Université Nice Sophia Antipolis
Membre de UNIVERSITÉ CÔTE D'AZUR

École d'ingénieurs
POLYTECH NICE SOPHIA

Année : 2020-2021

Formation : SI

Examen : QGL

N° étudiant : 22006480

Note de l'épreuve

15.5/20

(1) Le Candidat doit inscrire ici : ses nom, prénoms, lieu et date de naissance, puis rabattre suivant le pointillé le coin de la copie et coller.

Il est interdit au Candidat de signer sa copie ou d'y inscrire un signe quelconque pouvant en indiquer la provenance.

(1) Nom : NIGET

Prénoms : Tom

Né à : [redacted]

le 18/12/2002

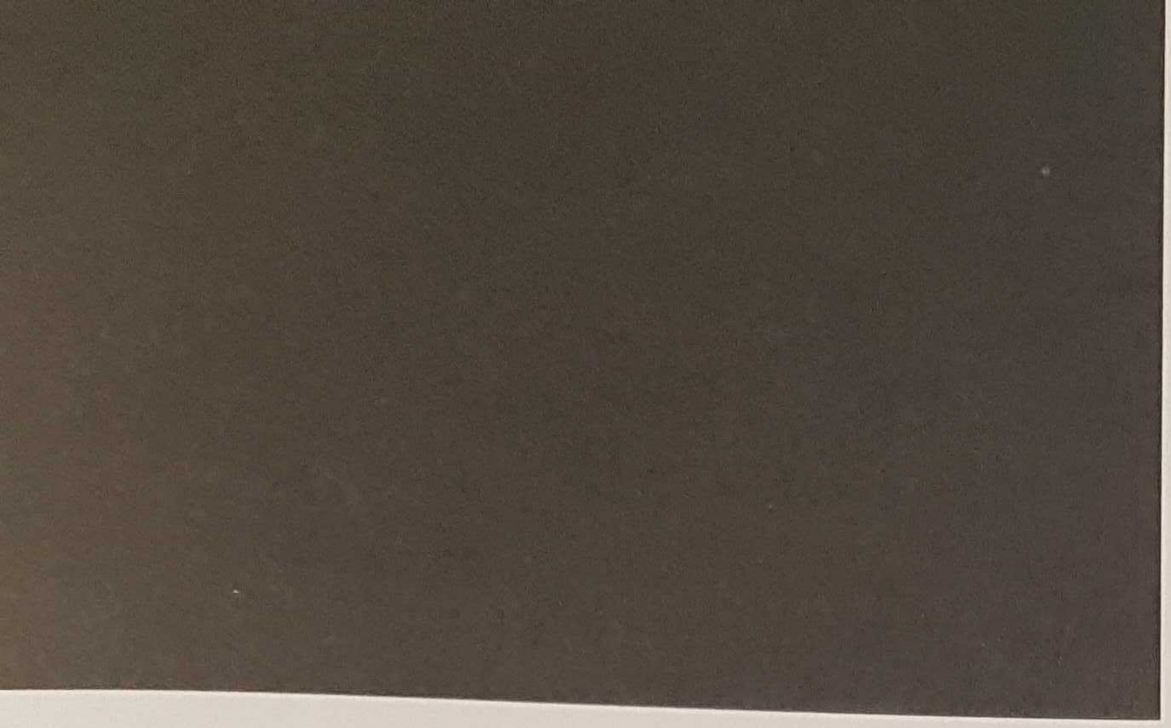
BIOT, le 27/05/21

4- Principes SOLID

a) Single responsibility (faire une chose et la faire bien)
Open (to extension), closed (to modification)
Liskov substition (pouvoir substituer par un objet de type plus spécifique)

Dependency inversion (dépendre d'abstractions, pas de détails d'impl)

b) Le LSP décrit une facette du concept général de polymorphisme, la relation "is a" à la base des hiérarchies objet. Le violer est contre intuitif, et sémantiquement douteux.
L'Open/Closed formalise la notion qu'un type plus spécifique doit se comporter "au moins" comme son parent, ç-à-d en pouvant étendre mais sans modifier. Le violer revient à violer le LSP. Le respecter offre des garanties quant à la maintenabilité du code sur le long-terme.
Le SRP décrit une bonne organisation du code : en attribuant à chaque composant une tâche bien précise il est à la fois plus simple d'étendre l'existant, et de trouver l'origine d'un bug



c) Keep it Simple & Stupid.

Écrire un code simple et compréhensible plutôt qu'un code "malin" et illisible.

Un code "malin" sera plus court et peut-être plus performant, mais beaucoup plus compliqué à déboguer, et rendre, ou tout simplement à relire.

3. Tests par mutation

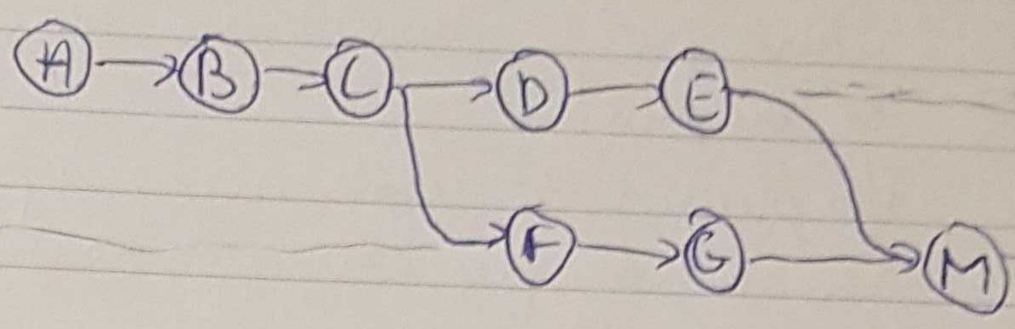
a) distance(0, 0, 0, 1) == 1

b) Retirer le Math.sqrt au début. $1^2 = 1$ donc le test n'est pas affecté.

