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 130 points (dont environ 25 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 4 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 À LA PAGE SUIVANTE

Exercice 1 : Programmation [12 points]

Donnez le code d'une méthode

```
private static int embedInXthBit(int value, boolean m, int i)
```

permettant de cacher une valeur dans le `i`ème bit de l'entier `value`. La valeur cachée sera 0 (zéro) si le booléen `m` vaut `false` et 1 sinon.
Vous considérerez que le bit zéro est le bit le moins significatif: lorsque `i` vaut zéro, la valeur est cachée dans le bit le moins significatif, lorsqu'il vaut 1, dans le bit précédant le moins significatif etc.
Vous considérerez les valeurs des paramètres comme correctes.

Suite au verso ☞

Exercice 2 : Conception OO [64 points]

L'architecture que vous avez mise en oeuvre dans votre second mini-projet est donnée en annexe dans les grandes lignes, ainsi qu'un code de base possible pour les `Actor`, `Heart`, `Signal` et `Simulator`.

Il vous est demandé d'étendre cette architecture pour incorporer le fait que :

1. d'autres acteurs que le `Player` puissent se déplacer et interagir avec l'environnement (« acteurs dotés de vie »);
2. le `Player` puisse collecter des objets utiles en les stockant dans un *inventaire* d'objets.

Partie 1 : Acteur dotés de vie

On suppose que tous les acteurs dotés de vie ont aussi un niveau de vie et un niveau de vie maximal, qu'ils sont également mortels et qu'ils sont par défaut impactés de la même manière que le `Player` par les dégâts.

Question (partie 1) [5 points] :

1. Comment proposeriez vous d'étendre/modifier l'architecture existante pour permettre de modéliser ces nouveaux acteurs ?

Donnez simplement une explication en français de ce qui doit être créé ou déplacé pour obtenir la nouvelle conception.

Partie 2 : Inventaire

On souhaite modéliser le fait que des objets peuvent être «collectés» par le `Player`. Certains des objets collectés sont stockables dans son inventaire, mais pas tous. Le `Heart` par exemple redonne des forces au `Player` mais ne sera pas stocké : il disparaît quand le `Player` passe dessus et réapparaît spontanément au bout d'un laps de temps.

Un objet «collectable» a donc simplement la propriété de pouvoir devenir indisponible, soit parce que le joueur l'a mis dans son inventaire, soit parce qu'il est passé dessus. Lorsqu'il est indisponible, il disparaît du monde (c'est à dire qu'il n'interagit plus et n'est plus dessiné).
Deux nouveaux types d'objets collectables peuvent être stockés dans l'inventaire du `Player` :

1. des pièces de monnaies;
2. et des diamants.

Ces objets sont stockés à l'unité (une pièce de monnaie à la fois, un diamant à la fois etc.).

On suppose que chaque objet stockable dans un inventaire a une valeur marchande. (par exemple chaque pièce vaut 5 et chaque diamant 50) et qu'il peut subir un effet en quittant l'inventaire.

Le joueur peut librement collecter de pièces de monnaies, simplement en passant dessus. Elles seront alors stockées dans son inventaire.

Les diamants ne peuvent être collectés qu'en échange d'objets stockés dans l'inventaire. Si un diamant a la valeur X et que le `Player` a dans son inventaire les objets $o_1 \dots o_n$ dont la somme des valeurs vaut au moins X, il prendra le diamant et relâchera les objets $o_1 \dots o_n$. Ces derniers reprennent alors la position qu'ils occupaient à l'origine (et disparaîtront alors bien sûr de l'inventaire). On suppose que ce troc se fait selon un algorithme intelligent et non par le biais d'interactions via le clavier ou la souris.

Les diamants font office de *signaux* qui s'activent lors de leur acquisition : l'acquisition d'un diamant particulier peut ainsi permettre d'ouvrir une porte donnée ou débloquer un ascenseur donné, par exemple.

Un contrôle supplémentaire doit permettre de visualiser le contenu de l'inventaire du `Player`. Presser sur `'I'` par exemple dessinera tous les objets de l'inventaire par dessus le `Player`.

