# DS Programmation Java - S3 - Janvier 2013 – 2h

Documents autorisés : polycopiés de cours, notes, programmes de TD et TP.  
Il est conseillé de lire l’énoncé en entier avant de commencer.  
Le barême est donné à titre indicatif.  
**On ne vous demande ni les accesseurs, ni les import.**

## Exercice 1 (2 pts)

1. Dans quelle zone de la mémoire sont stockés les variables d’instances ?

Dans le tas. (O,5 pt)

1. Dans quelle zone de la mémoire sont stockées les copies des paramètres passés à une méthode lors d’un appel ?

Dans la pile. (O,5 pt)

1. Dans quelle zone de la mémoire sont stockées les variables locales aux méthodes ?

Dans la pile. (O,5 pt)

1. Dans quelle zone de la mémoire sont stockés les instances de classes abstraites ?

Il est impossible d’instancier une classe abstraite ! (O,5 pt)

## Exercice 2 (16 pts)

On souhaite développer une application interactive permettant de saisir et visualiser des informations d’un ciné club.

1) Un film est défini par son titre, son année et le nom du fichier image de l’affiche du film.   
Définir la classe ModeleFilm (entête, attributs, constructeur).

**public** **class** ModeleFilm {

**private** String titre ;

**private** **int** annee ;

**private** String nomImageAffiche ;

**public** ModeleFilm(String titre, **int** annee, String nomImageAffiche) {

**this**.titre = titre;

**this**.annee = annee;

**this**.nomImageAffiche = nomImageAffiche;

}

}

2) On souhaite définir plusieurs vues sur un film. Une vue devra forcément contenir une méthode qui met à jour la vue en fonction de l’état du modèle. Comment faire pour éviter des codages « en dur » des classes concrètes ?

=> Remplacer le nom de la classe concrete ModeleFilm par une interface (par exemple InterfaceModeleFilm). On pourra changer la classe concrète sans avoir à modifier le code de la vue. Idem pour les vues : définir une interface InterfaceVueFilm. Par exemple une classe Controleur ne comportera pas de nom de classes concretes de vue.

Définir si besoin une interface / classe abstraite / classe concrète qui permettra à une vue de prendre plusieurs formes.

**package** cineclub.vues;

**public** **interface** InterfaceVueFilm {

**public** **void** miseAJour ();

}

3) On souhaite visualiser les informations concernant un film de la manière suivante (figure ci-dessous).   
Programmez la vue (entête, attributs, constructeur, et main() pour tester la vue et affiche les informations ci-dessous).



**package** cineclub.vues;

**import** java.awt.GridLayout;

**import** javax.swing.\*;

**import** cineclub.modeles.ModeleFilm;

**public** **class** VueFilmLabels **extends** JPanel **implements** InterfaceVueFilm {

**private** **static** **final** **long** *serialVersionUID* = 1L;

ModeleFilm m ;

JLabel titre ;

JLabel annee ;

**public** VueFilmLabels(ModeleFilm m) {

**this**.m = m ;

// Créer les composants

titre = **new** JLabel (m.getTitre());

annee = **new** JLabel (""+m.getAnnee());

// Mise en page

setLayout (**new** GridLayout (2, 2));

// Ajouter les composants

add (**new** JLabel ("Titre : "));

add (titre);

add (**new** JLabel ("Annee : "));

add (annee);

}

**public** **static** **void** main(String[] args) {

// Créer un modèle

ModeleFilm m = **new** ModeleFilm ("Gravity", 2013, "Gravity.jpg");

// Creer une fenetre

JFrame fenetre = **new** JFrame ("VueFilmLabels");

fenetre.setSize (300, 300);

fenetre.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

// Creer une vue

VueFilmLabels v = **new** VueFilmLabels (m);

// Ajouter la vue dans un des conteneurs de la fenetre

fenetre.getContentPane().add(v);

// Afficher la fenetre

fenetre.setVisible(**true**);

}

**public** **void** miseAJour() {

titre.setText(m.getTitre());

annee.setText(""+m.getAnnee());

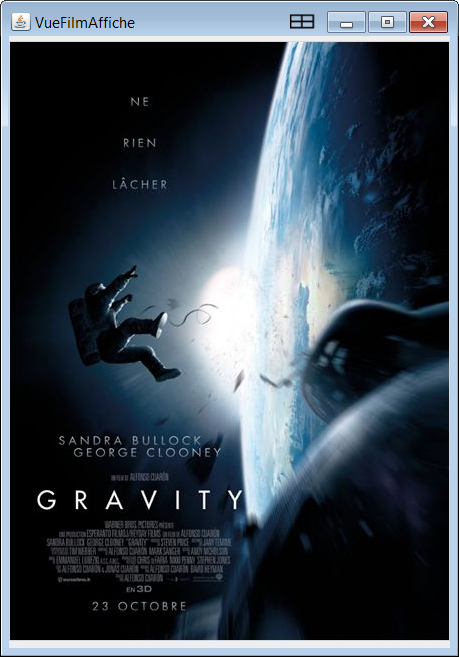
}

}

4) Est-ce possible d’avoir plusieurs vues sur un même modèle ?

