Espace Malraux - Place Saint Clair
BP 86
14203 Hérouville Saint-Clair Cedex

le cnam Normandie

CodeUE : UTC-503

Enseignant : Jean-Christophe HÉRON

Année universitaire 2021/2022

Sujet d'examen de *première* session

Date : 05 janvier 2022

Horaires : 13h30 - 15h30 (durée : 2h)

Préférences d'impression : *simple recto / couleur*

Modalités pratiques

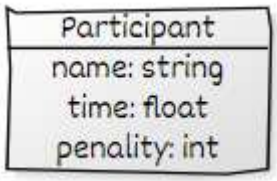
Éléments :	**autorisés**	**non autorisés**
Supports de cours (polycopiés)	x	
Documents manuscrits (prise de notes)	x	
Calculatrice		x
Autres : - Accès à internet		x
Autres consignes / remarques :		
- Tout appareil communicant ou de stockage de données (téléphone, tablette, ordinateur …) doit être éteint et rangé. - Sauf indication contraire, tout langage accepté, y compris pseudo-code		
Barème de notation :		
- Exercice 1 : 4 points - Exercice 2 : 4 points - Exercice 3 : 4 points - Exercice 4 : 4 points - Exercice 5 : 4 points		

Exercice 1 : Biathlon (4 points)

Dans le cadre de l'épreuve **20kms Individuel** du biathlon, les participants doivent effectuer 4 tirs (2 couchés et 2 debouts). Chaque cible manquée a un tir provoque une pénalité d'une minute (comptabilisée uniquement à l'arrivée).

Vous devez programmer une fonction qui lors de l'arrivée d'un participant, actualise le classement général provisoire de l'épreuve.

- Fonction **classement (Participant participant, Participant[] participantClasses) : Participant[]**
  - **Participant** est une structure de données définie de la façon suivante :



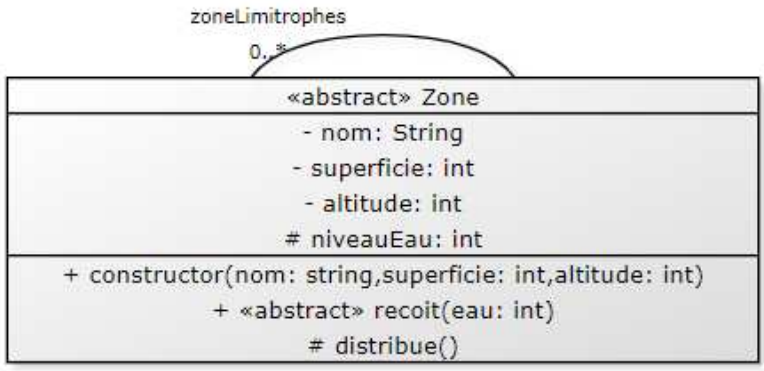
  - **Participant[] participantClasses** est le tableau des participants précédemment classés.
  - **time** est le temps total de parcours, sans tenir compte des pénalités.

a) Implémenter une version non optimisée de classement retournant le nouveau classement après l'arrivée du participant passé en paramètre. (le classement tiendra compte du temps de parcours (time), mis à jour avec les éventuelles pénalités).

b) Proposer des solutions d'optimisation de votre code.

Exercice 2 : Zones (4 points)

L'objectif d'un programme est de simuler les effets des précipitations au sol : le terrain est découpé en zones, chacune possèdant un nom, une superficie (en km²) et des zones voisines. Chaque zone posséde un niveau d'eau accumulée en surface (en mm), une méthode qui décrit la façon dont elle reçoit l'eau[1] (la pluie directe, ou l'eau des zones voisines), ayant en paramètre la quantité d'eau reçue en millimètres et enfin une méthode qui décrit comment elle distribue l'eau vers les zones voisines.



[1] s'il tombe x mm de pluie sur une surface de y km², la quantité d'eau totale reçue sur la zone, en m3 est x*y*1000

a) Implémenter les accesseurs en lecture et en écriture sur le membre **altitude**, en veillant à faire en sorte que l'altitude soit comprise entre **-10** et **1000 m**.
b) Implémenter le constructeur prenant en paramètre le nom, la superficie et l'altitude de la zone.
c) Expliquer pour quelles raisons la classe **Zone** est abstraite, et précisez les conséquences de ce choix.
d) Justifier le choix de mettre les membres **niveauEau** et **distribue**() en protégé (#) plutôt qu'en private (-).

Exercice 3 : Zones dérivées (4 points)

Il existe deux types de zones (pour simplifier) :

- les zones urbanisées n'absorbent pas du tout l'eau : quand elles reçoivent x millimètres de pluie, le niveau de l'eau sur la zone augmente de x.
- les zones naturelles ont une certaine capacité d'absorbtion de la pluie dans le sol (en m3). Quand une zone naturelle reçoit x millimètres de pluie, l'eau commence par remplir le sol, puis quand la capacité de stockage de la zone est atteinte, l'eau s'accumule alors à la surface et le niveau d'eau de la zone augmente.
  a) Représenter le diagramme de classes intégrant ces modifications.
  b) Implémenter le constructeur de la classe **ZoneNatuelle** initialisant ses membres, y compris sa capacité d'absorbtion.
  c) Implémenter la méthode **recoit(eau : int)** dans les classes dérivées.

Exercice 4 : Listes fonctionnelles (4 points)

a) Implémenter en ayant une approche procédurale les fonctions suivantes :
- Retourne le nombre d'éléments supérieurs à une certaine valeur, à partir d'une liste d'entiers passée en paramètre.

```
superieur(List<Integer> list, int val) : int
```

- Retourne le nombre d'éléments d'une liste égaux à une certaine valeur.

```
equals(List<Integer> list,int val) : int
```

b) Implémenter ces mêmes fonctionnalités en ayant une approche fonctionnelle, et en utilisant une fonction d'ordre supérieur **conditionalCount**, permettant de retourner le nombre d'éléments d'une liste satisfaisant une condition passée en paramètre, sous forme de callback.

Exercice 5 : Logique (4 points)

Soient les faits suivants :

- Adam aime les pommes et le couscous.
- Clara aime les carottes et les fraises tagada.
- Olivier aime les oranges et les carottes.
- Les pommes sont des fruits.
- Les oranges sont des fruits.
- Les carottes sont des légumes.
- Les fraises tagada sont des bonbons.
- Le couscous contient des carottes.
- Ceux qui aiment les fruits et les légumes sont en bonne santé.
- Ceux qui aiment les aliments à base de fruits et les aliments à base de légumes sont aussi en bonne santé.

a) Formalisez ces faits et règles en PROLOG.

b) Ecrivez la requête pour savoir qui est en bonne santé ?

c) Listez les fruits connus.

d) Déterminez qui aime les pommes ou les oranges.