Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

2019 – 2020
DEV2

DEV2 – Laboratoire Java

Consignes

Réalisation et évaluation de projet

Table des matières

Lisez bien et complètement ce document. Il vous renseigne sur les modalités d'évaluation de votre projet.

1 S'inscrire au projet

L'étape préalable au projet est de vous **inscrire auprès de votre professeur·e de laboratoire** qui va vous créer un dépôt sur le serveur GIT de l'école. C'est ce dépôt que vous utiliserez pour développer votre application et pour la remettre à votre enseignant ou enseignante.

Seul un projet se trouvant dans un dépôt *git-esi*[1] sera évalué.

2 Échéances

Comme indiqué dans le ou les énoncés, il y a plusieurs échéances importantes : celles correspondant à une remise et la défense.

1. Désigne le serveur gitlab de l'ESi se trouvant à l'adresse suivante `https://git.esi-bru.be`

Vérifiez avec la personne qui vous évalue, les modalités exactes de remises sous peine de voir votre travail non corrigé.

- ▷ Quelle est la date limite précise [2] ?
- ▷ Comment le remettre ? En main propre ou pas, avec une petite démo ou pas...

3 Défense

Le jour de la défense une ou plusieurs nouvelles fonctionnalités vous seront demandées. Vous devrez modifier votre projet en conséquence.

Ces modifications seront suivies par une défense orale, soutenue par une présentation sur machine de votre projet. N'oubliez pas que la défense est effectuée à l'école, sur une machine quelconque du laboratoire. Vous devez être capable de compiler, modifier, exécuter et expliquer votre code.

4 IDE et corollaires

Nous avons décidé d'utiliser l'IDE *Integrated Development Environment* (EDI *Environnement de Développement Intégré* en français) Netbeans pour faciliter le développement. L'utilisation d'un IDE facilite et accélère le développement.

Les corollaires de l'aide apportée par Netbeans sont les exigences suivantes.

Exigences rédhibitoires :

- ▷ le code doit être convenablement indenté.
  Rappel : presser Shift-Alt-F indente automatiquement le code) ;
- ▷ le code doit compiler (**aucune** erreur de compilation ne sera acceptée) ;
- ▷ la Javadoc doit être complète. Chaque méthode de chaque classe doit posséder une Javadoc ;
- ▷ toutes les méthodes pour lesquelles des tests sont demandés doivent posséder des tests unitaires ;
- ▷ tout écart par rapport à l'énoncé (ajout d'une méthode non privée, modification du comportement d'une méthode demandée) devra être validé au préalable par la personne qui évalue qui jugera de sa pertinence ;

Exigences simples :

- ▷ utilisez les éléments de langage les plus appropriés à ce que vous devez faire : par exemple, un *foreach* quand c'est possible ;
- ▷ tout le développement est en anglais : le code (les noms de variables, de méthodes, de classes, etc.), la Javadoc. Vous pouvez cependant écrire la documentation complémentaire dans le code ainsi que l'interaction avec l'utilisateur en français ;
- ▷ les méthodes doivent être courtes sauf cas particuliers rares. Vous écrirez donc des méthodes privées complémentaires quand cela est nécessaire pour satisfaire cette condition ;

2. On entend par *date limite*, un moment défini par date et heure, pour lequel le travail doit être remis au professeur, et au-delà duquel le travail ne sera plus accepté. Mais rien ne vous empêche de remettre votre travail plus tôt.

- ▷ le respect attentif des conventions d'écriture du code Java — du moins les plus importantes — telles que décrites par Oracle (voir [pdf [3]]) ;
- ▷ respectez l'orthographe autant que faire se peut ;

Faites des *commit* et des *push* réguliers. Ceci vous donne la possibilité de revenir en arrière et nous permettra, le cas échéant, de suivre l'évolution de votre travail.

Le fait de pousser votre projet sur *git-esi* ne vous dispense pas de faire des *backup* (sauvegarde en français) de votre projet. Le serveur *git-esi* n'est pas immortel.

Lorsque vous codez, ne vous limitez pas à l'implémentation des méthodes demandées. Vous êtes encouragés à écrire des méthodes **privées** qui simplifient l'écriture de votre code si cela est nécessaire. Il en va de même pour les méthodes `toString()`, `equals()` et `hashCode()` qui sont à récrire si vous en ressentez l'utilité.

5 Évaluation

L'évaluation du projet ne portera que sur ce que vous avez remis.

Il est inutile de remettre un projet qui ne respecte pas les exigences rédhibitoires énoncées ci-dessus. Par exemple, il faut que les tests soient présents, c'est-à-dire que des fichiers de tests unitaires pour toutes les méthodes pour lesquels des tests sont mentionnés existent. Selon la qualité et la pertinence de ces tests, vous aurez plus ou moins de points.

Pour rappel, le projet compte pour 24% de la cote finale de l'UE DEV2. La pondération des différentes parties se trouve dans le ou les énoncés du projet.