=> Oui

5) Programmez une autre vue qui permet d’afficher juste l’image de l’affiche du film (figure ci-dessous) : entête, attributs, constructeur, et main() pour tester la vue et affiche les informations ci-dessous.



**package** cineclub.vues;

**import** java.awt.GridLayout;

**import** javax.swing.\*;

**import** cineclub.modeles.ModeleFilm;

**public** **class** VueFilmAffiche **extends** JPanel **implements** InterfaceVueFilm {

**private** **static** **final** **long** *serialVersionUID* = 1L;

ModeleFilm m ;

JLabel affiche ;

**public** VueFilmAffiche(ModeleFilm m) {

**this**.m = m ;

// Ajouter les composants

affiche = **new** JLabel();

affiche.setIcon(**new** ImageIcon (m.getNomImageAffiche()));

add (affiche);

}

**public** **static** **void** main(String[] args) {

// Créer un modèle

ModeleFilm m = **new** ModeleFilm ("Gravity", 2013, "Gravity.jpg");

// Creer une fenetre

JFrame fenetre = **new** JFrame ("VueFilmAffiche");

fenetre.setSize (300, 300);

fenetre.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

// Creer une vue

VueFilmAffiche v = **new** VueFilmAffiche (m);

// Ajouter la vue dans un des conteneurs de la fenetre

fenetre.getContentPane().add(v);

// Afficher la fenetre

fenetre.setVisible(**true**);

}

**public** **void** miseAJour() {

affiche.setIcon(**new** ImageIcon (m.getNomImageAffiche()));

}

}

6) Dans quel dossier Eclipse doit se trouver l’image ?

=> Par exemple dans le dossier projet Eclipse

7) On souhaite afficher les différents vues d’un même film dans différents onglets (en anglais « TabbedPane »). On vous donne un exemple en annexe 1.   
Ecrire une classe VuesFilmTabbedPane qui affiche la vue labels dans un onglet et la vue image dans un autre onglet.

**package** cineclub.vues;

**import** javax.swing.ImageIcon;

**import** javax.swing.JFrame;

**import** javax.swing.JTabbedPane;

**import** cineclub.modeles.ModeleFilm;

**public** **class** VuesFilmTabbedPane **extends** JTabbedPane **implements** InterfaceVueFilm {

// Attributs

VueFilmAffiche va ;

VueFilmLabels vl ;

// Constructeur

VuesFilmTabbedPane (ModeleFilm m) {

// Creer une vue labels et l'ajouter dans un onglet

vl = **new** VueFilmLabels (m);

addTab("Vue Labels", vl);

// Creer une vue affiche et l'ajouter dans un onglet

va = **new** VueFilmAffiche (m);

addTab("Vue Affiche", va);

}

// Mettre à jour les deux vues

**public** **void** miseAJour() {

va.miseAJour();

vl.miseAJour();

}

**public** **static** **void** main(String[] args) {

// Créer un modèle de film

ModeleFilm m = **new** ModeleFilm ("Gravity", 2013, "Gravity.jpg");

// Creer une fenetre

JFrame fenetre = **new** JFrame ("Vues Film Onglets");

fenetre.setSize (300, 300);

fenetre.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

// Creer une vue onglets

VuesFilmTabbedPane v = **new** VuesFilmTabbedPane (m);

// Ajouter la vue dans un des conteneurs de la fenetre

fenetre.getContentPane().add(v);

// Afficher la fenetre

fenetre.setVisible(**true**);

}

}

8) Le Cine Club gère évidemmment un ensemble de séances. Chaque séance est caractérisée par un film et une heure. Est-ce correct de dire que ModeleSeance doit hériter de ModeleFilm ? Pourquoi ?

=> Non. Une séance « n’est pas » un film. Par contre une séance concerne un film, donc un des attributs de ModeleSeance sera un référence vers un ModeleFilm (ou plutôt InterfaceModeleFilm).

9) Définir la classe ModeleSeance qui permet de représenter une séance.

