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

Faculté Informatique et Communication
Cours de programmation aux sections IN/SC
Sam J.

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

INTRODUCTION A LA PROGRAMMATION

Test Semestre I

Instructions :

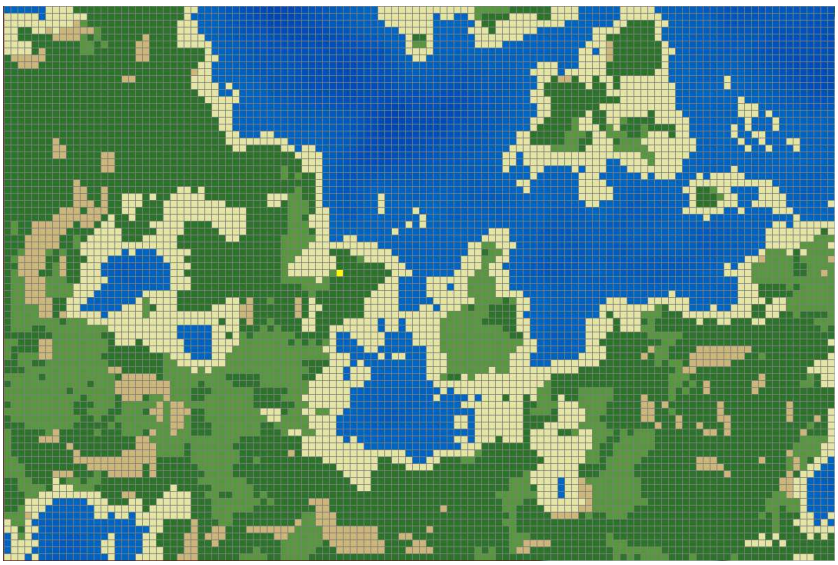
- Vous disposez de une heure quarante cinq minutes pour faire cet examen (8h15 - 10h);
- Nombre maximum de 110 points (dont environ 15 facultatifs);
- Toute documentation sur papier est autorisée;
- Veillez à **ne traiter *qu'un* exercice par feuille**, et à **indiquer votre numéro sciper** sur *chacune* des feuilles. Une feuille sans identification ne sera pas corrigée;
- L'examen compte 3 exercices. Ces exercices sont indépendants.
- Les exercices ne sont pas tous de même difficulté. Commencez par ceux dont vous maîtrisez le mieux les concepts impliqués.

SUJET A LA PAGE SUIVANTE

SUJET A LA PAGE SUIVANTE

Exercice 1 : Conception OO [52 points]

Des arbres de différentes sortes poussent sur un terrain. On souhaite anticiper par simulation graphique, la couverture arboricole qui peut se développer au cours du temps. Le terrain est rectangulaire. Il est quadrillé en cellules dont la nature va potentiellement influencer la croissance des arbres.



Une partie du programme, inspirée de votre second mini-projet, est fournie en annexe. Il s'agit :

- d'une modélisation minimale de la notion de terrain;
- d'un programme principal `Program`;
- d'une interface `Simulable`;
- et d'un type énuméré `Ground`.

Prenez connaissance de ce matériel avant de commencer l'exercice et ne négligez pas les commentaires qui y figurent.

Les cellules du terrain

On distingue des cellules: de terre siliceuse, de terre argileuse, de roche et d'eau.

Chaque cellule connait sa position sur le terrain (vous supposerez que vous disposez d'une classe `Vector`, analogue à celle de votre projet, et qui représente une paire de `int`). Une cellule peut avoir comme contenu éventuel l'arbre qui y pousse et l'on considère qu'il ne peut y avoir qu'un arbre par cellule.

Les cellules du terrain sont colorées en fonction de leur nature ou contenu (il y a une couleur particulière pour une cellule de sol argileux sans arbre, une couleur pour une cellule d'eau, une autre couleur encore si la cellule contient un résineux etc.). La taille des arbres joue aussi un rôle dans la détermination des couleurs. Les modalités précises du calcul des couleurs ne nous intéressent cependant pas.

