EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communication
Cours Informatique II (SV)
Jamila Sam

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Série notée – Sujet 1

Instructions

– Max 1h15
– Max 80 points (dont 10 en bonus)

1. Cliquez sur le lien **série notée** se trouvant sous la rubrique **Séries d'exercices** de la page web du cours et suivez scrupuleusement les instructions qui apparaissent.

2. Vous pouvez ouvrir à nouveau le navigateur une fois ces manipulations faites, pour consulter la page web du cours.

3. Pour **RENDRE** la série notée, en fin d'examen :
   (a) Cliquez à nouveau sur le lien **série notée** et allez au point 3 des instructions. Le menu qui apparaît comporte déjà le nom du fichier à rendre.
   (b) Pour envoyer un fichier, cliquez sur le bouton `Browse` correspondant au nom du fichier et sélectionnez le fichier en question dans la liste qui vous est proposée. Cliquez ensuite sur le bouton `Envoyer`.
   Les fichiers suivants sont à envoyer, **même si non abordés** :

   predator.cc, dosage.cc

   Pour terminer, n'oubliez pas de **VALIDER VOTRE ENVOI**.

Règles et recommandations :

1. La série est à réaliser **individuellement**. L'échange d'informations est strictement interdit (y compris l'échange de documents).

2. La série doit impérativement être réalisée sur les terminaux des salles CO020-023 et sur votre propre compte. Il ne doit y avoir **qu'un seul login sur votre compte durant le temps de la série notée**.

3. Vous pouvez amener votre propre clavier, mais il vous appartient de tester avant la série que ce matériel fonctionne correctement.

4. Il n'est pas permis : d'utiliser du matériel électronique, d'envoyer des emails, de vous connecter sur une autre machine ou d'imprimer pendant la durée de la série notée.

5. Documents autorisés: les documents du cours (transparents, séries, corrigés). Il vous est possible de consulter cette documentation, et seulement celle-là, en ligne. 1–3 livres de référence sur le langage C++, au plus 100 feuilles de notes personnelles. En cas de doute, demandez l'avis de l'assistant responsable.

6. Il est **impératif de mettre vos numéros SCIPER** en commentaire au début de chacun de vos fichiers.

7. Vous commenterez les parties "délicates" de votre code (ceci peut-être fait en anglais).

8. Lisez attentivement chaque question de façon à ne faire que ce qui vous est demandé, et servez-vous de l'exemple de fonctionnement accompagnant les exercices pour vérifier votre solution. **Si l'énoncé ne vous paraît pas clair, ou si vous avez un doute, demandez des précisions à l'un des assistants.**

9. Si pour des raisons de langue, vous ne comprenez pas bien ce qui vous est demandé, n'hésitez pas à faire appel à un assistant pour obtenir des clarifications.

10. Sauf mention explicite, les affichages ne doivent pas obligatoirement correspondre à ceux donnés en exemple de fonctionnement; les mêmes informations doivent être affichées, mais le format peut être différent.

11. La série comporte 2 exercices indépendants.

12. Le fait que le code soit compilable ou exécutable compte dans la notation mais ne constitue pas le seul critère. La correction tiendra compte également des "solutions approchées".

SUITE À LA PAGE SUIVANTE

SUITE À LA PAGE SUIVANTE

Exercice 1 : Prédateurs et proies (70 points)

Nous nous intéressons dans cet exercice à modéliser de façon très basique un écosystème peuplé de *prédateurs* et de *proies*.

– ces derniers peuvent se déplacer et se rencontrer. Ils peuvent également tomber malades.

– les prédateurs mangent les proies dans certaines conditions.

