IFT1020 - Session Été, Final - Solutionnaire

Mohamed Lokbani

IFT1020 – FINAL - Solutionnaire

Inscrivez tout de suite votre nom et code permanent.

Nom: _____| Prénom(s): _____|

Signature: _____| Code perm: _____|

Date : 28 juillet 2005

Durée: 3 heures (de 17h30 à 20h30) Local: 1360 du Pavillon A.-AISENSTADT.

Directives:

- Toute documentation permise.
- Calculatrice **non** permise.

- Répondre directement sur le questionnaire.
- Les réponses **doivent être brèves, précises et clairement présentées.**

1._____/22 (1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7)

2._____/12 (2.1, 2.2, 2.3)

3._____/12 (3.1, 3.2, 3.3, 3.4)

4._____/16 (4.1, 4.2, 4.3, 4.4)

5._____/13 (5.1, 5.2, 5.3, 5.4)

6._____/25 (6.1, 6.2, 6.3, 6.4)

Total:_____/100

Directives officielles

* Interdiction de toute communication verbale pendant l'examen.

* Interdiction de quitter la salle pendant la première heure.

* L'étudiant qui doit s'absenter après la première heure remet sa carte d'étudiant au surveillant, l'absence ne devant pas dépasser 5 minutes.

* Un seul étudiant à la fois peut quitter la salle.

* Tout plagiat, copiage ou fraude constitue une infraction et l'étudiant qui commet ce délit se voit attribuer la note F par le professeur.

F.A.S

Question 1 (22 points)

1.1 Dans Java, si une exception est de la catégorie « Checked », ceci signifie **qu'elle doit être traitée localement sinon elle doit être propagée en utilisant « throws »**
……..……

1.2 Quand la méthode « repaint() » est appelée dans une « Frame », ceci a comme incidence que la « Frame » est tout de suite redessinée (VRAI/FAUX) …**FAUX**…. car **« repaint() » est juste une requête pour que la « Frame » soit redessinée. Cette requête peut-être traitée avec un léger délai.**
……

1.3 Donner deux exemples d'un « Layout Manager » en Java : 1-… **FlowLayout** , 2-… **BorderLayout**. (Aussi **GridLayout, GridbagLayout, BoxLayout**)

1.4 Vous voulez réaliser une application de suivi de rendez-vous (calendrier électronique) pour votre agenda électronique. Cette application va vérifier en permanence l'état de vos rendez-vous, en vous alertant s'il y a lieu que vous avez des choses à faire. Parmi les collections suivantes, laquelle est la plus appropriée pour représenter les données de votre application de manière efficace: « **Liste, Table de hachage, Pile, Queue, Queue Prioritaire** » et dites pourquoi.

Les tâches seront organisées de telle manière qu'elles seront exécutées en fonction de leur importance, fixée préalablement. La collection « Queue Prioritaire » est la plus appropriée pour ce genre d'application.

1.5 Vous voulez construire une application de fouille pour collecter des adresses de courriel. Cette application va scanner votre disque dur à la recherche de toutes les adresses de courriel se trouvant sur votre disque. Par la suite elle va se charger d'envoyer les courriels recueillis à une tierce personne.
Parmi les collections suivantes « **List, Hashtable, HashSet, Array, Map** », laquelle est la plus appropriée pour contenir les données collectées et dites pourquoi.

1- On ne veut préserver que des données (courriels).
2- On ne veut pas que ces données soient en double.

La collection « HashSet » est la plus appropriée pour contenir les données collectées.

