Examen 2006-2007 - Durée : 2h

IB2 : Informatique de Base 2

Première année

Tous documents interdits.

★ **Exercice 1.** Questions de cours. (4 pt)

▷ **Question 1.** Expliquez la différence entre typage statique et dynamique. Donnez des exemples. (1 pt)

Réponse

Typage statique : type de la référence à la déclaration (seul type connu lors de la compilation)

Typage dynamique : type de l'objet référencé (type qui ne peut être connu que lors de l'exécution)

Fin réponse

▷ **Question 2.** Expliquez la différence entre la récursivité terminale et la récursivité non-terminale. Donnez un exemple pour chaque cas de récursivité. (1 pt)

Réponse

Récursivité terminale : tous les appels récursifs sont du genre `return f(...)`. Autrement dit, la valeur retournée est directement la valeur obtenue par un appel récursif, sans qu'aucune opérations ne modifie cette valeur.

Fin réponse

▷ **Question 3.** Expliquez la différence entre une méthode d'instance et une méthode de classe. Donnez des exemples. (1 pt)

Réponse

Méthode d'instance : une méthode qui nécessite une instance de la classe comme receveur de l'appel. Elle est en droit de manipuler les variables d'instance.

Méthode de classe : une méthode qui ne nécessite pas d'instance de la classe pour être appelée. Elle ne peut donc pas manipuler les variables d'instance. Par contre, elle peut manipuler les variables de classes (qui sont partagées par toutes les instances d'une même classe).

Fin réponse

▷ **Question 4.** Expliquez la différence entre une classe abstraite et une classe concrète. (1 pt)

Réponse

Classe abstraite : une classe qui ne pas peut être instanciée.

Fin réponse

★ **Exercice 2.** "Rendre la monnaie". (4 pt)

On considère le problème suivant : On possède un tiroir caisse contenant des pièces et des billets de valeurs différentes, et suite à un paiement, on souhaite rendre une certaine somme d'argent en utilisant le moins possible de pièces et billets. On cherche donc une solution optimale.

Voici différentes instances et solutions de ce problème :

coupures de monnaie (en €)	somme à rendre (en €)	solution
10, 5, 2	27	$10+10+5+2$
10, 5, 2	18	$10+5+2+$ ... coincé!!!
		$10+2+2+2+2$
10, 5, 2	3	n'a pas de solution
20, 10, 5, 4, 2, 1	38	$20+10+4+4$
		et non pas $20+10+5+2+1$!!!

▷ **Question 1.** À quel type de problème cela vous fait-il penser ? (1 pt)

Réponse

Sac à dos, n reines, pyramides, ... , un problème de recherche avec retour arrière (*back-tracking*).

Fin réponse

▷ **Question 2.** Écrivez un algorithme récursif permettant de résoudre ce problème. Les différentes valeurs des coupures de monnaie disponibles (pièce, billet) seront stockées dans un tableau nommé `valeurs`. On supposera que l'on possède une quantité infinie de chacune de ces coupures. Le tableau `valeurs` sera passé en paramètre au constructeur de la classe. Lors de l'exploration, la meilleure des solutions sera conservée dans un tableau nommé `bestPrise`, une variable entière `bestNombrePieces` conservera le nombre de pièces utilisées par la meilleure solution trouvée jusqu'ici.

```
1    int valeurs[];
2    int bestPrise[];
3    int bestNombrePieces = -1;
```

Il vous est demandé d'écrire le constructeur `public RendreLaMonnaie(int[] valeurs)` de la classe `RendreLaMonnaie` et d'implanter les méthodes suivantes :

- `int total(int prise[], int niveauMaximum)` qui calcule le nombre total de pièces utilisée dans la solution `prise`. Le paramètre `niveauMaximum` limite le niveau d'exploration (on ne compte pas les pièces qui sont utilisées dans `prise` et dont l'indice est supérieur à `niveauMaximum`).
- `void stockeSiMeilleure(int prise[])` vérifie si la solution `prise` requiert moins de pièces que la meilleure solution actuelle conservé dans le tableau `bestPrise`. Si c'est le cas, la solution prise est recopiée en tant que meilleure solution. Il faut noter que cette méthode également à jour la valeur de `bestNombrePieces`.
- `void cherche(int chercher, int niveau, int prise[])` qui recherche la meilleure solution. Le paramètre `chercher` contient la somme d'argent à rendre. Le paramètre `niveau` contient le niveau d'exploration (l'indice actuel dans le tableau `valeurs`). Le tableau `prise` contient la solution en cours d'élaboration.
- `void afficheSolution()` qui affiche si une solution a été trouvée et dans ce cas affiche le nombre de pièces utilisées au total, et combien de pièces de chaque coupure doivent être utilisées.