Vous prendrez note des directives suivantes avant de commencer à coder :

1. Vous travaillerez dans le fichier `predator.cc` et **n'utiliserez pas d'autres fichiers**. Quelques éléments y sont déjà fournis :
   – quelques constantes;
   – une programme principal auquel votre code devra strictement se conformer. Ce programme ne doit pas être modifié d'une autre façon que ce que l'énoncé vous suggère.
2. Il n'est pas nécessaire d'externaliser les définitions des méthodes en dehors de la classe.
3. Vous implémenterez des méthodes `get` et `set` uniquement si elles s'avérent vraiment nécessaires pour le fonctionnement du programme tel que demandé.
4. Votre code sera proprement encapsulé.
5. Votre programme devra être bien *modularisé* et évitera la *duplication de code*.
6. Si vous le jugez utile, vous êtes libres d'ajouter d'autres méthodes à celles qui vous seront explicitement demandées.

Suite de l'énoncé à la page suivante ⟶

Voici les éléments devant être modélisés :

1) Prédateurs et proies (Héritage, polymorphisme, 46 points)

Il s'agit d'abord d'implémenter une classe `Animal`, permettant de représenter un animal (prédateur ou proie) évoluant dans l'écosystème.

La classe `Animal` (25 points) Un *animal* est caractérisé par :

- une *coordonnée en x* et une *coordonnée en y* permettant de repérer sa position dans l'espace (deux entiers);
- une information indiquant s'il est *malade*;
- une information indiquant s'il est *mort*;

La classe `Animal` comportera :

- un constructeur initialisant les coordonnées de l'animal au moyen de valeurs passées en paramètre. L'animal "construit" sera *sain* et *vivant*.
- les getters `get_x`, `get_y` retournant les coordonnées de l'animal;
- les méthodes méthodes `bool isSick()` et `bool isDead()` permettant de savoir si l'animal est malade ou mort;
- la méthode `distanceTo`, conforme au `main` fourni, et retournant la distance séparant l'animal à un autre animal. La distance entre deux points (x_1, y_1) et (x_2, y_2) se calcule selon la formule $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
- les méthodes `void up(unsigned int units)` (respectivement `void down(unsigned int units)`) permettant d'augmenter (respectivement de diminuer) de `units` la coordonnée en y;
- les méthodes `void right(unsigned int units)` (respectivement `void left(unsigned int units)`) permettant d'augmenter (respectivement de diminuer) de `units` la coordonnée en `x`;
- la méthode `void die()` permettant de faire mourir l'animal;
- la méthode `void setSick(bool)` permettant de rendre l'animal malade ou de le guérir;

Il vous est demandé d'implémenter la classe `Animal`, de manière à ce que les contraintes suivantes soient respectées :

1. les coordonnées de l'animal ne peuvent être inférieures à 0 ou supérieures aux constantes `MAX_X` et `MAX_Y`. Les valeurs que l'on tente de leur affecter à la construction ou via d'autre méthodes veilleront à faire plafonner ces coordonnées à ces valeurs;
2. La classe doit être bien encapsulée.

Les sous-classes `Predator` et `Prey` (21 points) Vous ferez ensuite en sorte que la classe `Animal` se spécialise en deux sous-classes: les prédateurs (classe `Predator`) et les proies (classe `Prey`).

La hiérarchie de classes sera dotée :
- de constructeurs conformes au `main` fourni;
- d'une méthode polymorphique `string getName() const` retournant le nom générique de l'animal: `"Animal"` dans le cas de la super-classe, `"Predateur"` pour les prédateurs et `"Proie"` pour les proies;
- de la surcharge de l'opérateur `<<` permettant d'afficher un animal en indiquant son nom générique, ses coordonnées et son état (mort ou sinon malade ou pas).

  Exemples d'affichage:

```
Predateur en (100,100): animal sain
ou
Predateur en  (100,100): animal malade
ou
Proie en  (100,100): animal mort
```

Pour tester le programme implémenté jusqu'ici, décommentez la partie du programme principal comprise entre `// TEST 1` et `// FIN TEST 1`.

