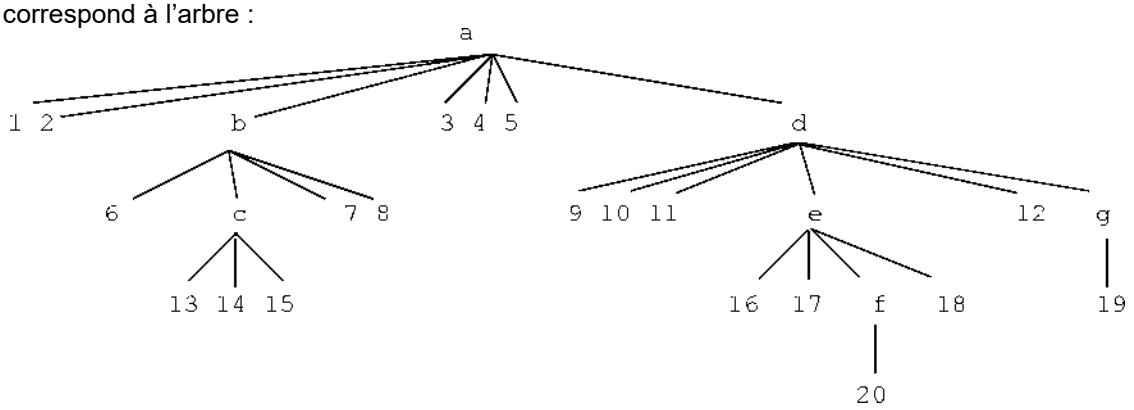
Exercices de Patterns (6)

Une structure de liste pouvant contenir des sous listes peut être représentée sous forme parenthésée mais aussi sous forme d'arbre général. Par exemple la liste :

```
( 1 2 ( 6 ( 13 14 15 ) 7 8 ) 3 4 5 ( 9 10 11 ( 16 17 ( 20 ) 18 ) 12 ( 19 )))
```

correspond à l'arbre :



Le programme suivant construit l'arbre à partir de la chaîne située dans le fichier question.txt. Ensuite il demande différents traitement sur cet arbre : d'abord reconstruire la chaîne de départ à partir de l'arbre ensuite calculer certaines statistiques : nombre de valeurs, nombre de groupe de parenthèses, somme des valeurs et moyenne de celles-ci. Enfin, il affiche le contenu de chaque groupe. La sortie est la suivante :

```
( 1 2 ( 6 ( 13 14 15 ) 7 8 ) 3 4 5 ( 9 10 11 ( 16 17 ( 20 ) 18 ) 12 ( 19 ) ) )
nombre de valeurs : 20
nombre de groupes : 7
somme des valeurs : 210
moyenne des valeurs : 10.5
```

```
a
b  6 c 7 8
c  13 14 15
d  9 10 11 e 12 g
e  16 17 f 18
f  20
g  19
```

Voici le code complet de ce programme. Identifiez dans ce code les divers patterns utilisés. Pour chacun d'eux, indiquez son nom et faites un tableau de correspondance entre les noms des participants théoriques (tels que vu au cours) et les noms des classes dans le code donné. Si le cours théorique propose des noms pour certaines méthodes, indiquez également la correspondance à ce niveau, en indiquant le nom théorique de la méthode et, en regard, le nom employé ici.

Partie.java :

```
public interface Partie {
  void demande(Traitement traitement);
  int getNiveau();
}
```

Valeur.java :

```
public class Valeur implements Partie {
  private int nombre;
  private int niveau = 0;
  public Valeur(int nombre, int niveau) {
    this.nombre = nombre;
    this.niveau = niveau;
  }
  public void demande(Traitement traitement) {
    traitement.traiteValeur(this);
  }
  public int getValeur() {
    return nombre;
  }
  public int getNiveau() {
    return niveau;
  }
}
```

Groupe.java :

```
import java.util.Collections;
import java.util.Iterator;
import java.util.Vector;

public class Groupe implements Partie {
  private static int dernier = 0;
  private Vector parties = new Vector();
  private int niveau;
  private String nom;
  private int numéro;
  public Groupe(String nom, int niveau) {
    this.nom = nom;
    this.niveau = niveau;
    numéro = dernier++;
  }
  public void add(Partie nouveau) {
    parties.add(nouveau);
  }
  public Iterator getParties() {
    return Collections.unmodifiableList(parties).iterator();
  }
  public int getNiveau() {
    return niveau;
  }
  public String getNom() {
    return nom;
  }
  public int getNuméro() {
    return numéro;
  }
  public void demande(Traitement traitement) {
    traitement.traiteGroupe(this);
  }
}
```

