Corrigé du Contrôle d'Informatique INF 311

Promotion 2012

Sujet proposé par François Morain et Philippe Chassignet

12 juillet 2013

Seuls les documents fournis dans le cadre du cours et les notes personnelles sont autorisés.

Durée : 2 heures. Les étudiants non-francophones ont le droit à 30 minutes supplémentaires.

Les exercices qui suivent sont indépendants et peuvent être traités dans n'importe quel ordre. On attachera une grande importance à la clarté, à la précision et à la concision de la rédaction.

Un barème pour chaque exercice, pour un total de 20 points, est donné à titre indicatif. Nous donnons le nombre de lignes attendues pour la réponse à chaque question ; si la réponse que vous envisagez est beaucoup plus longue, il est probable qu'il s'agisse d'une solution trop compliquée.

Exercice 1 *barème envisagé : 4 points*

a) On considère la classe:

```
import tc.TC;
public class Question1a{
  public static int f(int n){
    if(n == 0)
      return 1; // (*)
    else
      return n * f(n-1);
  }
  public static void main(String[] args){          // L0
    TC.println("3!="+f(3));                        // L1
  }
}
```

Faire un dessin de la pile d'exécution (mémoire locale) avant de quitter la ligne (*) de la méthode `f`.

(réponse attendue : des explications et un dessin)
Corrigé. cf. transparents de l'amphi 5.

b) On donne la classe

```
public class Entier{
  public static final Entier ZERO = new Entier(0);
  public static final Entier UN = new Entier(1);
  private int valeur;

  public Entier(int i){
    this.valeur = i;
  }
```

```
  public boolean equals(Entier n){
    return this.valeur == n.valeur;
  }
  public Entier subtract(Entier n){
    return new Entier(this.valeur - n.valeur);
  }
  public Entier multiply(Entier n){
    return new Entier(this.valeur * n.valeur);
  }
  public String toString(){
    return "" + this.valeur;
  }
}
```

i) Écrire une méthode récursive de calcul de $n!$ avec cette classe, de sorte qu'on puisse écrire dans la méthode `main`

```
TC.println("3!=" + f(new Entier(3)));
```

(réponse attendue 5 lignes)

Corrigé.

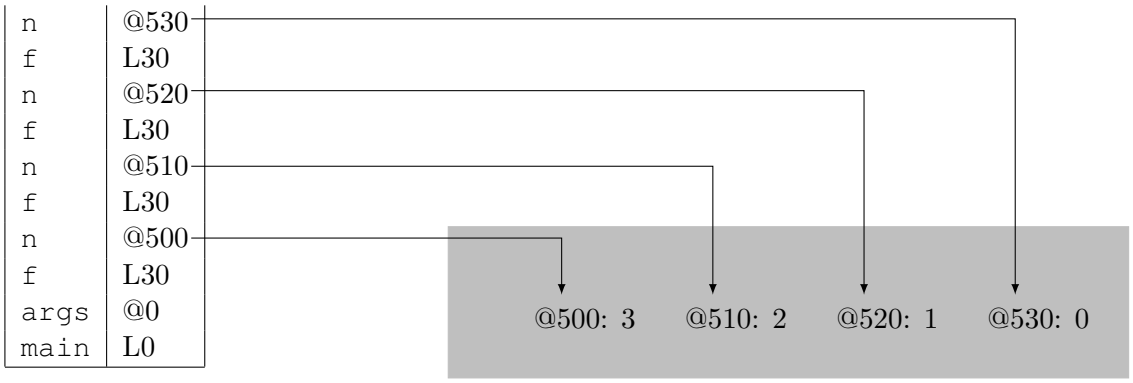
```
  public static Entier f(Entier n){                    // L30
    if(n.equals(Entier.ZERO))                          // L31
      return Entier.UN;                                // L32
    else                                               // L33
      return n.multiply(f(n.subtract(Entier.UN)));     // L34
  }                                                    // L35
```

ii) Il s'agit ici de dessiner la pile d'exécution et la mémoire globale au moment où on s'apprête à quitter l'équivalent de la ligne (*) de la méthode `f` de la question a).

(réponse attendue : des explications et un dessin)

iii) Indiquer, au retour dans `main`, tous les objets de type `Entier` qui ont été construits.

Corrigé. ii) On trouve



iii) Il y a deux constructions par niveau d'appel : `subtract` avant l'appel récursif et `multiply` au retour

```
--> 2           (3-1)
-----> 1        (2-1)
-------> 0      (1-1)
-------> 1      (1*1)
----> 2         (2*1)
--> 6           (3*2)
```

Exercice 2 *barème envisagé : 8 points*

On considère un arbre binaire de recherche $A = (r, G, D)$, dont les nœuds contiennent des **int**, que l'on supposera *distincts*. On définit la *taille* de A comme étant le nombre total de nœuds dans cet arbre. Ainsi, $T(A) = 0$ si A est **null**, et $T(A) = 1 + T(G) + T(D)$ sinon.

a) Compléter le dessin de la figure 1, en indiquant la taille de chaque sous-arbre de racine chaque nœud.

