour qu'il puisse vérifier le cumul de tous ses points
- "afficherListeCommerces" pour que l'utilisateur puisse visualiser tous les commerces partenaires du centre de tri où il pourra utiliser son "BonAchat". Aussi, dans cette fonction, nous mettons à jour la liste des partenaires via les "Contrats" effectués avec le "CentreDeTri". C'est à dire, que nous vérifions à chaque fois la date de fin du "Contrat" : si elle est avant la date d'aujourd'hui, nous l'ajoutons à la liste des "Contrats" passés et la supprimons de la liste des contrats en cours. La liste est donc à jour à tout moment.
- "convertirPointsEnBon" qui permet à notre utilisateur de convertir ses points de fidélités en "BonsAchats". Il vérifie que l'utilisateur a le nombre de points nécessaire pour créer un "BonAchat" (50 points). Si c'est le cas, chaque tranche de 50 points est retirée au total possédé par le "Ménage". Pour chacune de ces tranches de points, une valeur de 10€ est donnée au "BonAchat". L'utilisateur doit également sélectionner le commerce où il souhaite dépenser son "BonAchat".
- "afficherNbBonAchat" pour afficher la liste de "BonsAchats" de l'utilisateur ("Ménage" connecté).

IV. Problèmes rencontrés lors du testing du projet

Nous avons rencontré au moment de tester notre code quelques problèmes : plusieurs de nos fonctionnalitées étaient défectueuses. En effet, nous n'avions pas testé notre code au fur et à mesure du développement du projet.

Par exemple, pour la création du compte d'un "Ménage" dans notre "Application", nous avions prévu que deux utilisateurs ne puissent pas avoir le même "username" en utilisant un '.equals' et en renvoyant une erreur à notre utilisateur s'il tente de créer un compte avec un nom d'utilisateur déjà existant. Cela nous permettait de distinguer les "Ménages" lors du "login".
Cette fonctionnalité ne s'effectuait pas correctement lors du test car nous avions utilisé un '==' au lieu d'un '.equals' avant de nous rendre compte de notre erreur lors du test.

Nous avons aussi constaté au niveau de la fonction "login" de notre "Application", que lorsque l'utilisateur voulait se connecter, au lieu de trouver le "Ménage" associé à cet username, notre application renvoyait une erreur.
En effet, nous avions oublié de mettre le "username" et le "password" en paramètre du constructeur de "Ménage" : c'était d'autres champs "String" qui étaient remplis à la place. Cela ne fonctionnait donc pas lorsque l'on vérifiait la donnée comme prévue.

Une fois que ces quelques problèmes de première instance ont été résolus, nous avons pu passer au testing de nos différentes fonctionnalités, notamment celles associées à la “Poubelle” intelligente : le “Dépôt” de déchets, son ajout et son retrait de points au “Ménage” qui l’a réalisé. Ici encore nous avons rencontré une petite difficulté : les fonctionalités liées aux points renvoyaient des erreurs. A force de recherchesn nous nous sommes rendus compte que nous avions oublié de définir dans le constructeur de “Ménage” que le nombre de points initial était fixé à 0, afin de pouvoir l’incrémenter ou le décrémenter à partir de là.

Malheureusement, nous n’avons pas eu le temps de tester l’ensemble de nos fonctionalités, si nous l’avions fait nous aurions sûrement pû détecter d’autres potentielles erreurs.

V. Critiques et limites

Le regret principal que nous avons est de ne pas avoir eu plus de temps pour travailler sur ce projet afin de créer et de finaliser certaines fonctionnalités qui auraient rendu notre projet plus professionnel et utilisable dans un véritable environnement.
Par exemple, nous aurions pu :

- Faire un hachage du mot de passe pour plus de sécurité
- Créer des types de “BonsAchats” différents
- Parfaire notre système de “Poubelles intelligentes” (Ex. : Simulation des faux-positifs et faux-négatifs de la “Poubelle”)
- Sauvegarder nos données au format json ou csv : nous avons considéré que notre application n’avait pas besoin de tourner avec une base de données et conservait en mémoire les informations nécessaires sans avoir à les sauvegarder. Évidemment, nous aurions voulu réaliser cette fonctionnalité, mais celle-ci nous semblait moins critique que les autres.

Nous aurions également préféré être plus aidé dans la répartition des tâches du projet, même si ce projet nous a appris à mieux nous organiser pour les prochains.

Aussi, nous aurions aimé étudier plus en détail l’interaction de nombreuses classes dépendantes entre elles : le projet nécessite une harmonie et une cohérence entre de nombreuses classes, ce qui n’est pas facile de prime abord.

Enfin, nous sommes un peu déçus de ne pas avoir été plus “challengés” en classe sur la compréhension du sujet, ou sur l’importance de discuter en profondeur le projet en groupe avant même de commencer à coder : comprendre la direction dans laquelle partir dès le début est nécessaire pour mener à bien le projet.

VI. Conclusion

Malgré les limites et critiques du projet soulignées dans la partie précédente, nous sommes fiers et heureux d’avoir pu compléter dans les temps ce projet qui nous a challengé dans sa conception et son implémentation.

Dans l’ensemble, le projet nous a paru plutôt adapté à notre niveau, étant donné que nous avions vu et travaillé en cours tous les outils dont nous avions besoin pour l’implémenter, bien que nous avons trouvé difficile de mobiliser les bonnes connaissances au bon endroit et au bon moment.

Aussi, contrairement à ce que nous avons pensé lorsque nous avons reçu le projet, le volume de code à écrire s'est révélé plutôt faible. Nous nous attendions à nous retrouver avec des classes très longues et des méthodes particulièrement volumineuses en lignes de code, mais ce ne fut pas le cas à notre bonne surprise.

Ce qui a été finalement plus compliqué dans ce projet, ce sont les réflexions qui l'entourent. En effet, le projet en lui-même est plutôt complet et nous indique assez facilement la direction générale à prendre, mais quand il en vient aux détails, aux petites fonctionnalités et aux manières de voir ou penser certains objets, fonctionalités ou aspects, il nous laissait beaucoup de libertés d'interprétation.

Déjà, cette réflexion nous a pris du temps individuellement mais le plus gros problème est que, par manque de temps et d'organisation, nous n'avons pas réfléchi à ces libertés d'interprétations en groupe et sommes donc partis chacun avec nos propres interprétations. Cela a rendu la mise en commun du projet tout particulièrement difficile.

Une fois que nous avons terminé de travailler sur java texte, il nous a semblé que ce projet nous a fortement aidé à comprendre la programmation objet, et surtout son utilité. Nous en avons retiré beaucoup de connaissances ainsi que de l'aisance avec le langage de programmation Java, sa syntaxe et sommes désormais capable de coder en Java assez souplement. Il semble tout de même important de noter que la montée en compétence est assez hétérogène dans l'équipe, la quantité de travail effectuée par chacun n'étant pas la même.

En conclusion, ce projet, bien que dur et assez chronophage, nous a réellement aidé à comprendre et mettre en application notre connaissance du langage Java et de la programmation orientée objet. Le seul reproche que nous pouvons faire à ce projet est le manque de temps dont nous avons souffert, laissant plusieurs de nos idées sur le côté, et nous laissant une sensation de frustration de ne pas pouvoir pu rendre un projet qui nous semble parfait.