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

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

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

INTRODUCTION A LA POO

Test Semestre I : corrigé

Note :

Ce document donne dans les grandes lignes le types de réponses attendues pour les 6 exercices du test.

Exercice 1 : Concepts [35 points]

Répondez clairement et succinctement aux questions suivantes :

1. **[5 points]** Les deux constructeurs de la classe `B` utilisent le constructeur sans argument de la classe `A` : le premier y fait appel explicitement par l'instruction `super()`, le second (constructeur de copie) implicitement. La classe `A` définit explicitement un constructeur. Le constructeur sans argument (défini par défaut) disparait de ce fait. Les deux erreurs sont signalées en raison de l'appel à ce constructeur inexistant dans chacun des deux constructeurs de la classe `B`.

2. **[4 points]** Un lien d'héritage est mis en place à mauvais escient : un musicien n'est pas une partition! Ici il serait plus censé d'établir une relation de type "A-UN" entre les deux classe (un muscien a une partition). La classe Partition aurait donc pour attribut un objet de type `Partition`.

3. **[6 points]** Le programme affiche :

   ```
   1.5 2.5 3.0
   4.0 4.0 4.0
   ```

   Les références sont passées par valeur. C'est une copie de la référence `dblArray1` qui est modifiée dans la première ligne de la méthode `m`. `dblArray1` reste inchangé après l'exécution de `m`. Le contenu des objets référencé par `dblArray2` peut lui être modifié.

4. **[5 points]**
   Les lignes 1 et 2 ne compilent pas (tentative de modification d'un attribut `Final`). La ligne 3 compile. En Java on peut modifier l'objet référencé par un attribut final.

5. **[4 points]** Expliquez pourquoi le programme suivant ne compile pas :

   ```
   class MyPerfectString extends String
   {
   }
   ```

   La classe `String` est finale en Java. On ne peut pas en faire dériver de sous-classes. L'intérêt d'interdire ce genre de tournures : fixer la sémantique de la notion de chaîne de caractères (très utilisée).

6. **[4 points]** `A` implémente l'interface `I` sans être abstraite ni redéfinir la méthode `boolean isI(char C)`. On ne peut instantier un objet de type interface.

7. **[4 points]**

   ```
   103
   103
   201
   ```

   Justification: `unA.a` et `A.a` réfèrent à l'attribut statique `a` de la classe `A` (initialisé à 100 dans le constructeur). Ce dernier est incrémenté une fois lors de l'appel `unA.m()` et deux fois lors de l'appel à `unB.m()` (une fois par l'appel `super.m()` et une autre fois par `++super.a`). Comme il s'agit d'un

attribut statique, c'est toujours la même zone mémoire qui est incrémentée d'où le `103`. `unB.a` réfère à l'attribut `a` (non statique) de `B` (initialisé à 200 dans le constructeur). En fait une instance de `B` dispose d'un attribut statique `a` hérité de `A` et d'un attribut non statique `a` qui lui est spécifique. `unB.a` est incrémenté une fois lors de l'appel `unB.m()`.

8. – $O(n^k)$
   – $O(n)$
   – $O(n!)$

Exercice 2 : Conception d'algorithme et programmation [25 points]

```
import java.lang.Math;

public class BigPrime {
   //Finds the largest prime smaller or equal than its first argument.
   public static void main(String[] args) {
      int bound = stringToInt(args[0]);
      findBigPrime(bound);
   }

   private static void findBigPrime(int bound) {
      if (bound < 0) {
         System.out.println("Vous avez fourni un argument
                             strictement inferieur a 1.
                             Il n'y a aucun nombre premier inferieur a 1");
         return;
      }

      for (int n = bound; n > 2; n--) {
         if (isPrime(n)) {
            System.out.println("Le plus grand nombre premier plus petit
                                ou egal a " + bound + " est " + n);
            return;
         }
      }
   }
```



```
   private static int stringToInt(String s) {
      int i = 0;
      try {
         i = Integer.parseInt(s.trim());
      } catch (Exception e) {
         System.out.println("Erreur: la String n'est pas reconnue
                             comme un entier");
         System.exit(0);
      }
      return i;
   }

   /* Primality test. Uses the fact the a number n is prime if and only if
    * it has no divisor strictly greater than 1 and smaller or equal
    * than sqrt(n).
    */
   private static boolean isPrime(int n) {
      int sqrt = (int)Math.sqrt(n);
      for (int i = 2; i <= sqrt; i++) {
         if (divides(i, n)) {
            return false;
         }
      }
      return true;
   }

   private static boolean divides(int m, int n) {
       return ((n % m) == 0);
      }
}
```

Exercice 3 : Conception OO et programmation [40 points]

Pour la partie conception, seuls les classes, attributs et entêtes de méthodes étaient demandés