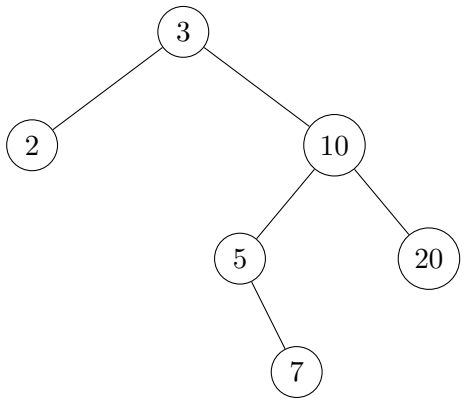


Figure 1: Un arbre binaire de recherche.

(réponse attendue : un dessin)
Corrigé. Voir figure 2.

b) On demande d'écrire une classe `ABRT` qui code un nœud d'un arbre binaire de recherche avec un champ supplémentaire `taille`. On écrira un constructeur pour la création d'une feuille, ainsi qu'un constructeur plus général qui crée l'arbre $A = (r, G, D)$ en mettant correctement à jour le champ `taille` de A.

(réponse attendue : 20 lignes)
Corrigé.

```
public class ABRT{
  private int contenu, taille;
  private ABRT g, d;

  public ABRT(int c){
    this.contenu = c;
    this.g = this.d = null;
    this.taille = 1;
```

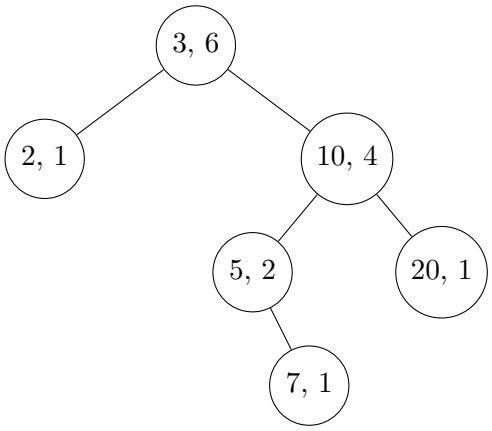


Figure 2: Un arbre avec les tailles de chaque sous-arbre.

```
  }

  public ABRT(int c, ABRT g0, ABRT d0){
    this.contenu = c;
    this.g = g0;
    this.d = d0;
    this.taille = 1;
    if(g0 != null)
      this.taille += g0.taille;
    if(d0 != null)
      this.taille += d0.taille;
  }
}
```

c) Écrire une méthode d'insertion récursive d'une nouvelle valeur dans un `ABRT` dont la signature est

```
public static ABRT inserer(ABRT a, int c){...}
```

Cette méthode retourne l'ABRT résultat et devra *mettre à jour* les tailles de chaque nœud rencontré lors de l'insertion. Si une valeur est déjà présente dans l'arbre, celui-ci reste inchangé.

(réponse attendue : 15 lignes)
Corrigé.

```
public static ABRT inserer(ABRT a, int c){
  if(a == null)
    return new ABRT(c);
  if(c == a.contenu)
    return a;
  if(c < a.contenu){
    a.g = inserer(a.g, c);
    a.taille = 1 + a.g.taille;
    if(a.d != null)
      a.taille += a.d.taille;
```

```
    }
    else{
      a.d = inserer(a.d, c);
      a.taille = 1 + a.d.taille;
      if(a.g != null)
        a.taille += a.g.taille;
    }
    return a;
  }
```

d) Soit A un ABRT avec n nœuds, dont on rappelle que les contenus sont tous distincts. Soit k un entier, $0 \leqslant k < n$. Le but de cet exercice est de trouver rapidement le $(k+1)$-ième entier par ordre croissant dans A.

i) Montrer comment résoudre le cas $k = 0$. On ne demande pas de le programmer. (*réponse attendue: une phrase*)

ii) Expliquer comment résoudre le cas général. Donner une méthode d'objet

```
public int selectionner(int k){...}
```

qui résout le problème dans le cas général. (*réponse attendue: une quinzaine de lignes, programme compris*)

iii) Quelle en est sa complexité en fonction des paramètres de l'arbre ? (*réponse attendue: une phrase*)

iv) Comment tester cette méthode ? (*réponse attendue: une phrase*)

Corrigé. i) Le cas $k = 0$ correspond au minimum des contenus, qui se trouve dans la feuille la plus à gauche de l'arbre.