Les arbres

Les arbres sont de différents types. Ils sont tous caractérisés par une *taille*, plafonnée à un certain maximum. Ce maximum est spécifique au type de l'arbre et ne peut pas être déterminé pour un arbre quelconque. Par exemple, la taille maximale des résineux est fixée à 10m et celle des feuillus à 15m (voir ci-dessous pour les catégories d'arbres). Un arbre peut avoir une taille non nulle à sa création.

Catégories d'arbres : les arbres sont soit des *résineux* soit des *feuillus*. Ils ne peuvent pas occuper une cellule contenant de l'eau, de la roche ou étant déjà occupée par un autre arbre.

S'ils occupent une cellule, ils poussent d'une façon qui leur est spécifique, dépendante du pas de temps `deltaTime` de la simulation et de la nature de la cellule. Les modalités exactes de cette croissance ne nous intéressent pas ici et l'on ne connaît pas les modalités de croissance d'un arbre quelconque.

Expansion de la couverture arboricole A chaque cycle de simulation (chaque appel à `update`) un arbre a une certaine probabilité d'essaimer (produire un arbre de même type) autour de lui.

Les résineux essaiment dans une cellule choisie au hasard dans un rayon autour de la position de la cellule qu'ils occupent. Les feuillus font de même, et en plus, ils vont avoir la même probabilité d'essaimer aussi sur n'importe quelle cellule du terrain. Les rayons sont spécifiques au type d'arbre et vous supposerez qu'ils sont calculés au moyen d'une méthode `getExpansionRadius()`. Le calcul des probabilités se feront au travers d'une méthode `boolean mustExpand()` retournant vrai si l'arbre doit essaimer et faux sinon. Les modalités d'implémentaion de `mustExpand` sont spécifiques au type d'arbre.

Lors de l'essaimage, si la cellule choisie pour une nouvelle pousse est une cellule d'eau ou de roche, ou si elle est déjà occupée, rien ne se passe. Sinon, un nouvel arbre du même type que celui qui a essaimé sera planté dans la cellule.

Vous considérerez enfin qu'un arbre connaît la cellule qu'il occupe et qu'une cellule connait le terrain auquel elle appartient. Le recours à des getters protégés tels que `getOwner()` est autorisé et vous considérerez que toutes les classes que vous proposerez appartiennent au même paquetage que la classe `Terrain`.

Question 1 : Conception [44 points]

1. On vous demande d'écrire la/les classe(s) permettant de compléter les éléments manquants à la classe `Terrain` et permettant de mettre en oeuvre les fonctionalités souhaitées; seules les entêtes des méthodes sont demandés. **On ne vous demande pas d'écrire de programme complet, ni le corps des méthodes, uniquement les attributs et les entêtes de méthodes**. Vous pouvez décrire les classes et leur contenu au moyen de diagrammes ou en pseudo-code Java.
2. les classes devront contenir les membres nécessaires pour mettre en oeuvre toutes les fonctionnalités souhaitées, **qu'elles soient explicitement demandées ou suggérées par les spécifications de l'énoncé** ;
3. Pour les méthodes pour lesquelles ce n'est pas clair **vous noterez au moyen d'un bref commentaire la signification du type de retour et ce qu'elles font dans les grandes lignes**.
4. **Ne négligez ni les constructeurs, ni les droits d'accès et les modificateurs**.

Les contraintes à respecter sont :

1. le droit `protected` ne doit pas être utilisé pour les attributs;
2. votre conception ne doit pas nécessiter de dupliquer du code;
3. vous supposerez que tout l'outillage nécessaire au graphisme est disponible ainsi que les classes fournies en annexe et la classe `Vector` permettant de modéliser une paire de coordonnées;
4. vous ne proposerez de getter/setter que lorsqu'ils sont nécessaires à la mise en oeuvre de la simulation décrite;
5. il n'est pas nécessaire d'écrire des directives d'importation.

Question 2 : Programmation [8 points]

Au vu de votre conception, donnez le corps de la méthode permettant aux feuillus d'essaimer. On suppose que les nouvelles pousses d'arbre ont la taille 1.0 à la création.