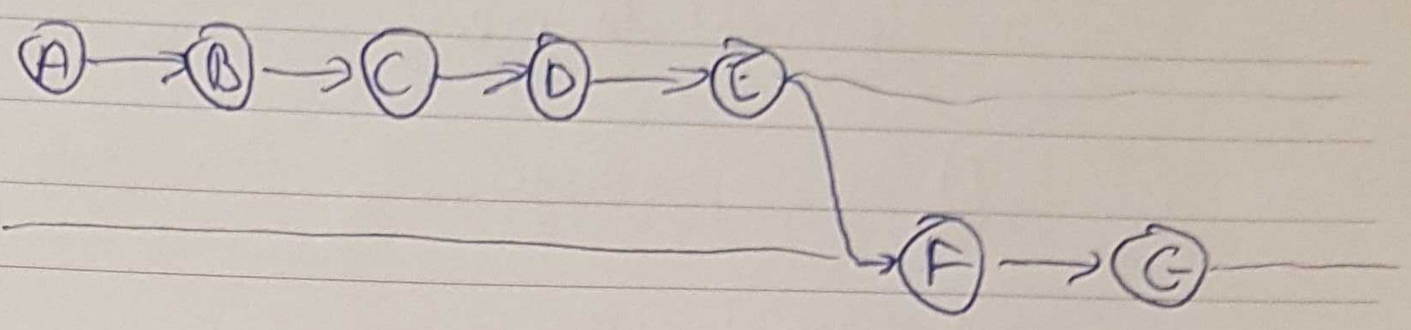
c) distance(0, 0, 3, 4) == 5

d) Les mutations permettent de trouver des tests qui passent "par coïncidence". Ici, le premier passait car $1^2 = 1$, qui n'est évidemment pas vrai $\forall n$.

2. Git

c) branch 1 

branch 2

d) branch 1 

branch 2

b) Les branches sont un outil essentiel de Git ; comme tout outil utilisé en équipe, il est important que son utilisation soit consistente. Des conventions de nommage / formatage gardent un code visuellement propre ; une branching strategy bien définie garde un arbre Git en bonne santé et sur le long-terme réduit le risque de problèmes du style énorme fusion entre 2 branches très divergentes, etc.

a) Les PR permettent à tous les membres de l'équipe de reviewer le code avant la fusion ce qui évite que du mauvais code se retrouve en branche principale. Sur des gros projets où tout le monde n'a pas accès en écriture sur le dépôt, les PR permettront de contribuer confortablement du code pour qu'il soit relu par des administrateurs.

6. Maven

a) Métadonnées du paquet (nom, version, ...)
Dépendances (librairies utilisées)
Plugins Maven (opérations à la compilation)

b) Oui, car on peut se connecter à un dépôt quelconque, y compris hébergé en local. Utile pour des déploiements sur des serveurs internes en entreprise par exemple où il est parfois souhaitable de ne pas relier une machine à Internet.

c) Une dépendance correspond à une librairie, un SDK utilisé par le projet, qui sera mis dans le classpath à la compilation.
Un plugin est utilisé mais pas par le code, mais par Maven lui-même, par exemple pour définir des actions supplémentaires de cycle de vie.

d) mvn package (si pom bien configuré et tests dans le bon dossier)

5) Automatisation

a) - Traitement du code (formatage, analyse)
- Compilation
- Lancement de tests
- Empaquetage
- Déploiement

} potentiellement sur plusieurs plateformes, automatiquement (ex: Windows, Linux ....)

b) 1. Vérification compilation (le code compile bien, même sur un serveur GitHub vierge ⇒ le pom est également OK)
2. Lancement tests (en complément du lancement avant de commit bien sûr)

3. Empaquetage / génération d'artéfacts ⇒ jar ensuite passé à la main sur le runner (car API du runner pas documentée + crédits limités).

Nous avons également un projet de mettre en action GH le lancement du mode "vérification" de notre simulateur qui teste automatiquement toutes les weeks, mais nous avons peur de consommer toutes les minutes GitHub Actions de l'école donc on le faisant en local.

c) Pas sûr d'avoir compris la question, mais par moi un automate est un algorithme mis en pratique, c'est quelque chose qui va effectuer systématiquement, rapidement et sans faire grève une opération pour laquel il a été programmé.
Un robot, mais stupide.

T. Niget -

QGL
N° 22006480

1. Métriques

b) Code Smells, Vulnérabilités, Debt, Lignes, complexité cyclomatique

a) Ça permet de voir de façon plus tangible la distribution du code dans le projet.

c) Vulnérabilités : un logiciel fiable et sécurisé est généralement préféré à un logiciel truffé de bugs les côtés et plein de failles.

Code Smells : Sonar est intelligent et peut trouver des problèmes dans le code qu'un humain ne verrait pas.