dubois Quentin
Mark: 19.26/20 (total score: 26/27)

+33/1/44+

QCM TEST

Introduction à la POO
13/01/2022

Nom et prénom : DUBOIS Quentin

Groupe : 2

Cochez les cases en mettant une ×. Le symbole ⊕ indique que la question peut avoir zéro, une ou plusieurs bonnes réponses. Pour ces questions :

- *cocher une bonne réponse apporte des points positifs*
- *ne pas cocher une mauvaise réponse apporte des points positifs*
- *cocher une mauvaise réponse peut apporter des points négatifs*
- *ne pas cocher une bonne réponse peut apporter des points négatifs*

Pour les question à une seule réponse :

- *cocher une bonne réponse apporte des points positifs*
- *cocher une mauvaise réponse peut apporter des points négatifs*
- *une réponse non-cochée apporte zéro points*

Dans tout le code, les package et les import sont censés être correctement déclarés. Toute classe est supposée être dans le bon package, dans le bon fichier, avec les bons import.

OMG Pandémie!

Vous devez compléter un programme en cours de développement simulant une pandémie. Dans ce jeu au tour par tour, certains perticipants peuvent infecter d'autres ! Les humains ne sont pas encore infectés, mais ce n'est qu'une question de temps pour que tous les humains attrapent une saleté. Le code fourni est divisé est cinq classes: le simulateur `Simulator`, et trois classes `Human`, `Covid` et `Flu` dérivant toutes d'une classe parente `Character`. *Flu* c'est la grippe, ceux qui l'ont attrapée sont des *grippés*. Un conseil: lisez toutes les questions avant de commencer. Gérez votre temps et ne paniquez pas, vous avez deux heures avant que les infectés ne débarquent. Le code fourni peut sembler long, mais vous n'avez besoin de le modifier qu'à des endroits spécifiques. Répondez simplement aux questions et ne faites pas de travail non demandé (il ne rapportera pas de points).

Question 1 ⊕ Que veut dire le mot clé `protected` pour une variable ou une méthode ?

- [x] Accessible depuis toute classe dérivée de la classe où elle est déclarée
- [x] Accessible depuis toute classe dans le même package que la classe où elle est déclarée
- [x] Accessible dans la classe où elle est déclarée
- [ ] Accessible depuis toute classe

1/1

Question 2

Dans la classe `Character`, pourquoi les variables `name` et `viralResistance` sont-elles `protected` ?

Soyez précis.

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☒6

2/2

Ces variables sont protected, cela permet de pouvoir être accédées par les classes héritières : Human, Covid et Flu

Question 3

Creez le constructor de la classe `Covid`. Au début du jeu, les covidés sont asympomatiques et ne sont pas contagieux (c'est-à-dire `Covid#isContagious == false`).

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☒6

2/2

```
Covid ( String name, int virtualResistance) {

    super (name, virtualResistance);
    this.isContagious = false;
}
```

Question 4 ⊕

Quels lignes sont affichées par le code ci-dessous ?

```
Character c1 = new Flu("Fluzie", 50);
c1.say("I am the fluzie!");
Flu c2 = (Flu) c1;
c2.say("No, it's me!");
```

☒ `Infect!`

1/1

☐ `I am the fluzie!`

☐ `No, it's me!`

Question 5

Avec c2 comme dans la question ci-dessus, quel sera le resultat de :

```
c2 instanceof Character
```

☒ true

1/1

☐ false

Question 6 ⊕

Avec c2 comme dans la question ci-dessus, indiquez les lignes qui compilent.

☒ `Character c1 = new Flu("Fluzie, 50);`

☐ `Flu f = c1;`

1/1

☐ `Flu f = (Character) c1;`

☒ `Flu f = (Flu) c1;`

☒ `Character c = c2;`

Question 7
Avec `c1` comme dans la question 4, quelle sera la sortie du code ci-dessous :

```
Character c = (Character) c1;
c.say("That's tricky!");
```

- [ ] `That's tricky!`
- [ ] `No, it's me!`
- [ ] `I am the fluzie!`
- [x] `Infect!`

1/1

Question 8
A la fin de chaque tour, chaque personnage doit executer le code écrit dans les méthodes `Human#endOfTurn`, `Covid#endOfTurn` ou `Flu#endOfTurn`. Malheureusement, le code actuel ne compile pas (le compilateur indique une erreur dans la dernière ligne de la méthode `Simulator#nextTurn`). Quelle modification (ou modifications) doit être apportée à la classe `Character` afin de résoudre le problème ? Ecrivez le code de la modification.