1.6 Après compilation du fichier « **`examen.java`** », parmi les instructions suivantes, laquelle va permettre de « l'exécuter » et pourquoi (expliquez aussi pourquoi les autres possibilités sont incorrectes)

a) **`java Examen`**
b) **`javac examen`**
c) **`java examen.java`**
d) **`java examen`**
e) **`java examen.main()`**

La réponse correcte est: d

a) incorrecte: Java fait la distinction entre majuscule et minuscule.

b) incorrecte: la commande « javac » est utilisée pour la compilation mais pas pour l'exécution d'un programme.

c) incorrecte: le programme doit exécuter un « .class » et non pas « .java » (code source).

d) correcte: « examen.java » étant le nom du programme source. Ainsi « examen.class » sera le nom du programme après compilation. Pour exécuter ce programme, il faut donc utiliser le nom du fichier sans extension avec la commande java.

e) incorrecte: le programme va chercher la classe examen dans le paquetage class, si nous faisons abstraction de l'emplacement des ().

1.7 Expliquez l'intérêt de définir la méthode « **`toString`** » dans une classe.

Chaque classe même sans l'utilisation de l'héritage dérive en réalité de la classe: « Object ». Dans cette classe, il est défini une fonction qui convertie les champs de X vers String si nécessaire.

En réalité si le mécanisme de conversion n'est pas la, tout ce que cette fonction va réaliser c'est d'afficher le type de l'objet qui fait l'appel et son emplacement mémoire (en réalité dans une table de hachage: une table spécialisée contenant des clés. Il est associé à chaque objet une clé).

Il est conseillé de définir dans chaque classe que vous allez définir, une méthode « toString » pour masquer celle par défaut et afficher par conséquence des informations que vous jugez plus utiles, que celles fournies par la méthode par défaut.

Question 2 (12 points) Soit le programme suivant qui **compile correctement**

```
public class Q2 {
 public static void main(String[] args){
   if( args.length > 0 ) {
     convert( args[0] );
     System.out.print("Bon ");
   }
   else {
     System.out.print("Mauvais ");
   }
   System.out.println( nombre );
 }
 static float nombre = 0 ;
 static boolean convert( String s ){
   try {
     nombre = Float.parseFloat( s );
     return true ;
   }catch (NumberFormatException e ){
     nombre = Float.NaN ;
     return false ;
   }
 }
}
```

Tout en justifiant votre réponse, dites ce qui sera affiché en sortie si le programme est exécuté avec les commandes suivantes :

2.1) **`java Q2`**

`Mauvais 0.0`

On ne passe le test « if(args.length>0) » car le nombre d'arguments est égal à 0! La méthode « convert(args[0]) » ne sera pas exécutée.

On se contente d'afficher « Mauvais » puis la valeur de la variable « nombre » qui est au départ « 0 ». Mais comme on affiche un Float, la valeur affichée est « 0.0 ».

2.2) **`java Q2 1-2`**

`Bon NaN`

On passe le test « if(args.length>0) » car le nombre d'arguments est supérieur à 0 vu la présence de la valeur « 1-2 ».
La méthode « convert(args[0]) » sera exécutée avec comme argument « 1-2 ». La conversion de l'argument « String-Float » fera lever l'exception « NumberFormatException » car « 1-2 » n'est pas un « Float ». On affiche en conséquence en sortie « NaN ».

2.3) **`java Q2 7.5 1-2`**

`Bon 7.5`

On passe le test « if(args.length>0) » car le nombre d'arguments est supérieur à 0 vu la présence de la valeur « 7.2 ».
La méthode « convert(args[0]) » sera exécutée avec comme argument « 7.5 ». La conversion de l'argument « String-Float » se fera correctement. Cet argument sera par la suite affiché en sortie.
On ne traite pas le second argument, qu'il soit correct ou incorrect! Il n'est jamais utilisé dans le programme.

Question 3 (12 points) « **DataVitales** » est le nom d'une classe faisant partie d'une série de classes que vous avez rassemblées dans le paquetage « Unis ». Ce paquetage sera utilisé par d'autres programmeurs dans votre compagnie. Vous ne voulez pas que ces programmeurs puissent étendre par héritage la classe « **DataVitales** ». Par ailleurs, vous voulez autoriser aussi l'accès à « **DataVitales** » aux classes se trouvant dans d'autres paquetages. **Comment doit-on déclarer cette classe?**

Traiter chaque sous question **à part de manière indépendante**. Les explications données pour une sous question ne doivent pas dépendre des réponses données dans les autres sous questions.