Réponse

```
1  package examenib2;
2
3  public class RendreLaMonnaieMartin {
4      int valeurs[];
5      int bestPrise[];
6      int bestNombrePieces = -1;
7
8      public RendreLaMonnaieMartin(int valeurs[]) {
9          this.valeurs = valeurs;
10     }
11
12     public void rendsLaMonnaie(int somme) {
13         bestPrise = new int[this.valeurs.length];
14         bestNombrePieces = -1;
15         cherche(somme, 0, new int[valeurs.length]);
16     }
17
18     public static void main(String args[]) {
19         RendreLaMonnaieMartin rlmb = new RendreLaMonnaieMartin(new int[]{50,20,10,5,4,2,1});
20         rlmb.rendsLaMonnaie(38);
21         rlmb.afficheSolution();
22     }
23
24     public void afficheSolution() {
25         if (bestNombrePieces <= 0) {
26             System.out.println("aucune solution");
27         } else {
28             System.out.print("solution: ");
29             for (int i=0; i<valeurs.length; i++) {
```

```
                if (bestPrise[i] > 0) {
                    System.out.print(bestPrise[i]+"x"+valeurs[i]+", ");
                }
            }
            System.out.println();
        }
    }

    public void cherche(int chercher, int niveau, int prise[]) {
        if (niveau >= prise.length)
            return;
        prise[niveau] = 0;
        cherche(chercher, niveau+1, prise);
        while (total(prise, niveau) < chercher) {
            prise[niveau]++;
            cherche(chercher, niveau+1, prise);
        }
        if (total(prise, niveau) == chercher)
            stockeSiMeilleure(prise, niveau);
    }

    public int total(int prise[], int niveauMaximum) {
        int total = 0;
        for (int i=0; i<=niveauMaximum; i++)
            total += prise[i]*valeurs[i];
        return total;
    }

    public void stockeSiMeilleure(int prise[], int niveauMaximum) {
        int nombrePieces = 0;
        for (int i=0; i<=niveauMaximum; i++) {
            nombrePieces += prise[i];
        }

        if (bestNombrePieces == -1 || nombrePieces < bestNombrePieces) {
            for (int i=0; i<niveauMaximum; i++) {
                bestPrise[i] = prise[i];
            }
            bestNombrePieces = nombrePieces;
        }
    }
}
```

Fin réponse

★ **Exercice 3.** Algorithme récursif “mystère”. (3 pt)

On considère l'algorithme suivant :

```
package examenib2;

public class Mystere {
    public int mystification(int[] tab) {
        return mystification(tab, 0);
    }

    public int mystification(int[] tab, int i) {
        if (i == tab.length-1) {
            return 0;
        } else {
            if (tab[i] > tab[tab.length-1]) {
                return 1+mystification(tab, i+1);
            } else  {
                return mystification(tab, i+1);
            }
        }
    }
}
```

▷ **Question 1.** Montrer la terminaison de cet algorithme. (1 pt)

Réponse

La fonction "s'arrête" quand `i` est égal à `tab.length-1`. Au premier appel, `i = 0`. Puis à chaque appel la valeur de `i` croît strictement. Elle atteindra donc inévitablement la valeur `tab.length-1`.

Fin réponse

▷ **Question 2.** On considére le tableau `tab` de longueur 4 contenant les valeurs suivantes : 3, 4, 1, 2. Explicitez l'appel `(new Mystere()).mystification(tab)` en indiquant le détail de chaque appel récursif. (1 pt)

Réponse

```
mystification([3,4,1,2], 0)
  3 > 2 -> return mystification([3,4,1,2], 1)
               4 > 2 -> return 1+mystification([3,4,1,2], 2)
                            1 > 2  -> return 1+(1+mystification([3,4,1,2], 3))
                                                1+mystification([3,4,1,2], 4)
                                                             return 1+1+0
```

Fin réponse

▷ **Question 3.** Que calcul cet algorithme ? (1 pt)

Réponse

Cette fonction calcule le nombre de valeurs contenues dans le tableau et qui sont strictement plus grande que la dernière valeur du tableau.

