IN413 (Design Patterns en java) : Contrôle final

ESIEE Engineering, Jean-Michel DOUIN & Denis BUREAU, 8 novembre 2010

2 heures, SANS CALCULATRICE, UN SEUL DOCUMENT AUTORISÉ.

Question 1 (/ 6 pts) : Questions de cours SUR LA COPIE 1

- **1.1)** Rappelez l'usage d'une interface vide, par exemple :

```
public interface Vide {}
```

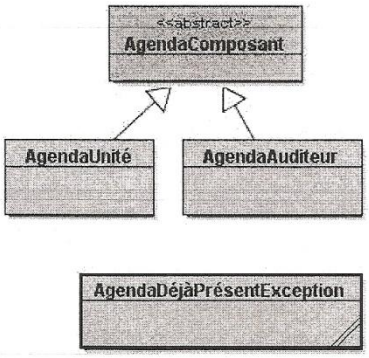
- **1.2)** Que signifie cette déclaration :

```
public class Conteneur< T extends Comparable<? super T> >
{
. . .
```

- **1.3)** Patterns :
  - Donnez un exemple du pattern Délégation (sans écrire de code Java).
  - Citez ses avantages/inconvénients par rapport à l'héritage.

Question 2 (/ 7 pts) : Patrons composite et visiteur SUR LA COPIE 2

Le **patron Composite** permet de définir une structure de données récursive. Ce patron est utilisé ici pour la définition d'un agenda ou d'un groupe d'agendas. A chaque unité d'enseignement est associée un agenda. L'agenda d'un étudiant de l'ESIEE est un composite d'agendas des unités auxquelles il est inscrit. L'agenda de l'unité est une feuille. L'agenda de l'auditeur est un composite. Une exception est levée si lors d'un ajout d'un agenda celui-ci est déjà présent. Ci-dessous l'architecture des classes reflétant ce patron :



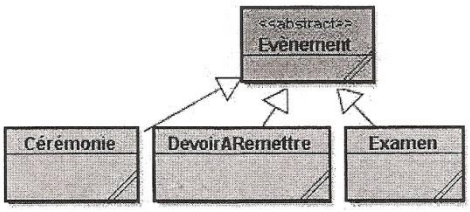
La classe abstraite ***AgendaComposant*** (non demandée) contient les entêtes des méthodes d'ajout, de test de la présence d'un évènement et la possibilité de parcourir cette structure à l'aide d'un visiteur. *(ajouter, estPrésent, accepter)*

La classe concrète ***AgendaUnité*** (non demandée) est une feuille de ce composite. Cette classe hérite de la classe ***AgendaComposant*** et définit toutes ses méthodes.

La classe ***AgendaAuditeur*** est un composite, constituée d'***AgendaComposant*** ; cette classe propose les méthodes d'ajout, de test de la présence d'un agenda, et définit les méthodes de la classe abstraite ***AgendaComposant*** .

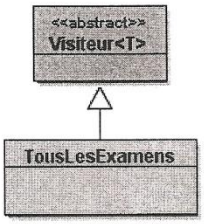
➔ **La documentation de ces 3 classes est en annexe.**

Un agenda contient des évènements ; les évènements contiennent une date (java.util.Calendar) et une description ; ceux-ci peuvent décrire une date d'examen avec une durée, ou une échéance de devoir à remettre, ou bien le jour d'une cérémonie comme le jour de la remise du diplôme.



La classe *Examen* hérite de la classe ***Evènement*** : un attribut mentionnant la durée de cet examen est ajouté.

Le **patron Visiteur** associé au composite d'agendas permet de parcourir une structure d'agendas.



La classe ***TousLesExamens*** est un visiteur possible : celui-ci fournit la liste de tous les examens d'un agenda (composant).

2.1) Précisez le contexte et les bénéfices escomptés de l'utilisation du patron Composite.

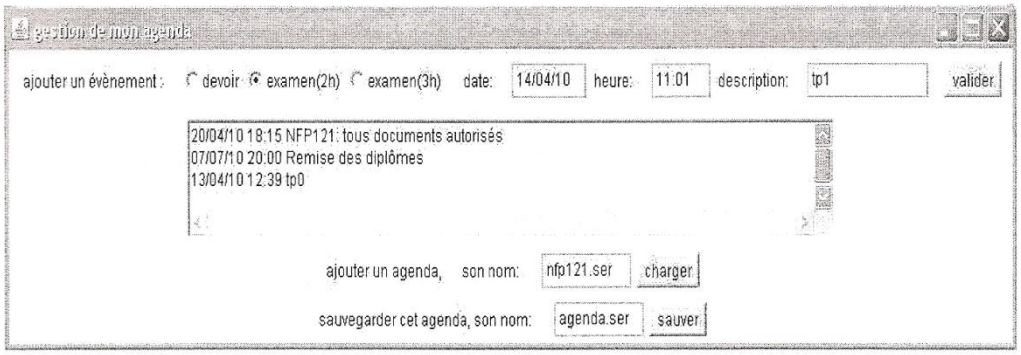
2.2) Proposez une implémentation de la classe composite d'agendas ***AgendaAuditeur***.