3.1) `protected final class DataVitales`

[ ] Correct [X] Incorrect

Pourquoi

On ne peut pas utiliser « protected » au niveau des classes. « protected » est utilisé qu'au niveau des membres de la classe.

3.2) `public final class DataVitales`

[X] Correct [ ] Incorrect

Pourquoi

L'utilisation de « final » va interdire l'utilisation de « extends » donc l'héritage. Le fait que la classe soit « public » va permettre aux classes se trouvant dans d'autres paquetages d'accéder à la classe « DataVitales ».

3.3) `public abstract class DataVitales`

[ ] Correct [X] Incorrect

Pourquoi

L'utilisation de « abstract » suppose qu'on va utiliser « extends ». Le fait qu'elle soit « public » l'accès est permis à tout le monde. Donc tout le monde peut « altérer » le contenu de cette classe, c'est justement ce que nous ne voulons pas permettre.

3.4) `public static class DataVitales`

[ ] Correct [X] Incorrect

Pourquoi

On ne peut pas utiliser « static » au niveau des classes primaires. Il est utilisé uniquement au niveau des variables, méthodes ou classes internes.

Question 4 (16 points) « **C1** » et « **C2** » sont des références à deux collections. Pour chacune des instructions suivantes, les affirmations associées (signalées en commentaire) sont-elles correctes?

4.1) *C1.retainAll(C2);* **// Elle ne modifie pas le contenu de « C1 »?**

[ ] Oui [X] Non

Pourquoi

La méthode « retainAll » ne garde dans « C1 » que les éléments se trouvant aussi dans « C2 », que les éléments communs. Donc elle peut modifier le contenu de « C1 ».

4.2) *C1.removeAll(C2);* **// Elle ne modifie pas le contenu de « C1 »?**

[ ] Oui [X] Non

Pourquoi

Le méthode « removeAll » retire de « C1 » tous les éléments se trouvant aussi dans « C2 ». Tous les éléments communs seront retirés. Donc elle peut modifier le contenu de « C1 ».

4.3) Pour: *C2.retainAll(C1); C1.containsAll(C2);* **// La 2e instruction va retourner « true »?**

[X] Oui [ ] Non

Pourquoi

La première instruction permet de garder dans C2 que les éléments communs. Forcement le test « C1.containsAll(C2) » va retourner « true » puisque après la première instruction, les contenus de « C1 » et « C2 » sont identiques.

4.4) Pour: *C2.addAll(C1); C1.retainAll(C2);* **// La 2e instruction n'aura aucun effet sur « C1 »?**

[X] Oui [ ] Non

Pourquoi

La première instruction va faire de telle sorte que « C2 » va contenir aussi les éléments de « C1 ». Les éléments de « C1 » sont communs aussi à « C2 ». L'instruction « C1.retainAll(C2) » ne va garder que les éléments communs à « C1 » et « C2 » et qui est comment entre les deux? C'est « C1 ». Donc la 2e instruction n'a aucun effet sur « C1 ».

Question 5 (13 points) Soit une description partielle d'une hiérarchie d'exceptions qui pourront être levées durant des opérations E/S sur des fichiers.

```
Exception
    +----IOException
              +-----FileNotFoundException
```

Soit la méthode « **X** » qui est supposée ouvrir un fichier à partir de son nom pour lire son contenu. Sachant que la méthode « **X** » n'a aucun bloc « **try-catch** », dites si les options suivantes sont correctes ou incorrectes, tout en expliquant votre choix de réponse.

5.1) La méthode « **X** » doit être déclarée comme levant l'exception « **IOException** » ou bien « **Exception** ».