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☒6

2/2

La classe Character ne contient pas la méthode endOfTurn. Nous allons donc lui rajouter pour résoudre l'erreur de compilation dans la méthode Simulator # nextTurn

```
class Character {
    ⋮
        void endOfTurn () {}
}
```

(Pas abstract car cela correspond à la question 16)

Question 9

Dans la boucle de la methode `Simulator#nextTurn`, chaque personnage prend par surprise ("encounter") le personnage suivant dans la liste, via la methode `encounterCharacter`. Modifier le programme afin que, si le personnage `A` "rencontre" le personnage `B` (c'est à dire `A.encounterCharacter(B)`) :

- si `A` est un humain, il dit "Keep away!"
- si `A` est un covidé, il infecte `B` et réduit sa résistance au virus de 10 points
- si `A` est un grippé, alors :
  - si `B` est aussi un grippé, `A` ne fait rien
  - si `B` est un humain, `A` infecte `B` et réduit sa résistance 5 points
  - si `B` est un covidé, il y a 50% de chances que `A` infecte `B` et réduit sa résistance de 5 points (vous pouvez utiliser la methode statique `Simulator.generateRandomBoolean` pour generer une booléenne aleatoire)

Lorsque `A` infecte `B`, il s'excuse "`B`, I'm gonna infect you, sorry!" (remplacez `B` par le nom du malheureux), et la `vaccineResistance` de `B` diminue. Specifications pour cette question:

- vous n'êtes pas autorisés à ajouter une variable d'instance ou de classe, où que ce soit
- les seules classes que vous êtes autorisés à modifier sont `Human`, `Flu`, `Covid`
- vous êtes autorisés à ajouter des méthodes dans une ou plusieurs classes. Cependant, les seules signatures autorisées sont :
  `public void encounterCharacter(Character c)`
  `protected void infect(Character c)`

Le non-respect de ces spécifications strictes vaudra zéro pour cette question. Vous devriez chercher à éviter la duplication de code tout en respectant ces consignes imposées. Et avant de répondre à cette partie, vous devriez lire la question suivante...

☐0 ☐1 ☐2 ☐3 ☒4 ☐5 ☐6 1.33/2

```
class Human extends Character {
    @ Override
    void encounterCharacter ( Character c) {
        System.out.println ("Keep away !");
    }
}

class Covid extends Character {
    @ Override
    void encounterCharacter ( Character c ) {
        if ( c instanceof Human h ) {
            c.infect(h);
        }
        c.reduceVirtualResistance (10);
    }
}

class Flu extends Character {
    protected void infect ( Character c ) {
        c.reduceVirtualResistance (5);
        Say (c)
    }
```

Voir suite feuille annexe

Question 10
A présent, un nouveau type de personnage fait son apparition: le grippé mutant. Il est en tout point similaire au grippé classique, mais s'il infecte un autre alors il réduit de 25 points la résistance de celui-ci (au lieu de 5).
Implémenter la classe `FluMutation` représentant ce personnage, en gardant à l'esprit que vous faites de la programmation orientée objet (donc profitez de l'héritage pour réduire au maximum la taille de cette nouvelle classe!). Les lignes correspondantes dans le constructeur `Simulator` seront alors décommentées. D'autres modifications sont-elles nécessaires dans le programme pour prendre en compte ces nouveaux personnages ?

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☒6 2/2

```
class FluMutation extends Flu {
    FluMutation (String name, int virtualResistance) {
        super (name, virtualResistance);
    }

    @ Override
    protected void infect (Character c) {
        c.reduceVirtualResistance (25);
    } super.say (c)
}
```

Non, nous n'avons pas besoin de faire d'autres modifications.

Question 11 Après la première boucle de la méthode `Simulator#nextTurn`, tous les personnages dont la résistance au virus est ≤ 0 doivent être retirés du jeu. Ecrire à l'endroit indiqué le code qui retire ces personnages de la liste `Simulator#characterList`. En une expression si possible.

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☒6 2/2

```
characterList.stream ()
        .filter (c -> c.getVirtualResistance > 0)
        .collect (Collectors.toList ());
```

Question 12 Après la première série d'infections, les covidés peuvent aller infecter les humains pour les transformer à leur tour en covidés. Chaque covidé qui est susceptible d'infecter (c'est-à-dire avec `Covid#isContagious == true`) doit appeler la methode `Covid#infect` sur le premier humain de la liste qui n'a pas encore été infecté (c'est-à-dire `Human#hasBeenInfected == false`). Ecrire le code correspondant à l'endroit indiqué dans la méthode `Simulator#nextTurn`.