2.3) Écrivez le visiteur ***TousLesExamens*** : cette visite doit permettre d'obtenir la liste de tous les examens d'un agenda.

➔ **Les documentations de certaines classes sont en annexe ; lisez attentivement la classe de test.**

Question 3 (/ 7 pts) : Listener et entrées-sorties SUR LA COPIE 2

Voici l'Interface Homme-Machine désirée :



Cette interface permet de saisir plusieurs champs :

- La date, l'heure, et la description d'un évènement,
- Le nom d'un fichier pour la lecture d'un agenda,
- Le nom d'un fichier pour la sauvegarde de cet agenda.

et une zone de texte reflétant le contenu actualisé de l'agenda.

De larges extraits de la classe IHM sont donnés en annexe, notamment la création de tous les composants et des méthodes de chargement/sauvegarde/affichage de l'agenda.

Ajoutez un "Listener" pour chaque bouton de cette interface :

- **3.1)** L'action sur le bouton **valider** permet d'ajouter un évènement à l'agenda ; la lecture des deux champs date et heure afin de produire une instance de la classe java.util.Calendar peut être effectuée à l'aide de cet exemple :

```
Calendar c = Calendar.getInstance();
DateFormat df = DateFormat.getDateTimeInstance(
  DateFormat.SHORT, DateFormat.SHORT, Locale.FRANCE);
Date d = df.parse( "13/04/10 12:41" );
c.setTime( d );
```

- **3.2)** L'action sur le bouton **charger**, permet d'ajouter un agenda à l'agenda actuel, à partir de la lecture d'un agenda préalablement sauvegardé.

- **3.3)** L'action sur le bouton **sauver**, permet de sauvegarder l'agenda actuel dans un fichier.

➔ **La documentation de certaines classes est en annexe.**

Annexe pour la question 2

AgendaComposant :

```
public abstract class AgendaComposantextends java.lang.Objectimplements java.io.Serializable
```

See Also:
Serialized Form

Constructor Summary
AgendaComposant()

Method Summary	
abstract <T> T	accepter(questionl.Visiteur<T> v) Acceptation d'un visiteur.
abstract void	ajouter(questionl.Evènement évènement) Ajout d'un évènement à cet agenda.
abstract boolean	estPrésent(questionl.Evènement évènement) Test de la présence d'un évènement.

AgendaUnité :

Constructor Summary
AgendaUnité(java.lang.String nom)

Method Summary	
<T> T	accepter(questionl.Visiteur<T> v)
void	ajouter(questionl.Evènement évènement)
java.util.List<questionl.Evènement>	contenu()
boolean	equals(java.lang.Object o)
boolean	estPrésent(questionl.Evènement évènement)
java.lang.String	toString()

AgendaAuditeur :

Constructor Summary
AgendaAuditeur()

Method Summary	
<T> T	accepter(question1.Visiteur<T> v) Acceptation d'un visiteur.
void	ajouter(question1.AgendaComposant agenda) Ajout d'un agenda au composite, une exception est levée si cet agenda est déjà présent.
void	ajouter(question1.Evènement évènement) Ajout d'un évènement à cet agenda.
java.util.List<question1.AgendaComposant>	contenu() Obtention du contenu d'un agenda.
boolean	equals(java.lang.Object o) Egalite de deux agendas auditeur.
boolean	estPrésent(question1.Evènement évènement) Test de la présence d'un évènement.
java.lang.String	toString() Version textuelle de l'agenda.

La classe Evènement :

Constructor Summary
Evènement(java.util.Calendar date, java.lang.String description) Création d'un évènement.

Method Summary	
java.util.Calendar	calendar() Obtention de la date.
protected java.lang.Object	clone() Obtention d'une copie d'un évènement.
int	compareTo(Evènement e1) Comparaison de deux évènements, relation d'ordre sur les dates.
java.lang.String	description() Obtention de la description.
boolean	equals(java.lang.Object o) Egalite de deux évènements.
int	hashCode() Obtention de la valeur de hashCode.
java.lang.String	toString() Une version textuelle de l'évènement.

La classe Examen qui hérite de la classe Evènement :

Field Summary	
static int	DEUX_HEURES
static int	TROIS_HEURES

Constructor Summary
Examen(java.util.Calendar date, int durée, java.lang.String description) Création d'un examen, date,durée et description.

Method Summary	
int	durée() Obtention de la durée de cet examen.

La classe Visiteur :

Constructor Summary
Visiteur()

Method Summary	
abstract T	visite(question1.AgendaAuditeur aa) Visite d'un composite : un Agenda d'un auditeur.
abstract T	visite(question1.AgendaUnité a) Visite d'une feuille : un Agenda d'une unité d'enseignement.