Suite au verso ☞

Exercice 2 : Concepts [36 points]

Répondez clairement et succinctement aux questions suivantes :

1. **[6 points]** Soit le code suivant :

```
0.  class A {
1.      private int a = 1;
2.
3.      public A() {
4.          System.out.println("A: " + a);
5.      }
6.
7.      public A (int a) {
8.          this();
9.          this.a = a;
10.     }
11.
12. }
13.
14. class B extends A {
15.     private int a;
16.
17.     public B(int a) {
18.         super(a);
19.         System.out.println("B: " + this.a);
20.
21.     }
22.     public B() {
23.         this.a = 4;
24.         System.out.println("B: " + a);
25.
26.     }
27. }
28. class P {
29.
30.     public static void main (String[] args) {
31.         A a = new A(2);
32.         B b1 = new B(2);
33.         B b2 = new B();
34.     }
35. }
```

   (a) Qu'affiche t'il ? Justifiez brièvement.

   (b) A la ligne 19, peut-on remplacer `this.a` par `a` sans corrompre la compilation ? Justifiez brièvement.

   (c) Peut-on ajouter l'instruction `this()` juste avant la ligne 23. Justifiez brièvement.

   (d) Peut-on inverser les lignes 8 et 9 ? Justifiez brièvement.

2. **[6 points]** Qu'affiche le code suivant ? Justifiez brièvement votre réponse.

```
0.  import java.util.ArrayList;
1.  import java.util.List;
2.
3.  class P {
4.      public static void f(List<Integer> list) {
5.          for (Integer i : list) {
6.              System.out.print(i + " " );
7.          }
8.          System.out.println();
9.      }
10.
11.     public static void g(List<Integer> list) {
12.         for (Integer i : list) {
13.             ++i;
14.         }
15.         list.remove(0);
16.     }
17.
18.     public static void h (List<Integer> list) {
19.         List<Integer> other = new ArrayList<Integer>();
20.
21.         for (Integer i : list) {
22.             other.add(i+2);
23.         }
24.         list = other;
25.     }
26.
27.     public static void main(String[] args) {
28.         List<Integer> list = new ArrayList<Integer>();
29.         list.add(1);
30.         list.add(5);
31.         list.add(22);
32.
33.         f(list);
34.         g(list);
35.         f(list);
36.         h(list);
37.         f(list);
38.     }
39. }
```

Suite au verso ☞

3. [**7 points**] Soit le programme suivant :

```
0.  class A {
1.      public final static A I = new A(2);
2.      private int a;
3.
4.      private A(int a) {
5.          this.a = a;
6.      }
7.
8.      public A() {
9.          this.a =  1;
10.     }
11.
12.     @Override
13.     public String toString() {
14.         return "value " + a;
15.     }
16.
17. }
18.
19. class S {
20.     A a = new A();
21.
22.     public static void main(String[] args) {
23.
24.     }
25. }
```

Pour chaque cas suivant, indiquez si l'on peut ajouter la ligne de code correspondante (et seulement cette ligne) dans la méthode main du programme principal sans corrompre la compilation. Justifiez brièvement dans chaque cas.

(a) `a = new A(2);`

(b) `A.I = new A(3);`

(c) `A.I = new A();`

(d) `System.out.println(a);`

(e) `System.out.println(A.I);`

(f) `System.out.println(new A().I);`

(g) `System.out.println(A.a);`

4. [**4 points**] Dans votre second mini-projet, le programme principal comportait une ligne de ce type :

   `Game game = new BikeGame();`

   Pourquoi n'avons nous pas écrit à la place :

   `ActorGame game = new BikeGame();`

5. [**7 points**] Soit la portion de programme suivante :

```
0.  import java.util.Scanner;
1.
2.  class Exception2 {
3.    private static Scanner clavier = new Scanner(System.in);
4.
5.    public static void main(String[] args) {
6.      String a = clavier.nextLine();
7.      String b = clavier.nextLine();
8.
9.      try {
10.       System.out.print("** ");
11.       System.out.println(f(a,b) + " **");
12.       System.out.println("Voila !");
13.     }
14.     catch (Exception err) {
15.       System.out.println(err.getMessage());
16.     }
17.     finally {
18.       System.out.println("C'est parti !");
19.     }
20.   }
21.
22.   public static int f(String x, String y) throws Exception {
23.     if (x.length() < y.length()) {
24.       throw new Exception("Non !");
25.     }
26.
27.     int z = 0;
28.     for (int i = 0; i + y.length() < x.length(); ++i) {
29.       if (x.substring(i, i+y.length()).equals(y)) {
30.         System.out.print(i + " ");
31.         ++z;
32.       }
33.     }
34.     System.out.println();
35.     return z;
35.   }
36. }
```

Qu'affiche t-il si l'on entre « HoHoHoHoHoHo » puis « oH » ?
Et si l'on entre « oH » puis « HoHoHoHoHoHo » ? Justifiez brièvement vos deux réponses.