Fabriquant.java :

```
public class Fabriquant {
  private Pile pile = new PileImpl();
  private Partie résultat;
  private int niveau = 0;
  char c = 'a';
  public void construireGroupe() {
    if (!pile.estVide()) {
      Groupe sommet = (Groupe)pile.pop();
      Groupe nouveau = new Groupe(String.valueOf(c++), ++niveau);
      sommet.add(nouveau);
      pile.push(sommet);
      pile.push(nouveau);
    } else {
      Groupe nouveau = new Groupe(String.valueOf(c++), 0);
      pile.push(nouveau);
      if (résultat == null) {
        résultat = nouveau;
      } else {
        throw new RuntimeException(); // fichier malformé
      }
    }
  }
  public void fermerGroupe() {
    pile.pop(); niveau-- ;
  }
  public void construireValeur(int nombre) {
    Valeur number = new Valeur(nombre, niveau+1);
    if (!pile.estVide()) {
      Groupe sommet = (Groupe)pile.pop();
      sommet.add(number);
      pile.push(sommet);
    } else résultat = number;
  }
  public Partie getRésultat() {
    return résultat;
  }
}
```

Lecteur.java :

```
import java.io.*;

public class Lecteur {
  private PushbackReader buffer;
  private Fabriquant fabriquant = new Fabriquant();

  public Lecteur(String fichier) throws FileNotFoundException {
    buffer = new PushbackReader(new BufferedReader(new FileReader(fichier)));
  }
  public Partie construire() throws IOException {
    int car;
    while ((car = buffer.read()) != -1) {
      if (car == '(') {
        fabriquant.construireGroupe();
      } else if (car == ')') {
        fabriquant.fermerGroupe();
      } else if (Character.isDigit((char)car)) {
        int nombre = 0;
        do {
          nombre *= 10;
          nombre += Character.digit((char)car, 10);
          if ((car = buffer.read()) == -1) break;
        } while(Character.isDigit((char)car));
        if (car != -1) buffer.unread(car);
        fabriquant.construireValeur(nombre);
      }
    }
    return fabriquant.getRésultat();
  }
}
```

Traitement.java :

```
public interface Traitement {
  void traiteValeur(Valeur unique);
  void traiteGroupe(Groupe plusieurs);
}
```

Listeur.java :

```
import java.util.Iterator;

public class Listeur implements Traitement {
  public void traiteValeur(Valeur valeur) {
    System.out.print(valeur.getValeur() + " ");
  }
  public void traiteGroupe(Groupe groupe) {
    System.out.print("( ");
    Iterator it = groupe.getParties();
    while(it.hasNext()) {
      Partie p = (Partie)it.next(); p.demande(this);
    }
    System.out.print(") ");
  }
}
```

Totaliseur.java :

```
import java.util.Iterator;
public class Totaliseur implements Traitement {
  private int nombreDeValeurs = 0;
  private int nombreDeGroupes = 0;
  private int sommeDesValeurs = 0;
  public void traiteValeur(Valeur valeur) {
    nombreDeValeurs++;
    sommeDesValeurs += valeur.getValeur();
  }
  public void traiteGroupe(Groupe groupe) {
    nombreDeGroupes++;
    Iterator it = groupe.getParties();
    while(it.hasNext()) {
      Partie p = (Partie)it.next();
      p.demande(this);
    }
  }
  public int getNombreDeGroupes() {
    return nombreDeGroupes;
  }
  public int getNombreDeValeurs() {
    return nombreDeValeurs;
  }
  public int getSommeDesValeurs() {
    return sommeDesValeurs;
  }
}
```