Pour la partie programmationle corps des méthodes `valider` et `content` (ou équivalent) étaient demandé.

```
// Le Gala et sa table d'invités
class Gala {

   Invite[] table;

   public Gala() {
      table = new Invite[10];
   }

   public void ajouteInvite(Invite inv, int index) {
      table[index] = inv;
   }


   public boolean valider() {
      for (int i = 0; i < table.length; i++) {
         if(table[i] == null) return false;
      }

      for(int i = 1; i < table.length - 1; i++) {
         if(!table[i].content(table[i-1], table[i+1])) {
            return false;
         }
      }
      if(!table[0].content(table[1], table[table.length-1])) {
         return false;
      }
      if(!table[table.length-1].content(table[table.length-2], table[0])) {
         return false;
      }

      return true;
   }
}
```



```
// hiérarchie d'invités
abstract class Invite {
   private String nom;
   private boolean milliardaire;
   private String parti;
   public Invite(String n, boolean m, String p) {
      nom = n;
      milliardaire = m;
      parti = p;
   }


   public String getParti() {
      return parti;
   }

   public String getNom() {
      return nom;
   }

   public boolean isMilliardaire() {
      return milliardaire;
   }

   public abstract boolean content(Invite gauche, Invite droit);

}

class Politicien extends Invite{
   public Politicien(String n, boolean m, String p) {
      super(n, m, p);
   }

   public boolean content(Invite gauche, Invite droit) {
      return (gauche.getParti()).equals(getParti()) &&
             (droit.getParti()).equals(getParti());
   }
}
```


Suite du code sur la prochaine feuille.

```
class Vedette extends Invite {
   String[] pasAime;

   public Vedette(String n, boolean m, String p, String[] pA) {
      super(n, m, p);
      pasAime = pA;
   }

   public boolean content(Invite gauche, Invite droit) {
      for (int i = 0; i < pasAime.length; i++) {
         if (pasAime[i] == gauche.getNom() || pasAime[i] == droit.getNom())
            return false;
      }

      return true;
   }
}

class VIP extends Vedette {

   public VIP(String n, boolean m, String p, String[] pA) {
      super(n, m, p, pA);
   }

   public boolean content(Invite gauche, Invite droit) {
      return super.content(gauche, droit) &&
         (gauche.isMilliardaire() || droit.isMilliardaire());
   }

}
```


Exercice 4 : Correction de programme [15 points]

1. Le programme affiche :

   ```
   La boutique contient les articles:
   Peluche (20.0CHF -20.0%)
   Peluche (20.0CHF -20.0%)
   Peluche (20.0CHF -20.0%)
   ```

2. Ceci est dû au fait que la méthode `ajoute` ajoute plusieurs fois la même référence au stock. Le programme principal donné ajoutait trois fois la même référence au stock. Lorsque l'entrée 1 du tableau est réduite, c'est toutes les entrées du tableau qui se sont trouvées réduites aussi.
3. La solution acceptée pour cet exrcice consiste à faire en sorte que la méthode `ajoute` ajoute au stock une *copie* de l'objet :

   ```
   articles[nbArticles+i] = new Article(art.getNom(), art.getPrix());
   ```

   ou

   ```
   articles[nbArticles+i] = new Article(art); // constructeur de copie
   ```


Cette solution est acceptée à ce stade de votre apprentissage. La bonne solution dans l'absolu est de faire en sorte que la classe Article soit immutable (notion vue en deuxième année).

Exercice 5 : Analyse de programme OO [20 points]

1. Non. La classe `Unite` devrait être abstraite car elle ne ne correspond à aucune unité concrète du jeu.
2. On peut éviter la duplication en transformant l'interface `Nain` en une classe abstraite et en y déclarant les membres `taille`, `hache` et `void frappHache()`. On procède de façon analogue pour l'interface `Elfe`.
3. Non car en Java on ne peut pas hériter de deux classes.
4. `clone()`, car un constructeur de copie n'est pas résolu dynamiquement. `hache` et `arc` pourraient nécessiter une copie profonde car se sont des références (en supposant que l'on veuille garantir que chaque nain ait sa propre hache et chaque elfe son propre arc, ce qui semble ici être la solution la plus naturelle).

Exercice 6 : Récursion, complexité [15 points]

1. (a) Le programme affiche :

   ```
   -
   1
   0
   2
   3
   0
   1
   0
   0
   1
   0
   2
   3
   ```

   (b) Il affiche le signe du nombre sur la première ligne s'il est négatif et ensuite, sur des lignes séparées, chaque chiffre composant le nombre (depuis le plus significatif jusqu'au moins significatif)
2. (a) Le programme calcule la suite de `Fibonacci` récursivement

   (b) Le calcul de la complexité de ce programme est donné dans les transparents 33-38 du cours 11 (http://cowww.epfl.ch/proginfo/wwwhiver/documents/transparents11-algo-complexite.pdf)