Indications :
— L'appel `s.substring(int begin, int end)` retourne la sous-chaîne de `s` comprise entre `begin` (inclus) et `end` (non inclus).
— Les lignes 6 et 7 lisent bien l'une après l'autre les deux chaînes mentionnées (il n'y a pas de piège avec des retours de lignes non consommés par la lecture).

Suite au verso ☞

6. [**6 points**] Supposons que l'on ait écrit le code suivant dans une classe `Truc` :

```
abstract int f();
```

Pour chacune des propositions suivantes, indiquez si elle est correcte ou non et justifiez à chaque fois brièvement votre réponse.

(a) La méthode `f` retourne toujours zéro.

(b) Il n'est pas possible de déclarer une variable de type `Truc`.

(c) Il n'est pas possible d'instancier une variable de type `Truc`.

(d) Il n'est pas possible de redéfinir la méthode `f` dans les sous-classes de `Truc`.

(e) `Truc` doit avoir plusieurs sous-classes.

(f) La classe `Truc` doit être déclarée `abstract`.

Exercice 3 : Déroulement de programme [22 points]

Le programme suivant compile et s'exécute sans erreurs.

```
0.  interface Builder {
1.    abstract void expand(Kingdom k);
2.  };
3.
4.  abstract class Kingdom {
5.    private int size; // surface du royaume
6.    private int money; // richesse
7.
8.    public abstract void launch(Builder b);
9.
10.    public Kingdom(int aSize,
11.                   int someMoney) {
12.      size = aSize;
13.      money = someMoney;
14.    }
15.
16.    public void grow(int aSize,
17.                     int someMoney) {
18.      size += aSize;
19.      money += someMoney;
20.    }
21.
22.    public String toString () {
23.      return ("Size: "  + size  + "\n" +
24.              "Money: " + money + "\n");
25.    }
26. }
27.
28. class Alfheim extends Kingdom {
29.    public Alfheim(int aSize,
30.                   int someMoney) {
31.      super(aSize, someMoney);
32.    }
33.
34.    public String toString() {
35.      return ("* Royaume des Elfes *\n" +
36.              super.toString());
37.    }
38.
39.    public void launch(Builder b) {
40.      b.expand(this);
41.    }
42. }
```

```
43.
44. class Nidavel extends Kingdom {
45.    public Nidavel(int aSize,
46.                   int someMoney) {
47.      super(aSize, someMoney);
48.    }
49.
50.    public String toString() {
51.      return ("* Royaume des Nains *\n" +
52.              super.toString());
53.    }
54.
55.    public void grow(int aSize,
56.                     int someMoney) {
57.      super.grow(2*aSize, 2*someMoney);
58.    }
59.
60.    public void launch(Builder b) {
62.      b.expand(this);
63.    }
64. }
65.
66. class CityBuilder implements Builder {
67.    public void expand(Kingdom kingdom) {
68.      kingdom.grow(10, 20);
69.    }
70. }
71.
72. class KingdomExpansion {
73.    public static void main(String[] args) {
74.      Kingdom elves  = new Alfheim(20, 30);
75.      Kingdom dwarfs = new Nidavel(30, 10);
76.      System.out.println(elves);
77.      System.out.println(dwarfs);
78.
79.      System.out.println("-----");
80.      Builder cb = new CityBuilder();
81.      elves.launch(cb);
82.      dwarfs.launch(cb);
83.
84.      System.out.println(elves);
85.      System.out.println(dwarfs);
86.    }
87. }
```

Qu'affiche t-il ? Expliquez succinctement sont déroulement. **Il ne s'agit pas ici de paraphraser le code, mais bien d'*expliquer* les étapes et le déroulement du programme.**