[X] Correct [ ] Incorrect

Pourquoi

Ouverture de fichier et lecture des données donc exception associée est « IOException ». Vu la hiérarchie d'héritage, elle peut lever aussi « Exception »

5.2) La méthode « **X** » doit être déclarée comme levant l'exception « **FileNotFoundException** ».

[ ] Correct [X] Incorrect

Pourquoi

À cause de l'héritage. Ici on ne précise que le cas d'un fichier ne pouvant être lu parce qu'il serait introuvable. Mais quand est-il du cas où la lecture poserait problème?

5.3) N'importe quelle méthode faisant appel à la méthode « **X** » doit utiliser « **try-catch** » pour spécifiquement capturer l'exception « **FileNotFoundException** ».

[ ] Correct [X] Incorrect

Pourquoi

Il y a une alternative à l'utilisation d'un « try-catch ». Par exemple, la méthode qui fait l'appel peut être déclarée comme levant l'exception « IOException », elle passe ainsi la responsabilité de traiter cette exception à la chaîne d'appel.

5.4) Aucune précaution n'est nécessaire lors de la déclaration ou l'utilisation de la méthode « **X** ».

[ ] Correct [X] Incorrect

Pourquoi

L'exception « IOException » appartient à la catégorie « Checked ». Elle doit donc être déclarée et capturée.

Question 6 (25 points) On suppose pour cette question que la classe « **Item** » a été définie dans le fichier « **Question6.java** » comme suit :

```
class Item {
       String nom;
       int CodeBarre;
       public Item(String x, int y){nom=x;CodeBarre=y;}
       // + d'autres membres (attributs, méthodes, constructeurs etc.)
}
```

Les informations relatives aux items ne sont pas nécessairement uniques, sauf le code barre. En effet, à chaque item est associé un code barre unique représenté dans la classe « **Item** » par « **CodBarre** ». Vous pouvez supposer que deux items (instance de « **Item** »), ayant le même code barre, ont forcément le même nom. Par contre, deux items ayant le même nom, peuvent avoir des codes barre différents.

Pour la suite de cet exercice, chaque sous question dépend des précédentes

6.1) Écrire le code de la méthode « **equals** » associée à la classe « **Item** ».

```
public boolean equals(Object o) {
    if (!(o instanceof Item))
        return false;
    return ( ((Item) o).CodeBarre == CodeBarre);
}
```

6.2) Quelque part dans la méthode « **main** » se trouvant dans la classe « **Question6** », il y a ce fragment de code :

```
Set MesItems = new TreeSet();
Item A = new Item("a",1);
Item B = new Item("b",2);
MesItems.add(A);
MesItems.add(B);
```

Pour que le précédent fragment de code puisse s'exécuter correctement :

a) Énumérer les changements, si nécessaire, que vous devez apporter à la classe « **Item** ».

« TreeSet » supposent que ses éléments peuvent être triés suivant une relation d'ordre. De ce faite, vous avez besoin d'implémenter l'interface « Comparable », sinon une exception du type « ClassCastException » sera levée lors de l'exécution du programme au moment de l'insertion d'un second item dans « MesItems » car un tri va avoir lieu suivant une relation d'ordre non définie!

b) Est-ce que la classe « **Item** » doit avoir une méthode additionnelle (autre que « **equals** »)? Si oui donner la signature d'une telle méthode?

```
public int compareTo(Object)
```

c) Si vous avez répondu « **oui** » à la précédente question, écrire le code de cette méthode.

```
public int compareTo(Object o) throws ClassCastException {
    if (o instanceof Item)
        return CodeBarre - ((Item)o).CodeBarre;
    else
        throw new ClassCastException("ce n'est pas un item!");
}
```

6.3) Plus loin dans la même méthode « **main** » de la classe « **Question6** », il y a ce fragment de code :

```
Set AutresItems = new HashSet();
AutresItems.add(A); // -A- instance créée dans 6.2
AutresItems.add(B); // -B- instance créée dans 6.2
```

Pour que le précédent fragment de code puisse s'exécuter correctement :

a) Est-ce que la classe « **Item** » doit avoir une méthode additionnelle (autre que « **equals** »)? Si oui laquelle?