ListeurNom.java :

```
import java.util.Iterator;
import java.util.Vector;

public class ListeurNom implements Traitement {
  private Vector lesNoms = new Vector();
  private int enCours = -1;
  public void traiteValeur(Valeur valeur) {
    String val = valeur.getValeur() + " ";
    if (enCours == -1) {
      lesNoms.add(val);
    } else {
      String s = (String) lesNoms.get(enCours);
      lesNoms.set(enCours, s + val);
    }
  }
  public void traiteGroupe(Groupe groupe) {
    if (enCours != -1) {
      String s = (String) lesNoms.get(enCours);
      lesNoms.set(enCours, s + groupe.getNom() + " ");
    }
    int exEnCours = enCours;
    enCours = groupe.getNuméro();
    lesNoms.setSize(enCours + 1);
    lesNoms.set(enCours, groupe.getNom() + ": ");
    Iterator it = groupe.getParties();
    while(it.hasNext()) {
      Partie p = (Partie)it.next();
      p.demande(this);
    }
    enCours = exEnCours;
  }
  public Iterator getLesNoms() {
    return lesNoms.iterator();
  }
}
```

Pile.java :

```
public interface Pile {
  boolean estVide();
  void push(Object n);
  Object pop();
  Object sommet();
  int taille();
  String toString();
}
```

PileImpl.java :

```
public class PileImpl implements Pile {
  private NoeudPile tête;
  private int taille;
  public PileImpl() {
    this.tête = null;
  }
  public void push(Object élément) {
    this.tête = new NoeudPile(élément, this.tête);
    this.taille++;
  }
  public Object pop() {
    Object résultat = sommet();
    this.tête = this.tête.getSuivant();
    this.taille--;
    return résultat;
  }
  public Object sommet() {
    return this.tête.getElément();
  }
  public boolean estVide() {
    return this.tête == null;
  }
  public int taille() {
    return this.taille;
  }
  public String toString() {
    NoeudPile courant = this.tête;
    String résultat = "";
    while (courant != null) {
      résultat += courant.getElément();
      résultat += " ";
      courant = courant.getSuivant();
    }
    return résultat;
  }
  class NoeudPile {
    private Object élément;
    private NoeudPile suivant;
    public NoeudPile(Object élément) {
      this(élément, null);
    }
    public NoeudPile(Object élément, NoeudPile suivant) {
      setElément(élément);
      setSuivant(suivant);
    }
    public void setElément(Object élém) {
      this.élément = élém;
    }
    public void setSuivant(NoeudPile suiv) {
      this.suivant = suiv;
    }
    public Object getElément() {
      return this.élément;
    }
    public NoeudPile getSuivant() {
      return this.suivant;
    }
```

```
  }
}
```

Donnons enfin un exemple de programme principal illustrant l'usage de ces classes :
Main.java :

```
import java.io.IOException;
import java.util.Iterator;

public class Main {
  public static void main(String[] args) throws IOException {
    Lecteur lecteur = new Lecteur("question.txt");
    Partie partie = lecteur.construire();
    Listeur listeur = new Listeur();
    Totaliseur total = new Totaliseur();
    partie.demande(listeur);
    System.out.println();
    System.out.println();
    partie.demande(total);
    System.out.println("nombre de valeurs : " + total.getNombreDeValeurs());
    System.out.println("nombre de groupes : " + total.getNombreDeGroupes());
    System.out.println("somme des valeurs : " + total.getSommeDesValeurs());
    System.out.println("moyenne des valeurs : +
      total.getSommeDesValeurs() / (double) total.getNombreDeValeurs());
    System.out.println();
    ListeurNom listeurNom = new ListeurNom();
    partie.demande(listeurNom);
    Iterator itérateur = listeurNom.getLesNoms();
    while(itérateur.hasNext()) {
      String s = (String) itérateur.next();
      System.out.println(s);
    }
    System.out.println();
  }
}
```

Nom du Pattern :

Participants théoriques **Noms utilisés dans le code**

Nom théorique des méthodes **Nom utilisé dans le code**

Nom du Pattern :

Participants théoriques **Noms utilisés dans le code**

Nom théorique des méthodes **Nom utilisé dans le code**

Exercices de Patterns (6)

<u>Nom du Pattern</u> :

Participants théoriques **Noms utilisés dans le code**

Nom théorique des méthodes **Nom utilisé dans le code**

<u>Nom du Pattern</u> :

Participants théoriques **Noms utilisés dans le code**

Nom théorique des méthodes **Nom utilisé dans le code**

<u>Nom du Pattern</u> :

Participants théoriques **Noms utilisés dans le code**

Nom théorique des méthodes **Nom utilisé dans le code**