Questions (partie 2) [59 points] :

1. [**31 points**] Si l'on impose que chaque objet «collectable», qu'il soit stockable dans un inventaire ou pas, doit disposer d'une méthode :

   ```
   boolean collected(Player other) {
    // retourne vrai si l'objet peut être collecté par le Player
       }
   ```

   Comment proposeriez vous de modéliser les pièces de monnaie et les diamants (voir plus bas comment il vous est demandé de répondre)?
2. [**3 points**] Cette modélisation implique t-elles des retouches à la classe `Heart` ? si oui lesquelles ?
3. [**2 points**] Quelle méthode dans votre conception rendrait un objet disponible lorsqu'il quitte l'inventaire ?
4. [**9 points**] Sachant que la notion d'inventaire doit exister en tant que telle (on peut imaginer d'introduire plus tard des inventaires pour d'autres acteurs dotés de vie), comment proposez vous de modéliser et utiliser l'inventaire du joueur ?
5. [**8 points**] Comment proposez-vous de coder les méthodes `collected` impliquées dans votre conception ?
6. [**3 points**] Dans quelle méthode de quelle classe pensez-vous coder la gestion de la touche permettant l'affichage de l'inventaire ?
7. [**3 points**] La simulation devrait être stoppée pendant que l'inventaire s'affiche (les acteurs ne devraient plus continuer à se mouvoir pendant ce moment). Comment proposeriez vous de mettre en oeuvre cela? Quelles fonctionnalités devraient être ajoutés et à quelles classes ?

Pour les questions 1 et 4, indiquez les membres (attributs et entêtes de méthodes) de toutes les classes nécessaires ainsi que leur rôle en commentaire. Indiquez aussi ce qui doit être ajouté aux classes existantes et explicitez les liens d'héritage/implémentation. Les constructeurs ne seront pas oubliés et vous indiquerez ce qui est abstrait ou final.

Pour la question 5, donnez le code Java de chaque méthode `collected`.

Pour les autres questions, donnez une explication en français.

Suite au verso ☞

Exercice 3 : Concepts [33 points]

Répondez clairement et succinctement aux questions suivantes :

1. **[8 points]** Soit le code suivant :

```
0.  class A {
1.      private int a = 2;
2.      public A() { System.out.println("A:" + a); }
3.  }
4.  class B extends A {
5.      private int b;
6.      public B() { this(10); }

7.      public B(int val) {
8.          b = val;
9.          System.out.println("B:" + b);
10.     }

11.     public B(B autreB) { b = autreB.b; }
12. }

13. class Program {
14.     public static void main(String[] args) {
15.         B b1 = new B();
16.         B b2 = new B(b1);
17.     }
18. }
```

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

   (b) Peut-on ajouter l'instruction `super()` avant l'instruction `this(10)` dans la ligne 6 ? Justifiez brièvement.

   (c) Qu'affiche le programme si on remplace `this(10)` par `super(10)` dans la ligne 6 ? Justifiez brièvement.

2. **[6 points]** Soit le programme suivant :

```
interface I1 {
   default void m1() {}
}

interface I2 {
   default void m1() {}
}

class C implements I1, I2 {}
```

   (a) Pourquoi ne compile t'il pas ? Justifiez brièvement.

   (b) Compile t'il si l'on remplace le contenu de chacune des interfaces par `void m1();` ? Justifiez brièvement.

   (c) Quel est le minimum de code à ajouter à la classe `C` pour que le programme fourni ci-dessus compile ?

3. Que signifie la règle « déclarer ou traiter » dans la gestion des exceptions ? pourquoi ne s'applique t'elle pas aux `RuntimeException` ? Y a t'il d'autre types d'erreurs analogues dans l'API de Java ?

4. [**8 points**] Soit la portion de programme suivante :

```
1.  import java.util.List;
2.  import java.util.ArrayList;

3.  class A {
4.      public void m() {
5.          System.out.println("A");
6.      }

7.      public A(A other) {}
8.      public A() {}
9.  }

10. class B extends A {
11.     @Override
12.     public void m() {
13.         System.out.println("B");
14.     }

15.     public B() {}
16.     public B(B other) {}
17. }

18. class Program {
19.     public static void main(String[] args) {
20.         List<A> list = new ArrayList<A>();
21.         list.add(new A(new B()));
22.         list.get(0).m();
23.  }
24. }
```

(a) Pourquoi ce code ne compile t'il pas si l'on supprime le constructeur par défaut de `A` ?

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

(c) Qu'affiche t'il si l'on remplace la ligne 21 par :

```
list.add(new B());
```

? Justifiez brièvement.

Suite au verso ☞

5. [**6 points**] Soit le code suivant :

```
class A {
    private String a1;
    private int a2;
    public A(String s, int i) {
        a1 = s;
        a2 = i;
    }

    public String getA1() {
        return a1;
    }
    public int getA2() {
        return a2;
    }
}
```

(a) La classe `A` est-elle mutable ? Justifiez brièvement.