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☒6 2/2

```
for ( Character c1 : CharacterList ) {
    if ( c1 instanceof Covid cov && cov.getIsContagious() ) {
        for ( Character c2 : CharacterList ) {
            if ( c2 instanceof Human h && !h.hasBeenInfected() ) {
                cov.infect(h);
                break;
            }
        }
    }
}
```

Question 13 A l'issue de ces infections, tous les humains ayant été infectés doivent se transformer en covidés. Ces covidés nouvellement créés ont le même nom et la même résistance au virus que les humains correspondant au moment de l'infection, et sont contagieux dès leur création. Remplir le code de la méthode correspondante `Human#turnIntoCovid`, et mettre à jour la liste de personnages dans `Simulator#nextTurn` : les objets `Human` ayant été infectés doivent être remplacés par les nouveaux objets de type `Covid` issus de la transformation, à la même position dans `Simulator#characterList`.
Evidemment, aux tours suivants, ces nouveaux covidés pourront à leur tour infecter les humains restants !

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☒6 2/2

```
Covid turnIntoCovid() {
    Covid cov = new Covid( super.nom, super.virtualResistance );
    cov.setIsContagious( true );
    return cov;
}

int size = CharacterList.size();
for ( int i = 0; i < size; i++ ) {
    Character c = characterList.get(i);
    if ( c instanceof Human h && h.hasBeenInfected ) {
        characterList.set( i, h.turnIntoCovid() );
    }
}
```

Question 14 Présentez un exemple du code de l'application OMG Pandémie qui démontre le fonctionnement du *typage statique* et du *typage dynamique* et expliquez comment cela marche.

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☒6 2/2

Character c = new Elu (Eluzie, 50)
typage statique — typage dynamique

c.say (" ")
→ Résultat : " Eluzie says : Infect "
Utilisation de la méthode définie dans Elu correspondant au typage dynamique de l'objet.

Question 15 Quelles modifications au code seraient nécessaires avec ce changement de signature :
`class AllHumansInfectedException extends Exception`

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☒6 2/2

Il faudrait rajouter throws AllHumansInfectedException à la signature de la méthode main de Simulator. C'est le seul changement nécessaire car la méthode run() l'inclut déjà dans sa signature.

Question 16 Quelles modifications au code seraient nécessaires pour que la classe initiale `Character` soit `abstract` ?

☐0 ☐1 ☐2 ☐3 ☐4 ☒5 ☐6

1.67/2

Il faudrait mettre la méthode endOfTurn en abstract.
Ainsi la classe Character devient abstract.

* Plus rajouter le mot clé abstract dans la signature de la classe Character.

Signatures :

- abstract class Character
- abstract void endOfTurn();

Character

```
/**
 * Parent Character class
 */
class Character {
    protected final String name; // name of the character
    // represents the resistance to the virus
    // (once down to 0, this character snuffs it (dies) and will be removed)
    protected int viralResistance;

    /**
     * @param name name of the character
     * @param viralResistance initial resistance
     */
    Character(String name, int viralResistance) {
        this.name = name;
        this.viralResistance = viralResistance;
    }

    String getName() {
        return name;
    }

    int getViralResistance() {
        return viralResistance;
    }

    /**
     * Decrease the viral resistance by a certain amount.
     * Resistance cannot go below 0.
     * @param reduction number of HP to reduce
     */
    void reduceViralResistance(int reduction) {
        viralResistance -= reduction;
        viralResistance = viralResistance < 0 ? 0 : viralResistance;
    }

    /**
     * Output a character's saying to the screen
     * @param str what the character says
     */
    void say(String str) {
        System.out.println(name + " says: " + str);
    }

    /**
     * Method triggered when the character described
     * by the current object meets another character,
     * and does something to him (for example, infects).
     * @param c the other character that this character meets
     */
    void encounterCharacter(Character c) {
        // Default action: do nothing
        System.out.println(name + " meets " + c.name + " and does not infect!");
    }
}
```

Human

```
class Human extends Character {

    private boolean hasBeenInfected; // false, until contact with covid
    private int turnsSinceInfection; // infected humans lose resistance

    /**
     * At the beginning of the game, humans have max resistance.
     * @param name name of the character
     * @param viralResistance initial resistance
     */
    Human(String name, int viralResistance) {
        super(name, viralResistance);
        hasBeenInfected = false;
        turnsSinceInfection = 0;
    }

    boolean getHasBeenInfected() {
        return hasBeenInfected;
    }

    void setHasBeenInfected(boolean hasBeenInfected) {
        this.hasBeenInfected = hasBeenInfected;
    }

    /**
     * Method triggered on each character at the end of each turn.
     */
    void endOfTurn() {
        // Increment the number of turns
        turnsSinceInfection++;

        // Infected human loses resistance
        if (turnsSinceInfection > 3) {
            viralResistance -= 2;
        }
    }

    /**
     * Transform this human who has been infected, into a covid.
     * @return a new object of class Covid,
     * with the same name and viralResistance as this human;
     * the new covid is immediately contagious
     */
    Covid turnIntoCovid() {
        // ... add your code here (question 13) ...
    }
}
```

