Contrôle continu
Licence Informatique (L2)
« Programmation objet avancée »

F. BERTRAND

Année universitaire 2020–2021

Préambule

Ce TP évalué sera réalisé **individuellement** et toute communication avec d'autres étudiants est **interdite** (je vous rappelle qu'il est assez facile de montrer avec des outils d'analyse que deux codes distincts proviennent de la même personne... Donc ne soyez pas tenté de partager votre travail avec d'autres personnes car la fraude concerne aussi bien la personne qui met son travail à disposition de tiers que ceux qui s'approprient ce travail).

Différentes classes de test sont fournies (cf. Moodle). Le code que vous développerez devra comporter les méthodes permettant d'exécuter ces classes de test sans déclencher d'erreur.

Remarques importantes :

— Vous rendrez uniquement les classes Java que vous aurez développées sous la forme d'une archive zip, les classes de test ne sont pas à rendre avec le projet (des classes similaires mais différentes seront utilisées pour tester votre code). Néanmoins les tests n'étant pas exhaustifs, le fait qu'une classe de test s'exécute sans erreur ne signifie pas obligatoirement que votre code est entièrement correct. Il sera donc contrôlé par chaque correcteur.

— Comme il sera nécessaire d'introduire des modifications aux classes développées au fur et à mesure des différentes questions, il est **fortement** recommandé de **sauvegarder l'ensemble des classes à la fin de chaque question**. Dans le cas contraire, vous risquez d'avoir des classes qui ne fonctionnent pas pour la question en cours et qui ne fonctionnent plus, du fait des modifications apportées, pour les questions précédentes (tests générant des erreurs). La notation s'effectuera uniquement sur le code fonctionnel.

— Si, en fin de TP, votre code n'est pas entièrement fonctionnel (c'est-à-dire ne répondant à l'ensemble des questions du sujet), **vous fournirez, avec votre code source, un fichier texte indiquant quels sont les tests non fonctionnels**.

— Pour des raisons pratiques, liées à la correction, les classes n'appartiendront à **aucun package** (donc attention au respect de ce point, notamment pour les utilisateurs de Netbeans, qui crée, automatiquement, un package par projet, de déclarer les classes sans package de rattachement).

— Dans l'ensemble des classes développées tous les attributs devront être qualifiés **privés et non protégés**.

— Dernière recommandation : insérez les classes de test dans votre projet au fur et à mesure des tests demandés car sinon cela va provoquer des erreurs de compilation.

Il est recommandé de lire l'ensemble du sujet avant de débuter.

Les questions sont numérotées par section et il existe des dépendances c'est-à-dire que pour traiter une question, il peut être nécessaire d'avoir traité une autre question au préalable. Cela signifie également que s'il n'y a pas de dépendance alors la question peut être

traité indépendamment (par ex. ici la question 7 peut être traitée indépendamment de la question 6).

Les dépendances :
— question n° 1 : pas de dépendance ;
— question n° 2 : dépend de la question n° 1 ;
— question n° 3 : dépend de la question n° 1 ;
— question n° 4 : dépend de la question n° 3 ;
— question n° 5 : dépend des questions n° 1 ;
— question n° 6 : dépend des questions n° 1 et n° 3 ;
— question n° 7 : dépend de la question n° 1 et n° 3.

Enfin une classe `Utils` vous est fournie. Elle contient une méthode `creerDate` utilisée par les classes de test.

1 Soignants et patients...

On souhaite représenter un domaine qui est malheureusement toujours d'actualité, celui de la pandémie de COVID.

Pour débuter nous allons représenter deux types de personnes concernées par cet événement : les soignants et les patients.

Un soignant possédera les informations suivantes :
— un nom et un prénom ;
— une date de naissance (représentée par le type `java.time.LocalDate`) ;
— un numéro de professionnel de santé.

Un patient possédera les informations suivantes :
— un nom et un prénom ;
— une date de naissance (représentée par le type `java.time.LocalDate`) ;
— un numéro de dossier.

Travail à réaliser (n'implémenter que les méthodes accesseurs en lecture appelées dans la classe `TestPersonne` **en respectant les noms utilisés**) :

1. Définir une classe `Personne` factorisant les attributs communs aux soignants et aux patients et possédant trois méthodes uniquement :
   — une méthode `donneNom` retournant le nom de la personne ;
   — une méthode `donnePrenom` retournant le prénom de la personne ;
   — une méthode `donneDateNaissance` retournant la date de naissance de la personne.