Fin réponse

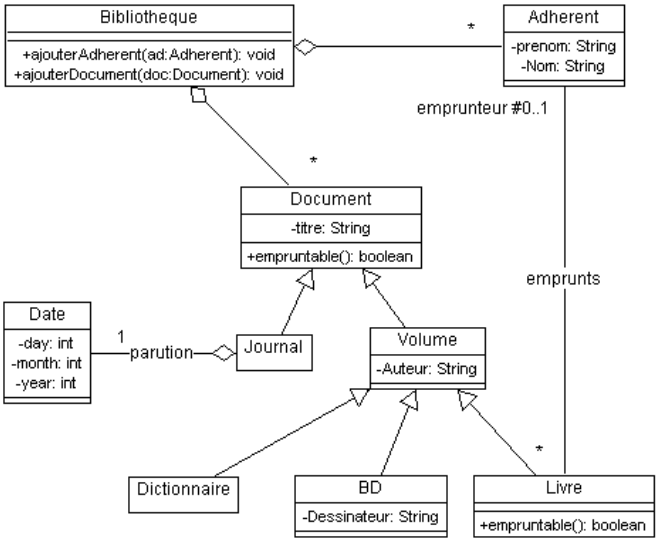
★ **Exercice 4.** Héritage. (3 pt)

On désire réaliser un programme permettant de gérer une petite bibliothèque municipale. L'analyse du problème a montré que l'on avait besoin d'une classe `Bibliotheque`, d'une classe `Adherent` et d'un ensemble de classes de `Document` qui présentent les caractéristiques suivantes :

- Les adhérents ont un prénom (chaîne de caractères) et un nom (chaîne aussi).
- La bibliothèque comprend un ensemble de documents et un ensemble d'adhérents
- Ces documents sont soit des journaux, soit des volumes.
- Les volumes sont soit des dictionnaires soit des livres, soit des BD.
- Les documents sont caractérisés par un titre (chaîne de caractères).
- Les volumes ont en plus un auteur (chaîne de caractères). Les BD ont en plus un nom de dessinateur (chaîne de caractères).
- Les journaux ont (outre les caractéristiques de document) une date de parution (on prendra la classe `java.util.Date`).
- Seuls les livres sont empruntables.
- Les adhérents peuvent emprunter des livres (et uniquement des livres) et on doit pouvoir savoir à tout moment quels sont les livres empruntés par un adhérent.

▷ **Question 1.** Représentez sous la forme d'un diagramme les différentes classes en précisant si elles sont abstraites ou non. Indiquez les méthodes de chaque classes, ainsi que leurs éventuelles redéfinitions. Vous préciserez également les relations de composition ainsi que les relations d'héritages. (3 pt)

Réponse



- 1pt pour les relations d'héritage
- 1pt pour les relations de composition
- 1pt pour le placement des méthodes dans les différentes classes

Fin réponse

★ **Exercice 5.** Typage statique et typage dynamique. (4 pt)

On considère le code suivant :

```
1  package examenib2;
2
3  class A {
4      protected int x;
5
6      public A(int x) {
7          this.x = x ;
8      }
9      public void f(A a) {
10         x = x - a.x ;
11     }
12     public String toString() {
13         return "A, x = "+x;
14     }
15 }
16
17 class B extends A {
18     protected A a;
19
20     public B(A a, int x) {
21         super(1);
22         this.a = a ;
23     }
24     public void f(A b) {
25         a.x = a.x - b.x;
26         x = x + b.x;
27     }
28     public void f(B b) {
29         a.x = a.x - 2*b.x;
30         x = x + 2*b.x ;
31     }
32     public String toString() {
33         return "B, x = "+x+", a = <"+a.toString()+">";
34     }
35 }
36
37
38 public class OnTeste2 {
39     public static void main(String[] args) {
40         A a = new A(1), b = new B(a, 1);
41         System.out.println(b);
42         a.f(a);
```

```
        System.out.println(b); //schema1
        a = new A(1);b = new B(a, 1);
        b.f(b);
        System.out.println(b); //schema2
        a = new A(1); b = new B(a, 1);
        B b0 = new B(a, 1);
        b.f(b0);
        System.out.println(b);//schema3
        a = new A(1);b0 = new B(a, 1);
        b0.f(b0);
        System.out.println(b0);//schema4
    }
}
```