Si une partie n'est pas remise, l'étudiant ou l'étudiante peut continuer son projet mais, bien sûr, iel n'aura pas de points pour les parties non remises.

La cote est attribuée au terme de la défense orale, faite à l'école, sur une machine de laboratoire aléatoire. La **défense est obligatoire** pour recevoir une cote de projet. Un étudiant ou une étudiante qui ne se présente pas à la défense du projet, a 0 pour le projet.

6 À propos de collaboration et triche

Contrairement à ce que beaucoup imaginent, la programmation est le plus souvent un travail de groupe, mené en collaboration étroite au sein d'une équipe qui compte au minimum des développeurs et des développeuse et des utilisateurs et des utilisatrices mais également des analystes, des graphistes… Vous pouvez bien sûr demander l'avis de vos camarades de classe pour les travaux préparatoires, la compréhension du projet, la réflexion sur les algorithmes, l'utilisation des outils, l'interprétation des messages d'erreurs et des comportements inattendus.

Cependant, dans le contexte scolaire, nous devons aussi mener un travail d'évaluation qui lui est individuel. Il est donc attendu que les étudiants et les étudiantes remettent un projet **individuel**. Nous ne pouvons pas tolérer de codes copiés. **Tout plagiat de code donnera lieu à l'annulation du projet des deux personnes concernées**.

Ne copiez donc pas le code d'un autre et ne laissez personne vous copier.

3. https://www.oracle.com/technetwork/java/codeconventions-150003.pdf *consulté le 10 mars 2020*

Chacun est responsable de la confidentialité de son code source. Ceci comprend la sécurité d'accès aux ordinateurs sur lesquels est stocké votre projet, la non divulgation de votre mot de passe, l'aspect privé de votre dépôt git...

7 Méthodologie de développement

Dans cette section, nous discutons brièvement de méthodologies de développement [4]. La première est liée au cycle de développement qui est dit itératif et la seconde est le développement dirigé par les tests [5]. Le cycle de développement itératif va vous imposer de remanier une partie de votre code source. Finalement, nous parlerons de ce qu'est le refactoring [6] et quelle aide peut apporter Netbeans.

7.1 Cycle de développement itératif

Il existe différents types de cycles de développement entrant dans la réalisation d'un logiciel. Ces cycles prendront en compte toutes les étapes de la conception d'un logiciel.

Le cycle itératif est le suivant :

- identification du besoin (quoi) ;
- élaboration (comment) ;
- développement ;
- transition (livraison au client, en l'occurrence votre enseignant ou enseignante).

En pratique, nous vous demanderons deux ou trois itérations qui représentent chacune une version de l'application avec un ensemble de règles plus ou moins complet. Pour chacune de ces itérations, l'identification du besoin et l'élaboration a été faite par nos soins. Vous êtes en charge du développement et de la présentation au client ou à la cliente.

Veillez donc à :

- **tester** en utilisant la librairie JUnit ;
- **documenter** via une Javadoc complète pour chacune de vos méthodes et pour chacune de vos classes ;
- **commiter** (et pusher) régulièrement votre projet et accompagner ce commit d'un message explicite sur le travail réalisé.

Pour la première itération, les tests vous sont offerts. Prenez exemple sur ceux-ci, afin de choisir méthodologiquement le nom de vos méthodes de tests de manière à retrouver la méthode testée ainsi que la description du cas :

```
test<NomMéthode><descriptionDuCas>()
```

Exemple pour une méthode `isFree(Coordinates position)` qui retourne `true` si la position donnée en paramètre est libre, sinon `false`.

```
@Test
public void testIsFreeWhenPosIsFree() {
  ...
  assertTrue(instance.isFree(position));
}
```

4. https://fr.wikipedia.org/wiki/Cycle_de_développement_(logiciel) *consulté le 10 mars 2020*
5. https://fr.wikipedia.org/wiki/Test_driven_development *consulté le 10 mars 2020*
6. https://fr.wikipedia.org/wiki/Réusinage_de_code *consulté le 10 mars 2020*

7.2 Développement dirigé par les tests

Le *test-driven development (TDD)* ou en français développement piloté par les tests est une technique de développement de logiciel qui préconise d'écrire les tests unitaires avant d'écrire le code source d'un logiciel.

Les étapes suivantes doivent être respectées :

1. écriture d'un premier test ;
2. vérifier qu'il échoue (car le code qu'il teste n'est pas implémenté) ;
3. écrire le code suffisant pour faire passer le test ;
4. vérifier qu'il passe ;
5. améliorer le code.

Par amélioration du code, on entend : vérifier le respect des conventions, améliorer l'aspect du logiciel, optimiser le code...

7.3 Refactoring avec Netbeans

Le *refactoring* de code est l'opération consistant à retravailler le code source d'un programme informatique — sans toutefois y ajouter des fonctionnalités ni en corriger les bugs — de façon à en améliorer la lisibilité et par voie de conséquence la maintenance.

Dans notre cas, le refactoring sera principalement dû aux modifications induites par les différentes itérations.