   De cette classe, **ne représentant aucun objet concret**, aucune instance ne pourra être créée.
2. Puis définir les classes `Soignant` et `Patient`.

   La classe `Soignant` possédera :
   — un constructeur avec la signature suivante :
   ```
   public Soignant(String nom, String prenom, LocalDate dateNaissance, int numPS)
   ```
   La classe `java.time.LocalDate` sera utilisée pour représenter les dates car par rapport à la classe `java.util.Date`, elle offre plus de facilités pour calculer des périodes entre dates ou les comparer.
   — une méthode `donneNumPS` retournant le numéro de professionnel de santé.

   La classe `Patient` possédera :
   — un constructeur avec la signature suivante :
   ```
   public Patient(String nom, String prenom, LocalDate dateNaissance, int numDossier)
   ```
   — une méthode `donneNumDossier` retournant le numéro de dossier du patient.
3. Tester le code développé avec les trois classes de test `TestPersonne`, `TestSoignant` et `TestPatient` fournies.

2 Durée d'isolement...

Nous allons maintenant définir une durée d'isolement. Cette notion sera commune à la fois aux soignants et aux patients à travers la méthode `dureeIsolement` possédant la signature suivante :

```
public int dureeIsolement()
```

Cette méthode retournera le nombre de jours durant lequel une personne devra s'isoler si elle est infectée :

— ce nombre sera de 7 pour les soignants ;
— et 10 pour les patients.

Travail à réaliser :

1. Modifier les classes déjà développées pour ajouter cette nouvelle méthode.
2. Tester le code développé avec la classe `TestIsolement` fournie.

3 Dépistage...

Dans cette question, nous allons représenter de manière simplifiée (et erronée) le dépistage.

Pour notre représentation, nous adopterons deux simplifications (peu réalistes, heureusement !...) :

— une personne contaminée reste contaminée indéfiniment ;
— une personne contaminée devient immédiatement positive à un test.

Chaque personne pourra effectuer un test de dépistage (qui sera représenté par la classe `CovidTest` [1]). La classe `CovidTest` devra contenir les informations suivantes :

— une référence à la personne testée ;
— une référence au soignant qui a réalisé le test ;
— la date du test (`java.time.LocalDate`) ;
— le résultat du test (booléen).

Le résultat d'un test dépendra de l'état de la personne. Lors de la création d'une instance de `Personne` celle-ci sera considérée comme non infectée. Cet état initial (sain) pourra ensuite être modifié par une méthode `devientContamine` et un accesseur à cet état sera défini via une méthode `donneEtatContamination` :

```
public void devientContamine(LocalDate dateContamination)
public boolean donneEtatContamination(LocalDate dateCourante)
```

À ce stade, ces deux méthodes reçoivent une date en paramètre mais qui ne sera pas utilisée pour l'instant.

Ici on pourrait s'interroger sur l'utilité de faire des tests alors qu'il existe une méthode permettant de consulter l'état d'une personne. Néanmoins, comme dans notre modélisation on souhaite s'approcher de la réalité, ce sera le test (et seulement le test) qui permettra de connaître l'état de contamination d'une personne. Le résultat du test dépendra évidemment de la valeur booléenne retournée par la méthode `donneEtatContamination` de `Personne` : vrai la personne est contaminée, faux elle ne l'est pas. Ici on ne gère pas le fait qu'une personne contaminée redevient saine à l'issue d'une certaine durée.

L'association entre la classe `CovidTest` et la classe `Personne` devra prendre en compte le fait qu'une personne peut réaliser plusieurs tests. La création d'un test sera représentée par la méthode `seFaireDepister` :

1. Cette classe n'a pas été appelée `TestCovid` pour éviter la confusion avec les classes de test qui sont fournies et dont le nom débute par « `Test...` ».

```
public CovidTest seFaireDepister(Date dateDepistage, Soignant soignant)
```

Le second paramètre (`soignant`) fera référence au soignant qui aura réalisé le test.
L'accès aux tests d'une personne se fera par la méthode `donneDepistagesEffectues` :

```
public List<CovidTest> donneDepistagesEffectues()
```

Travail à réaliser :

1. Créer la classe `CovidTest` avec les accesseurs requis par les tests (cf. classe de test `TestDepistage`).
2. Modifier la classe `Personne` en définissant les méthodes citées précédemment :
   — `devientContamine`
   — `donneEtatContamination`
   — `seFaireDepister`
   — `donneDepistagesEffectues`
3. Tester le code développé avec la classe `TestDepistage` fournie.