ii) L'idée est la suivante : soit t la taille du sous-arbre gauche G de $A = (r, G, D)$. Si $k = t$, on sait que le $(k+1)$-ième élément est r. Si $k < t$, alors on doit chercher ce k-ième élément dans G. Si $k > t$, on doit chercher le $(k - t - 1)$-ième élément dans D. On retournera -1 quand la recherche échoue.

```
public int selectionner(int k){
  int t;
  if(this.g == null)
    t = 0;
  else
    t = this.g.taille;
  if(t == k) return this.contenu;
  if(t > k) return this.g.selectionner(k);
  if(t < k) return this.d.selectionner(k-t-1);
  return -1;
}
```

iii) La complexité est en $O(h)$ où h est la hauteur de l'arbre. Dans le cas le plus favorable, $h = O(\log n)$.

iv) Il suffit de générer une permutation aléatoire de $[0..n[$ et de demander en séquence $k = 0, \ldots, n-1$ et à chaque étape, on doit retrouver k.

Exercice 3 *barème envisagé : 8 points*

Pour des raisons de confidentialité industrie, tous les champs demandés dans cet exercice seront ***private*** *et les méthodes* ***public****, sauf justification du contraire.*

Partie I

Vous lancez votre start-up de vente par correspondance à la sortie de l'École. Cette entreprise, nommée FROMAGIX, vend des articles typiquement français : des fromages. Chaque fromage a un nom (une String), une région d'origine (une String), un type de lait (vache, brebis ou chèvre) codé sur un caractère (`'v'`, `'b'` ou `'c'`).

On donne quelques exemples de fromages dans la table 1.

Beaufort	RhoneAlpes	vache
Bleu	Auvergne	vache
Brie	IleDeFrance	vache
Camembert	BasseNormandie	vache
Livarot	BasseNormandie	vache
Picodon	RhoneAlpes	chèvre
Reblochon	RhoneAlpes	vache
Roquefort	MidiPyrénées	brebis

Table 1: Une partie de la table des fromages.

a) i) Écrire une classe `Fromage` qui représente un fromage, ainsi qu'un constructeur qui peut initialiser tous les champs. (*réponse attendue: moins de 10 lignes*).

ii) Écrire une méthode `toString()` qui crée une chaîne de caractère permettant d'afficher un fromage comme dans l'exemple ci-dessous

```
Beaufort (RhoneAlpes)
```

c'est-à-dire le nom suivi de la région entre parenthèses. (*réponse attendue: moins de 3 lignes*)

iii) Pour chaque champ de la classe `Fromage`, écrire une fonction qui permet de récupérer ce champ : `getNom()`, etc. (*réponse attendue: moins de 10 lignes*)

Corrigé.

```
public class Fromage{

  private String nom, region;
  private char lait; // parmi 'b', 'c', 'v'

  public Fromage(String n, String r, char l){
    this.nom = n;
    this.region = r;
    this.lait = l;
  }

  public String toString(){
```

```
    return this.nom+" ("+this.region+")";
  }

  public String getNom(){
    return this.nom;
  }

  public String getRegion(){
    return this.region;
  }

  public char getLait(){
    return this.lait;
  }

}
```

b) Écrire une classe `Fromagerie` qui gère un catalogue de fromages, et qui utilise un tableau de `Fromage` pour son stockage interne. Tout le monde sait qu'il y a en France autant de fromages que de jours dans une année normale, c'est-à-dire 365. Écrire une méthode

```
  public void ajouterFromage(...)
```

qui permette d'ajouter un fromage facilement à partir de la méthode

```
  public static void main(String[] args){
    Fromagerie fromagiX = new Fromagerie();
    fromagiX.ajouterFromage("Beaufort", "RhoneAlpes", 'v');
    fromagiX.ajouterFromage("Bleu", "Auvergne", 'v');
    fromagiX.ajouterFromage("Brie", "IleDeFrance", 'v');
    fromagiX.ajouterFromage("Camembert", "BasseNormandie", 'v');
    fromagiX.ajouterFromage("Livarot", "BasseNormandie", 'v');
    fromagiX.ajouterFromage("Picodon", "RhoneAlpes", 'c');
    fromagiX.ajouterFromage("Reblochon", "RhoneAlpes", 'v');
    fromagiX.ajouterFromage("Roquefort", "MidiPyrenees", 'b');
//    fromagiX.afficherCatalogue();
  }
```

(*réponse attendue: moins de 15 lignes*)