La trace d'exécution pour cette partie devrait ressembler à ce qui suit :

```
Test de la partie 1 :
--------------------
Predateur en (0,0): animal sain
Proie en (25,0): animal sain
Distance : 25
Predateur en (100,100): animal sain
Proie en (0,100): animal sain
Predateur en (0,0): animal malade
Predateur en (0,100): animal malade
Predateur en (0,100): animal malade
Predateur en (0,100): animal mort
```

Suite de l'énoncé à la page suivante ⟶

2) Mauvaises rencontres (classe abstraite, polymorphisme, 24 points)

Pour simuler l'écosystème, on adopte les critères suivants :

1. deux animaux *se rencontrent* s'ils sont tous les deux en vie et que la distance les séparant est inférieure à `MEETING_RADIUS`;

2. un prédateur (sain ou malade) qui rencontre une proie la *mange* si elle est saine (elle meurt donc). Dans l'autre sens, une proie qui rencontre un prédateur se fait manger si elle est saine. Les proies malades ne se font pas manger.

Il vous est demandé de doter la hiérarchie d'une méthode `meet` virtuelle pure, qui gère l'éventuelle rencontre de deux animaux. Cette méthode va donc :
– tester s'il peuvent se rencontrer vu leur état et leur distance;
– gèrer les cas de prédation.

Vous respecterez les contraintes suivantes:

1. le prototype de la méthode `meet` sera conforme au `main` fourni (une fois la portion de code entre `// TEST 2` et `// FIN TEST 2` décommentée);

2. la méthode `meet` sera proprement modularisée : les critères ci-dessus pourront être mis en oeuvre par des méthodes utilitaires;

3. la hiérarchie de classes comportera une méthode polymorphique `bool isPrey() const` retournant vrai si l'animal est une proie et faux sinon. Cette méthode sera utilisée pour mettre en oeuvre les traitements décrits.

Pour tester le programme implémenté jusqu'ici, décommentez la partie du programme principal comprise entre `// TEST 2` et `// FIN TEST 2`.

La trace d'exécution pour cette partie devrait ressembler à ce qui suit :

```
Test de la partie 2:
--------------------
Proie saine, Predateur trop eloignes :
Proie en (0,0): animal sain
Predateur en (5,5): animal sain
apres tentative de rencontre :
Proie en (0,0): animal sain
Predateur en (5,5): animal sain

Proie malade, Predateur se rencontrent :
Proie en (0,0): animal malade
Predateur en (1,1): animal sain
apres rencontre  (dans les deux sens):
Proie en (0,0): animal malade
Predateur en (1,1): animal sain
Predateur en (1,1): animal sain
Proie en (0,0): animal malade

Proie saine, Predateur se rencontrent :
Proie en (0,0): animal sain
Predateur en (1,1): animal sain
apres rencontre :
Proie en (0,0): animal mort
```



```
Predateur en (1,1): animal sain

Predateur, Proie saine  se rencontrent :
Predateur en (1,1): animal sain
Proie en (2,2): animal sain
apres rencontre :
Predateur en (1,1): animal sain
Proie en (2,2): animal mort

Deux proies saines de rencontrent:
Proie en (20,20): animal sain
Proie en (18,18): animal sain
apres rencontre :
Proie en (20,20): animal sain
Proie en (18,18): animal sain

Deux predateurs se rencontrent:
Predateur en (20,20): animal sain
Predateur en (18,18): animal sain
apres rencontre :
Predateur en (20,20): animal sain
Predateur en (18,18): animal sain
```

Exercice 2 : savants dosages (10 points, bonus)

Compilez et exécutez le programme fourni dans le fichier `dosage.cc`. (Si le programme boucle indéfiniment, vous pouvez le stopper au moyen de `Ctrl-C`).

Il est censé afficher :

```
alcool 0.3
formol 0.4
```

mais il produit un résultat erroné.

1. Ajoutez au fichier `dosage.cc` un commentaire expliquant pourquoi.

2. Corrigez le programme fourni, sans toucher au programme principal