4 Registre de tests COVID

À présent on souhaite que les tests soient enregistrés dans un registre de tests et qu'on puisse trier les tests enregistrés selon différents critères :
— par la date de réalisation (le plus ancien en premier) ;
— par le nom de la personne testée (par ordre alphabétique).
Pour chacun de ces critères, s'il y a égalité durant la comparaison, le second critère s'appliquera. Par exemple, si on trie des tests selon leur date et qu'il existe deux tests effectués à la même date alors ils seront triés selon le nom de la personne testée.

Travail à réaliser :

1. Créer une classe `RegistreCovid` avec deux méthodes :

   ```
   public void enregistrer(CovidTest test)
   public List<CovidTest> donneListePCR()
   ```

   la première permettant d'enregistrer un test dans le registre et la seconde permettant d'obtenir l'ensemble des tests enregistrés sous la forme d'une liste.
2. puis créer les classes `TriTestParDate` et `TriTestParNom` permettant d'utiliser la méthode `sort` de la classe `java.util.Collections` prenant une instance de la classe `java.lang.Comparator` en paramètre.
3. Tester le code développé avec la classe `TestRegistreCovid` fournie.

5 Vaccination...

On cherche maintenant à déterminer quel type de vaccin peut être administré à une personne.

Nous allons définir une interface `Vaccin` ayant la structure suivante :

```
import java.time.LocalDate;

public interface Vaccin {
    boolean condition(Personne p, LocalDate dateCourante);
}
```

Le paramètre `dateCourante` permet de déterminer la date à laquelle la condition est évaluée, elle sert notamment à pouvoir calculer l'âge de la personne `p` au moment où cette condition est évaluée.

Travail à réaliser :

1. Définir deux classes `VaccinARN` et `VaccinAttenue` implémentant l'interface `Vaccin`. Pour `VaccinARN` la condition sera vraie si la personne est âgée de moins de 55 ans (cf. classe de test fournie) et pour `VaccinAttenue` la condition sera toujours vraie (pas de limite d'âge).
2. Tester le code développé avec la classe `TestVaccin` fournie.

Pour le calcul d'âge vous pourrez utiliser ce fragment de code :

```
int age = java.time.Period.between(this.dateNaissance, dateCourante).getYears();
```

6 Type de tests COVID...

À la question 3 nous n'avons pas précisé le type de test réalisé. Nous allons ici l'indiquer sachant qu'il existe (au moins) 4 types de test :

— RT_PCR avec une fiabilité de 100 %;
— Antigénique avec une fiabilité de 71 %;
— Salivaire avec une fiabilité de 82 %;
— Sérologique avec une fiabilité de 90 %.

Vous pourrez constater que la moyenne de ces taux est de 85,75 % (cf. classe de test fournie).

Travail à réaliser :

1. Mettre en œuvre la notion de type de test en définissant une énumération nommée `TypeCovidTest` qui prendra en charge le taux de fiabilité de chaque type de test.
2. Modifier les classes `Personne` et `CovidTest` pour que la classe `TestTypeCovidTest` puisse se compiler en s'assurant que les classes de test données précédemment soient compatibles avec ces modifications (les tests qui s'exécutaient correctement précédemment ne doivent pas être mis en échec par vos modifications).
3. Tester le code développé avec la classe `TestTypeCovidTest` fournie.

7 État sanitaire...

Pour cette dernière question, nous allons considérer l'état sanitaire d'une personne.

Pour simplifier :

— l'état d'une personne sera « sain » si elle a effectué deux tests dont le résultat est négatif, le dernier datant de moins de 7 jours (par rapport à la date où on demande l'état) et il ne doit pas exister plus de 7 jours entre le dernier test et le test précédent;
— l'état d'une personne sera « contaminé » si elle a effectué un test dont le résultat est positif, ce test datant de moins de 14 jours (par rapport à la date où on demande l'état);
— dans tous les autres cas, l'état d'une personne sera considéré comme « inconnu ».

Travail à réaliser :

1. Mettre en œuvre la notion d'état sanitaire en définissant une énumération nommée `EtatSanitaire` qui définira les 3 états possibles d'une personne.
2. Ajouter une méthode `donneEtatSanitaire` à la classe `Personne` qui aura la signature suivante :

   ```
   public EtatSanitaire donneEtatSanitaire(Date dateDemandeEtat)
   ```

Cette méthode vérifiera les différentes conditions énumérées ci-dessus et renverra l'état correspondant.

3. Tester le code développé avec la classe `TestEtatSanitaire` fournie.