(b) Contient-elle des failles d'encapsulation ? Justifiez brièvement.

(c) Même question que la précédente si l'on remplace partout `int` par `Integer` et le constructeur de `A` par :

```
public A(String s, Integer i) {
   a1 = s;
   a2 = i;
}
```

6. [**5 points**] Soit le programme suivant :

```
1.  import java.util.List;
2.  import java.util.ArrayList;

3.  abstract class A {
4.  }

5.  class B extends A {
6.      void m1(List<A> list) {
7.          for (A a : list) {
8.              if (a instanceof B) {
9.                  a.m2();
10.             }
11.         }
12.     }

13.     void m2() {
14.     }
15. }

16. class Program {
17.     public static void main(String[] args) {
18.         B b = new B();
19.         b.m1(new ArrayList<B>());
20.     }
21. }
```

(a) Pourquoi a t-on le droit d'écrire `A a = new B();` alors que `A` est non instanciable ?

(b) Quelle erreur le compilateur va t'il signaler dans la classe `B` ? Comment la corriger ?

(c) Quelle erreur le compilateur va t'il signaler dans le programme principal ?

Suite au verso ☞

Exercice 4 : Gestion des exceptions [21 points]

```
1.  import java.util.Scanner;
2.  import java.util.InputMismatchException;

3.  class OperatorException extends Exception {
4.      public String getMessage() {
5.          return "invalid operator";
6.      }
7.  }

8.  class Program {
9.      private final static Scanner clavier = new Scanner(System.in);

10.     static int calculate(int oper1, int oper2, char op)
11.     throws OperatorException {
12.       // affiche le résultat de l'évaluation de oper1 op oper2 si tout se passe bien
13.       // Fait un throw(new ArithmeticException("division by zero"))
14.       // s'il y a une division par zéro
15.       // Fait un throw(new OperatorException())
16.       //si l'opérateur ne vaut pas :+, -, * ou /
17.     }

18.     static int parse(String number) {
19.        // convertit le string number en un nombre
20.        // fait un throw(new InputMismatchException("invalid number")
21.        //si number représente autre chose qu'un entier
22.     }

23.     public static void main(String[] args) {
24.         System.out.println("Entrez un entier, un operateur, et un entier");
25.         String number1 = clavier.next();
26.         char op = clavier.next().charAt(0);
27.         String number2 = clavier.next();
28.         int operand1 = 0;
29.         int operand2 = 0;
30.         try {
31.             operand1 = parse(number1);
32.             operand2 = parse(number2);
33.         } catch (InputMismatchException e) {
34.             System.out.println(e.getMessage());
35.             return;
36.         }
37.         try {
38.             int result = calculate(operand1, operand2, op);
39.             System.out.println(result);
40.         } catch (ArithmeticException e) {
41.             System.out.println(e.getMessage());
42.         } catch(OperatorException e) {
43.             System.out.println(op + " est un " + e.getMessage());
44.         }
45.     }
46. } // fin de la classe Test
```

Examinez le programme de la page précédente :

1. Qu'affiche ce programme si l'utilisateur entre :
   (a) z98 / 0
   (b) 98 / 0
   (c) sjhu * 45
   (d) 123 % 18
2. Le codage de `OperatorException` est-il conforme aux conventions usuelles ? Justifiez votre réponse
3. Que se passe-t-il si l'on fait hériter `OperatorException` de `ArithmeticException` au lieu de `Exception` ? Comment y remédier en gardant le lien d'héritage entre `OperatorException` et `ArithmeticException` ?
4. Quelles modifications devraient être apportées au programme principal (vous indiquerez quelles lignes de code ajouter/modifier et à quel(s) endroit(s)) si l'on veut que :
   — le programme ne s'arrête pas lorsqu'un des opérandes est invalide;
   — que dans ce cas il le remplace par zéro et qu'il indique à l'utilisateur ce qui a été fait;
   Exemple de déroulement:

```
Entrez un entier, un opérateur,  et un entier:
9 * kjdlk
deuxième  opérande invalide remplacé par zéro
résultat : 0
Entrez un entier, un opérateur,  et un entier:
kjdlk + 2
premier  opérande invalide remplacé par zéro
résultat : 0
Entrez un nombre, un opérateur et un nombre:
2 / élasdalk
deuxième  opérande invalide remplacé par zéro
division par 0!
```