▷ **Question 1.** Une fois le programme compilé, on lance la commande : `java examenib2.OnTeste2`. En vous aidant de schémas mémoire (4 schémas) dire ce qui s'affiche et expliquer pourquoi. (4 pt)

Réponse

1pt par schéma + explications relatives.

```
B, x = 1, a = <A, x = 1>
B, x = 1, a = <A, x = 0>
B, x = 2, a = <A, x = 0>
B, x = 2, a = <A, x = 0>
B, x = 3, a = <A, x = -1>
```

Fin réponse

★ **Exercice 6.** Implanter un itérateur. (3 pt)

On considère l'interface Java suivante :

```
package examenib2;

public interface IterateurTabInt {
    public abstract int suivant();
    public abstract int indiceDuSuivant();
    public abstract boolean aUnSuivant();
}
```

On veut écrire une classe d'itérateurs qui parcourent uniquement les cases contenant un nombre pair dans un tableau quelconque d'entiers. Cette classe se nommera `IterateursDesPairs` et implantera l'interface `IterateurTabInt`.

Le sens du parcours est celui des indices croissants. La référence du tableau à parcourir est un attribut noté `tab` et est transmise lors de l'instanciation de l'itérateur. L'indice du tableau dont la valeur est retournée par la méthode `suivant()` est noté `pos`. Cet indice est mis à jour la première fois à l'instanciation, et, les fois suivantes, lors des appels à la méthode `suivant()`. Il n'y a plus de suivant quand cet indice est égal à la longueur du tableau.

Les premières déclarations de la classe sont donc :

```
    private int[] tab;
    private int pos;

    public IterateurDesPairs(int[] tab) {
        this.tab = tab;

        // à compléter
    }
```

▷ **Question 1.** Écrivez le code de la classe `IterateurDesPairs`. (2 pt)

Réponse

```
package examenib2;

public class IterateurDesPairs implements IterateurTabInt {
    private int[] tab;
```

```
    private int pos;

    public IterateurDesPairs(int[] tab) {
        this.tab = tab;
        pos = -1;
    }

    public int suivant() {
        pos = indiceDuSuivant();
        return tab[pos];
    }
    public int indiceDuSuivant() {
        int indice = pos+1;
        while ((indice < tab.length) && (tab[indice] % 2 != 0)) {
            indice++;
        }

        return indice;
    }
    public boolean aUnSuivant() {
        int indice = indiceDuSuivant();
        return (pos < indice) && (indice < tab.length) ;
    }
}
```

Fin réponse

▷ **Question 2. (Question Bonus)** Complétez le corps du constructeur de la classe `Test` pour avoir à l'exécution les affichages suivants (format : indice de la valeur entière paire → entier pair) (1 pt)

```
test1 : 1->2,2->6,4->8,7->12,8->14,
test2:
```

```
package examenib2;

public class TestIterateur {
    public TestIterateur(String message, int[] t) {
        IterateurDesPairs i = new IterateurDesPairs(t);
        System.out.print(message);

        // à compléter
    }

    public static void main(String[] args) {
        new TestIterateur("test1 : ", new int[] {1,2,6,7,8,9,11,12,14,13,5});
        new TestIterateur("test2 : ", new int[] {1, 79});
    }
}
```

Réponse

```
package examenib2;

public class TestIterateur {
    public TestIterateur(String message, int[] t) {
        IterateurDesPairs i = new IterateurDesPairs(t);
        System.out.print(message);

        while (i.aUnSuivant()) {
            System.out.print(i.indiceDuSuivant()+"->"+i.suivant()+",");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        new TestIterateur("test1 : ", new int[] {1,2,6,7,8,9,11,12,14,13,5});
        new TestIterateur("test2 : ", new int[] {1,79});
    }
}
```

Fin réponse