Covid

```
class Covid extends Character {

    private boolean isContagious;

    // ... add your constructor code here ... (question 3)

    boolean getIsContagious() {
        return isContagious;
    }

    void setIsContagious(boolean isContagious) {
        this.isContagious = isContagious;
    }

    /**
     * Method triggered on each character
     * at the end of each turn.
     */
    void endOfTurn() {
        // The covid has 50% chance of becoming contagious
        if (isContagious || Simulator.generateRandomBoolean()) {
            isContagious = true;
            say("I am contagious now!!");
        }
    }

    /**
     * Method called when a covid infects a human
     * @param h Human who gets infected by this covid
     */
    void infect(Human h) {
        // The human has no way to protect themself.
        // They get infected.
        h.setHasBeenInfected(true);
        say("I have infected you, " + h.getName() + "!");
    }
}
```

Flu

```
class Flu extends Character {
    /**
     * @param name name of the character
     * @param viralResistance initial resistance to virus
     */
    Flu(String name, int viralResistance) {
        super(name, viralResistance);
    }

    /**
     * Output a character's saying to the screen
     * @param str what the character says
     */
    void say(String str) {
        System.out.println(name + " says: Infect!");
    }

    /**
     * Method triggered on each character
     * at the end of each turn.
     */
    void endOfTurn() {
        // Do nothing.
    }
}
```

AllHumansInfectedException

```
class AllHumansInfectedException extends RuntimeException {
    AllHumansInfectedException(String msg) {
        super(msg);
    }
}
```

Simulator

```
class Simulator {
    // Default viral resistance (VR) for our characters
    private static final int VR_HUMANS = 100;
    private static final int VR_COVID = 150;
    private static final int VR_FLU = 30;

    // List of characters currently in the game
    private List<Character> characterList;

    Simulator() {
        // Create characters
        characterList = List.of(
            new Human("Human 1", VR_HUMANS),
            new Human("Human 2", VR_HUMANS),
            new Covid("Covid 1", VR_COVID),
            new Covid("Covid 2", VR_COVID),
            new Flu("Flu 1", VR_FLU)
        );

        // uncomment the following lines for question 10
        // characterList = new ArrayList<>(characterList); // OK this is a bit of a hack
        // characterList.add(new FluMutation("FluMutation 1", VR_FLU));
    }

    /**
     * @return the number of human characters currently in the game
     */
    int nbHumansAlive() {
        // Need to iterate through the list
        // of characters and count the number of humans
        int[] nbHumans = {0};
        characterList.forEach(c -> {
            if (c instanceof Human) {
                nbHumans[0]++;
            }
        });
        return nbHumans[0];
    }

    /**
     * Perform all game logic for next turn.
     */
    void nextTurn() {
        // Each character encounters the next character in the list (question 9)
        for (int i = 0; i < characterList.size(); ++i) {
            Character c = characterList.get(i);
            Character encountered = characterList.get((i+1) % (characterList.size()));
            c.encounterCharacter(encountered);
        }

        // Dead characters are removed from the
        // character list
        // ... add your code here (question 11) ...

        // Each covid (if contagious) infects
        // the first Human in the list who has
        // not been infected yet
        // ... add your code here (question 12) ...

        // Humans that have been infected become covids
        // ... add your code here (question 13) ...

        // Perform end-of-turn actions for all characters (question 8)
        characterList.forEach(c -> c.endOfTurn());
    }
```

Simulator cont'd

```
    public static void main(String[] args) {
        Simulator sim = new Simulator();
        sim.run();
    }

    private void run() throws AllHumansInfectedException {
        System.out.println("Game starts with " + nbHumansAlive() + " humans!");

        // Iterate until no human remains alive
        while (sim.nbHumansAlive() > 0) {
            sim.nextTurn();
        }
        throw new AllHumansInfectedException("All humans have snuffed it!");
    }

    /**
     * Generate a pseudo-random boolean.
     * @return pseudo-random boolean
     */
    static boolean generateRandomBoolean() {
        Random random = new Random();
        return random.nextBoolean();
    }
}
```

La documentation de `ArrayList` indique les méthodes `set`, `remove` et `removeIf`, que vous pouvez utiliser, ou pas :

set `Object set(int index, Object element)`
Replaces the element at the specified position in this list with the specified element.

Parameters :
`index` - index of element to replace.
`element` - element to be stored at the specified position.

Returns : the element previously at the specified position.

remove `Object remove(int index)`
Removes the element at the specified position in this list. Shifts any subsequent elements to the left (subtracts one from their indices).

Parameters : `index` - the index of the element to removed.

Returns : the element that was removed from the list.

removeIf `boolean removeIf(Predicate filter))`
Removes all of the elements of this collection that satisfy the given predicate.

Parameters : `filter` - a predicate which returns true for elements to be removed.

Returns : true if any elements were removed.