Corrigé.

```
public class Fromagerie{
  final private static int NFROMAGES = 365;
  private Fromage[] fromages;
  private int nbfromages;

  public Fromagerie(){
    this.fromages = new Fromage[NFROMAGES];
    this.nbfromages = 0;
  }

  public void ajouterFromage(String n, String r, char l){
```

```
    this.fromages[this.nbfromages++] = new Fromage(n, r, l);
  }
}
```

c) Écrire une méthode qui affiche le catalogue, en regroupant les fromages faits avec le même type de lait, en respectant l'ordre alphabétique des noms de fromage. On supposera qu'on a rempli ce tableau dans l'ordre alphabétique de ces noms. On devra faire une seule passe sur le catalogue. On trouvera ainsi pour l'exemple

```
Brebis :
Roquefort (MidiPyrenees)
Chevre :
Picodon (RhoneAlpes)
Vache :
Beaufort (RhoneAlpes)
Bleu (Auvergne)
Brie (IleDeFrance)
Camembert (BasseNormandie)
Livarot (BasseNormandie)
Reblochon (RhoneAlpes)
```

Vous devrez utiliser des listes de type `LinkedList<Fromage>`. On supposera donnée une méthode

```
public static void afficher(LinkedList<Fromage> l){...}
```

qui affiche une liste de `Fromages` dans l'ordre de la liste. Vous pourrez utiliser une des méthodes objet de `LinkedList<Fromage>` : `addFirst(Fromage f)` qui ajoute en tête de liste et `addlast(Fromage f)` qui ajoute en fin de liste. (*réponse attendue: une vingtaine de lignes*)

Corrigé. On cumule dans trois listes les fromages par type.

```
public void afficherCatalogue(){
  LinkedList<Fromage> b = new LinkedList<Fromage>();
  LinkedList<Fromage> c = new LinkedList<Fromage>();
  LinkedList<Fromage> v = new LinkedList<Fromage>();
  for(int i = 0; i < this.fromages.length; i++){
    if(this.fromages[i] != null){
      char l = this.fromages[i].getLait();
      switch(l){
      case 'b':
        b.addLast(this.fromages[i]);
        break;
      case 'c':
        c.addLast(this.fromages[i]);
        break;
      case 'v':
        v.addLast(this.fromages[i]);
        break;
      }
    }
```

```
    }
    TC.println("Brebis :"); afficher(b);
    TC.println("Chevre :"); afficher(c);
    TC.println("Vache :");  afficher(v);
  }
```

Pour mémoire :

```
  public static void afficher(LinkedList<Fromage> l){
    for(Fromage f : l)
      TC.println(f);
  }
```

Partie II

Vous décidez de vous diversifier en ajoutant d'autres aliments à votre catalogue. Pour cela, il est temps de réfléchir à la redéfinition des produits vendus. Vous définissez l'interface

```
public interface Produit{
  public String typeDeProduit();
  public String toString();
}
```

d) Expliquer comment modifier la classe `Fromage` pour qu'elle respecte l'interface. (*réponse attendue: moins de 5 lignes*)

Corrigé.

```
public class Fromage implements Produit{
  public String typeDeProduit(){
    return "fromage";
  }
  ...
}
```

e) i) Donner une interface `Fournisseur` qui permette une gestion souple de catalogues de produits (ajout d'un produit, affichage du catalogue), sans connaître à l'avance le type de données utilisé pour ce stockage. On donne la classe qui utilisera une implantation `Grossiste` de l'interface `Fournisseur`.

```
public class TestFournisseur{
  public static void main(String[] args){
    Fournisseur produiX = new Grossiste();
    produiX.ajouterProduit("fromage", "Beaufort RhoneAlpes v");
    produiX.ajouterProduit("fromage", "Bleu Auvergne v");
    produiX.ajouterProduit("fromage", "Brie IleDeFrance v");
    produiX.ajouterProduit("fromage", "Camembert BasseNormandie v");
    produiX.ajouterProduit("fromage", "Livarot BasseNormandie v");
    produiX.ajouterProduit("fromage", "Picodon RhoneAlpes c");
    produiX.ajouterProduit("fromage", "Reblochon RhoneAlpes v");
    produiX.ajouterProduit("fromage", "Roquefort MidiPyrenees b");
    produiX.afficherCatalogue();
  }
}
```

(*réponse attendue: moins de 5 lignes*)

ii) Donner un exemple d'implantation de l'interface `Fournisseur`, qui fera également intervenir l'interface `Produit`. (*réponse attendue: une dizaine de lignes*)

Corrigé. i) On se contente du minimum syndical

```
public interface Fournisseur{
  public void ajouterProduit(String type, String s);
  public void afficherCatalogue();
}
```

ii) Une implantation avec listes est la suivante :

```
import tc.TC;
import java.util.LinkedList;

public class Grossiste implements Fournisseur{
  private static LinkedList<Produit> L;

  public Grossiste(){
    this.L = new LinkedList<Produit>();
  }

  public void ajouterProduit(String type, String s){
    if(type.equals("fromage")){
      String[] tmp = TC.motsDeChaine(s);
      this.L.addLast(new Fromage(tmp[0], tmp[1], tmp[2].charAt(0)));
    }
  }

  public void afficherCatalogue(){
    for(Produit p : this.L)
      TC.println(p);
  }

}
```