|  |  |
| --- | --- |
| **EXAMEN** | |
| Matière: **Programmation Orientée Objet** Documents : **Non autorisés** Enseignants: **Chelbi. S, Mhafdhi. H, Daghfous. N, Hedhli. J, Sghaier. I** Durée: **1h30**Classes : **DSI2, RSI2, SEM2** Date : **Jan 2020** | |

*Le devoir comporte* **3** *pages. Il vous est demandé d’apporter un soin particulier à la présentation de votre copie.*

**Exercice1 : Question de cours (3 points)**

**Question 1**

En considérant les classes définies ci-dessous, indiquer quelles instructions de code peuvent être insérées individuellement à la place du marqueur **//...à compléter...**afin que l’exécution du code produise l’affichage suivant :

***Student says good lecture!***

***Gamer says good game!***

class Student {

void print() {

System.out.println("Student says good lecture!");}

}

class Gamer extends Student {

void print()

{System.out.println("Gamer says good game!");

}}

class MyApp {

public static void main(String []args)

{

Student a = new Student();

**// A COMPLETER**

a.print();

b.print();

}

1. Gamer b = new Gamer();
2. Student b = new Student();
3. Gamer b = new Student();
4. Student b = new Gamer();
5. Gamer b = ((Gamer) new Student());
6. Student b = ((Gamer) new Student());

**Question 2**

En considérant les définitions de classes suivantes, quels morceaux de code insérés individuellement à la place du marqueur **// à compléter** définiront une utilisation valide de la méthode fly() dans la méthode takeATrip(). (3 réponses)