Comme nous l'avons mentionné dans les notes de cours, toute classe qui redéfinit « equals() » doit redéfinir la méthode « hashCode() ». En effet, deux objets égaux par « equals() » doivent avoir la même clé de hachage par « hashCode() ».

b) Si vous avez répondu « **oui** » à la précédente question, écrire le code de cette méthode.

Le « CodeBarre » étant unique c'est l'élément à utiliser pour calculer le hashCode et s'assurer d'une clé de hachage unique pour éviter les collisions.

```
public int hashCode() {
    return CodeBarre;
}
```

6.4) Plus loin dans la même méthode « **main** » de la classe « **Question6** », il y a ce fragment de code :

```
Set BisItems = new TreeSet(new ItemComparateur());
BisItems.add(A);
BisItems.add(B);
```

Avec les mêmes hypothèses de départ …...

a) Énumérer les changements, si nécessaire, que vous devez apporter à la classe « **Item** ».

La classe ne nécessite pas des modifications. Elle contient déjà le nécessaire.

b) Définir la classe « **ItemComparateur** » (écrire le code complet de la classe).

```
class ItemComparateur implements Comparator {
  public int compare(Object o1, Object o2) {
      int x = ((Item) o1).nom.compareTo(((Item) o2).nom);
      // L'égalité des noms ne garantie pas forcément le même item
      if (x != 0) return x;
      return ((Item) o1).CodeBarre – ((Item) o2).CodeBarre;
  }
}
```

Vous devez comparer d'abord sur les « nom » avant le code barre, pour que les items soient ordonnés en fonction de leur nom. Si vous omettez de comparer sur les noms, « compare » et « compareTo » seront identiques! La question est : pourquoi l'avoir introduit si c'est pour faire le même travail? Le but de faire un tri différent, donc vous avez besoin de comparer sur deux critères : nom et code barre.
Vous devez tenir compte aussi du Code barre sinon deux items ayant le même nom mais un code barre différent, seront vus comme deux items identiques et donc on risque de ne préserver qu'une seule copie!

c) Tout en justifiant votre réponse, est-ce que vous avez besoin de modifier la définition de la méthode « **equals** » de la classe « **Item** »? Si oui comment?

Pas besoin! Deux items identiques ont toujours des codes barres identiques.

d) La classe « **Question6** » contient la méthode « **Affiche** » qui sera utilisée dans la méthode « **main** ». Cette méthode permet d'afficher en sortie les noms des items ainsi que le code barre associé, dans l'ordre établi par la collection utilisée. Un exemple d'appel de « **Affiche** » est donné dans le fragment de code suivant :

```
Set AutresItems = new HashSet();
AutresItems.add(A); // -A- instance créée dans 6.2
AutresItems.add(B); // -B- instance créée dans 6.2
Affiche(AutresItems);
```

Écrire le code de la méthode « **Affiche** » en utilisant **les nouvelles propriétés** de Java 5 (comme le « **foreach** » etc.), sachant que l'avertissement suivant **ne doit pas apparaître** suite à la compilation de votre programme :

« `uses unchecked or unsafe operations. Note: Recompile with -Xlint:unchecked for details` ».

Si on fait abstraction du faite qu'il faut revoir tout le code en y introduisant les génériques dans la méthode « main » et dans la classe « ItemComparateur », la fonction « Affiche » sera comme suit :

```
static void Affiche(Set<Item> Items) {
       for (Item i : Items){
                 System.out.println("Nom de l'item: " + i.nom);
                 System.out.println("Son code barre: " + i.CodeBarre);
            }
}
```

On a besoin de spécifier que l'argument est un « Set » et en plus il est générique. Le type générique est « Item », sinon on obtient le warning signalé précédemment au moment de la compilation.

Bonnes vacances!