Une classe de tests unitaires :

```
public class UnTest extends junit.framework.TestCase
{

  public void testUnAgendaAuditeur() throws Exception
  {
    AgendaAuditeur monAgenda = new AgendaAuditeur();
    AgendaUnité in413    = new AgendaUnité( "IN413" );
    AgendaUnité ov5SEJA = new AgendaUnité( "OV5_SEJA" );
    monAgenda.ajouter( in413 );
    monAgenda.ajouter( ov5SEJA );

    Calendar c = Calendar.getInstance();
    c.set( 2010, Calendar.NOVEMBER, 8, 10, 00 );
    Evènement examenIN413
      = new Examen( c, Examen.DEUX_HEURES, "IN413: design patterns" );
    in413.ajouter( examenIN413 );

    Calendar c1 = Calendar.getInstance();
    c1.set( 2010, Calendar.DECEMBER, 14, 14, 00 );
    Evènement e1
      = new Examen( c1, Examen.DEUX_HEURES, "OV5_SEJAV: aucun document autorisé" );
    ov5SEJA.ajouter( e1 );

    Calendar c2 = Calendar.getInstance();
    c2.set( 2010, Calendar.NOVEMBER, 11, 23, 59 );
    Evènement e2 = new DevoirARemettre( c2, "tp4" );
    in413.ajouter( e2 );

    Calendar c3 = Calendar.getInstance();
    c3.set( 2010, Calendar.JULY, 7, 20, 00 );
    Evènement e3 = new Cérémonie( c3, "Remise des diplômes" );
    monAgenda.ajouter( e3 );

    List<Examen> li = monAgenda.accepter( new TousLesExamens() );

    assertTrue( li.contains( examenIN413 ) );
  } // testUnAgendaAuditeur()

} // UnTest
```

Annexe pour la question 3

java.awt.event

Interface ActionListener

Method Summary	
void	actionPerformed(ActionEvent e) Invoked when an action occurs.

java.awt

Class Checkbox

boolean	getState() Determines whether this check box is in the "on" or "off" state.

```
// Extraits de la classe IHM

package question3;

import question2.*;

// import java . . .

public class IHM extends Frame
{
  public static final String NOM_PAR_DEFAUT = "agenda.ser";

  private TextField date, heure, description;
  private Checkbox  devoir, examen2h, examen3h;
  private TextArea  contenu;
  private TextField nomAgendaACharger, nomAgendaASauvegarder;
  private Button    valider, charger, sauver;

  private AgendaAuditeur agenda;

  public IHM() throws Exception
  {
    super( "gestion de mon agenda" );

    CheckboxGroup groupe = new CheckboxGroup();
    this.devoir   = new Checkbox( "devoir",     groupe, false );
    this.examen2h = new Checkbox( "examen(2h)", groupe, true  );
    this.examen3h = new Checkbox( "examen(3h)", groupe, false );

    this.date        = new TextField(  6 );
    this.heure       = new TextField(  4 );
    this.description = new TextField( 12 );
    this.description.setText( "tp1" );

    Calendar calendar = Calendar.getInstance();
    DateFormat df = DateFormat.getDateInstance( DateFormat.SHORT, Locale.FRANCE );
    DateFormat dt = DateFormat.getTimeInstance( DateFormat.SHORT, Locale.FRANCE );
    this.date.setText(  df.format( calendar.getTime() ) );
    this.heure.setText( dt.format( calendar.getTime() ) );

    this.valider = new Button( "valider" );
    this.charger = new Button( "charger" );
    this.sauver  = new Button( "sauver"  );

    this.contenu = new TextArea( 4, 80 );
    this.contenu.append( "le contenu : " );

    this.nomAgendaACharger = new TextField( 8 );
    this.nomAgendaACharger.setText( "nfp121.ser" );

    this.nomAgendaASauvegarder  = new TextField( 8 );
    this.nomAgendaASauvegarder.setText( NOM_PAR_DEFAUT );

//  setLayout( . . .

//  . . . add( . . .

    agenda = new AgendaAuditeur();

    try {
      chargerLAgenda( NOM_PAR_DEFAUT );
      afficherLeContenuDeLAgenda();
    } catch( Exception e ) {
      e.printStackTrace(); // il vaudrait mieux afficher dans this.contenu
    }
```

```
    // Les listener à écrire :
    valider . . .
    charger . . .
    sauver  . . .

    pack();
    setVisible(true);

  } // IHM()

  // 3 méthodes très utiles :

  private void chargerLAgenda( String nomAgenda ) throws Exception
  {
    ObjectInputStream ois = new ObjectInputStream(
      new FileInputStream( nomAgenda ) );
    AgendaComposant a = (AgendaComposant)ois.readObject();
    ois.close();
    agenda.ajouter( a );
  } // chargerLAgenda()

  private void sauverLAgenda() throws Exception
  {
    ObjectOutputStream oos = new ObjectOutputStream(
      new FileOutputStream( nomAgendaASauvegarder.getText() ) );
    oos.writeObject( agenda );
    oos.close();
  } // sauverLAgenda()

  private void afficherLeContenuDeLAgenda()
  {
    contenu.setText( "" );
    for ( Evènement e :  agenda.accepter( new TousLesEvènements() ) )
      contenu.append( e +"\n" );
  } // afficherLeContenuDeLAgenda()

} // IHM
```