class NoMoreFuelException extends Exception {}

class Plane {

void fly(long distance) throws NoMoreFuelException {

//.. some code

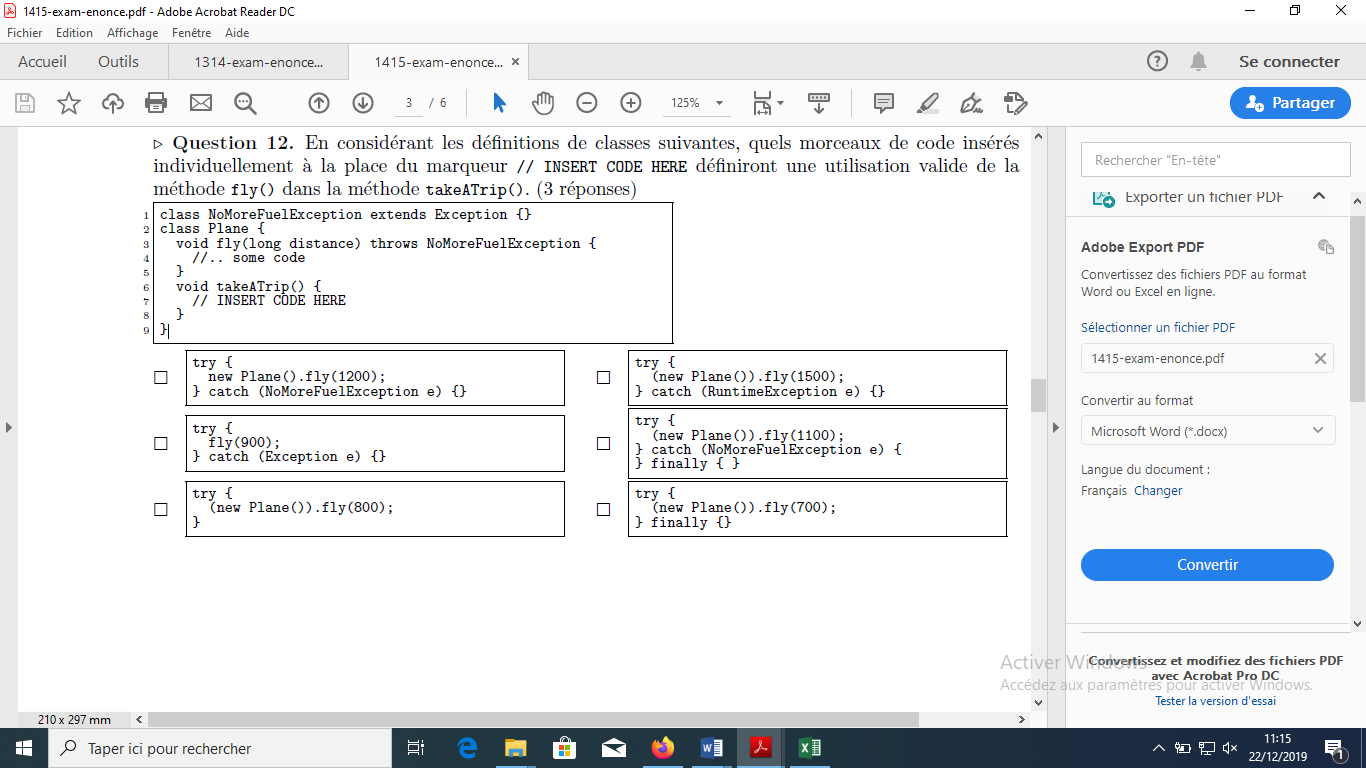
}

void takeATrip() {

**// A COMPLETER**

}

}

**Question 3**

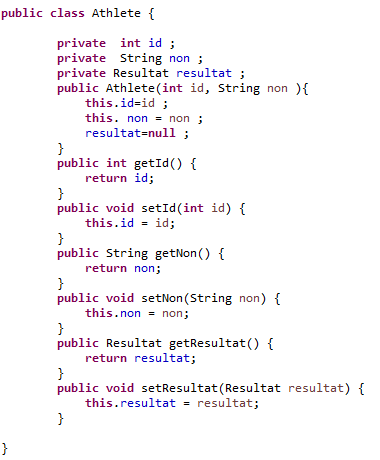
Indiquez les réponses correctes

1. Une classe est une instance d’un objet
2. Dans une méthode, il est possible d’accéder à l’instance courante à l’aide du mot-clé this.
3. Il est possible de déclarer des variables dans une interface si elles sont déclarées final
4. Il est impossible d’appeler un constructeur à partir d’un autre constructeur de la même classe
5. Il faut toujours définir un constructeur sans paramètre dans une classe.
6. Une classe mère ne peut pas définir de membres protected.
7. Une classe fille a accès à tous les membres de ses super-classes.
8. En Java, toutes les classes héritent de la classe Object.
9. Les variables statiques sont accessibles uniquement à partir de méthodes statiques.
10. Une classe abstraite ne peut hériter d’une autre classe abstraite.
11. Une classe abstraite doit définir au moins une méthode abstraite.
12. Une méthode abstraite peut être appelée dans la classe qui la définit.
13. Les exceptions sont des objets.

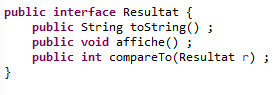
**Exercice 2 (17 points)**

Les organisateurs des jeux olympiques Tokyo 2020 souhaitent informatiser la gestion de cet événement. Dans ce sujet on s’intéresse en particulier aux épreuves et à la gestion de leurs résultats.

Les jeux olympiques réunissent des athlètes issus de très nombreux pays. On vous donne le code de la classe Athlète.



Les résultats des athlètes sont représentés par des objets du type Resultat. On vous donne l’interface résultat.



La méthode **toString** fournit le résultat sous forme de chaîne de caractères.

La méthode **affiche** l’affiche sur la sortie standard.

La méthode **compareTo()** compare le résultat courant avec celui passé en paramètre selon la nature des résultats. Elle retourne **0** s’ils sont égaux, **1** si le résultat courant est supérieur au résultat passé en paramètre et **-1** dans le cas contraire.

Selon les épreuves la nature du résultat change. Il peut se mesurer :

– par un temps, c’est par exemple le cas des courses, une telle donnée est modélisée par une instance de la classe ResultatTemps ;

– par une distance, c’est par exemple le cas des lancers ou des sauts, une telle donnée est modélisée par une instance de la classe ResultatDistance ;

– par un nombre de points, comme c’est le cas en plongeon, en tir ou au décathlon, une telle donnée est modélisée par une instance de la classe ResultatPoints ;

– par un rang dans un classement, comme c’est le cas au tennis de table ou pour les sports de combat, une telle donnée est modélisée par une instance de la classe ResultatRang.

Les quatre classes de résultats présentées ci-dessus **implémentent** l’interface **Resultat** donnée ci-dessus.

1. Donnez un code java pour la classe **ResultatDistance**. Un tel résultat est caractérisé par la **distance** réalisée par l’athlète, cette donnée est représentée par un **double**. Un tel résultat est meilleur qu’un autre si sa distance est plus grande
2. Donnez un code java pour la classe **ResultatRang**. Un tel résultat est caractérisé par le **rang** de classement de l’athlète, cette donnée est représentée par un **entier**. Un tel résultat est meilleur qu’un autre si son rang est inférieur.

Les épreuves sont modélisées par des instances de la classe **Epreuve**. Ces objets permettent de gérer les informations concernant les athlètes participant à l’épreuve et leur résultat à celle-ci. Les objets **Epreuve** ont une **dénomination** (exp :”100m haies féminin”, ”Tennis de table masculin”, etc.), ils permettent de gérer **un tableau des athlètes** participants à l’épreuve, sachant que chaque épreuve possède un *nombre maximum* de participants et un nombre effectif de participants. Une épreuve possède un **état** (boolean) permettant d’indiquer si elle est terminée ou non. Lorsque c’est le cas les résultats sont définitifs.

Les méthodes accessibles pour des épreuves sont :

* **Un** **constructeur** à deux paramètres **Epreuve (String denomination, int max)**
* **String** **toString()** qui permet de connaitre la dénomination et l’état (terminée ou non) de l’épreuve et les nom des athlètes participants
* **void ajoutAthlete(Athlete a)** qui permet l’ajout d’un nouvel athlète à l’ensemble d’athlètes participants à l’épreuve, si le nombre maximum n’est pas atteint.
* **void terminer ()** quipermet de terminer une épreuve
* **boolean estTerminee()** quipermet de vérifier si une épreuve est terminée ou non,
* **Resultat getRecordOlymique()** permet de connaître le meilleur résultat réalisé par les participant s à l’épreuve
* **Athlete getVinqueur()**permet de retourner l’athlète vainqueur  une fois l’épreuve est terminée
* **void** **fixeResultat(int id, Resultat r)** permet l’ajout ou la mise à jour d’un résultat pour un participant de l’épreuve (identifié par id). Rien ne se passe si le participant n’était pas inscrit à l’épreuve, une exception **IllegalStateException** est déclenchée si l’épreuve a été déclarée terminée,
* **Resultat getResultat(int id)** permet de connaître le résultat d’un athlète à l’épreuve(identifié par id), la valeur null est renvoyée si l’athlète ne participe pas à l’épreuve ou si aucun résultat n’a été ajouté pour lui.

1. Donnez un code java pour la classe **IllegalStateException**.
2. Donnez un code java pour la classe **Epreuve**

***Bon travail***