**package** cineclub.modeles;

**import** java.util.Date;

**public** **class** ModeleSeance {

**private** ModeleFilm modeleFilm ;

**private** **int** heure ;

**public** ModeleFilm getModeleFilm() {

**return** modeleFilm;

}

**public** **void** setModeleFilm(ModeleFilm m) {

**this**.modeleFilm = m;

}

**public** **int** getHeure() {

**return** heure;

}

**public** **void** setHeure(**int** heure) {

**this**.heure = heure;

}

**public** ModeleSeance(ModeleFilm modeleFilm, **int** heure) {

**super**();

**this**.modeleFilm = modeleFilm;

**this**.heure = heure;

}

}

10) Définir la classe VueSeance qui permet par exemple d’afficher les informations suivantes. Vous devez ré-utiliser la classe VueFilmLabels définie plus haut.



**package** cineclub.vues;

**import** java.awt.GridLayout;

**import** java.sql.Date;

**import** javax.swing.\*;

**import** cineclub.modeles.ModeleFilm;

**import** cineclub.modeles.ModeleSeance;

**public** **class** VueSeanceLabels **extends** JPanel {

ModeleSeance s ;

**public** VueSeanceLabels(ModeleSeance s) {

**this**.s = s ;

// Créer les composants

VueFilmLabels vueFilm = **new** VueFilmLabels (s.getModeleFilm());

JLabel heure = **new** JLabel (""+s.getHeure()+":00");

// Ajouter les composants

GridLayout gl = **new** GridLayout (2, 1);

setLayout (gl);

add (heure);

add (vueFilm);

}

**public** **static** **void** main(String[] args) {

// Créer un modèle de film et un modele de seance pour ce film

ModeleFilm m = **new** ModeleFilm ("Gravity", 2013, "Gravity.jpg");

ModeleSeance s = **new** ModeleSeance (m, 19);

// Creer une fenetre

JFrame fenetre = **new** JFrame ("VueSeanceLabels");

fenetre.setSize (300, 300);

fenetre.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

// Creer une vue

VueSeanceLabels v = **new** VueSeanceLabels (s);

// Ajouter la vue dans un des conteneurs de la fenetre

fenetre.getContentPane().add(v);

// Afficher la fenetre

fenetre.setVisible(**true**);

}

}

11) Une interface InterfaceModeleSeances définit les méthodes qui gèrent les séances de la semaine en cours.  
Une interface InterfaceVueResultatsRecherche définit les méthodes pour visualiser les résultats.   
Définissez ces deux interfaces.

A INSERER

**package** cineclub.modeles;

**import** java.util.ArrayList;

**public** **interface** InterfaceModeleSeances {

**void** ajouterSeance (ModeleSeance s);

ArrayList <ModeleSeance> rechercheSeance (String titreFilm);

ArrayList <ModeleSeance> rechercheSeance (**int** heureSeance);

}

**package** cineclub.vues;

**import** java.util.ArrayList;

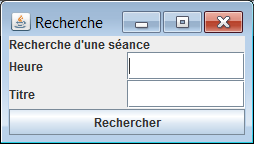
**import** cineclub.modeles.ModeleSeance;

**public** **interface** InterfaceVueResultatsRecherche {

**void** afficherResultatsRecherche (ArrayList <ModeleSeance> resultats);

}

Programmer l’interface graphique qui permet de rechercher les séances film par un mot du titre. On Il s’agit juste de définir l’interface graphique qui permet de saisir les critères de recherche :



1. Est-ce que ce formulaire de recherche est plutôt une vue (si oui sur quel modèle) ou un contrôleur (si oui qui mets en relation quel modèle et quelle vue) ? Justifiez votre réponse.

=> Il s’agit plutôt d’un contrôleur puisqu’il ne représente pas de vue sur un film ou une séance particulière : il sert juste à saisir des informations qui permettront ensuite la recherche.

1. Donner le code Java de la classe qui permet d’afficher le formulaire ci-dessus. Vous devez donner les instructions (éventuellement d’autres classes) servant à gérer les événements associés. La recherche devra être lancée dès qu’un retour chariot est tapé dans un des champs de texte ou que le bouton « Rechercher » est cliqué. Vous devez appeler des méthodes définies dans les interfaces InterfaceModeleSeances et InterfaceVueResultatsRecherche.

**package** cineclub.controleurs;

**import** java.awt.\*;

**import** java.awt.event.\*;

**import** java.util.ArrayList;

**import** javax.swing.\*;

**import** cineclub.modeles.InterfaceModeleSeances;

**import** cineclub.modeles.ModeleSeance;

**import** cineclub.modeles.ModeleSeances;

**public** **class** FormulaireRecherche **extends** JPanel **implements** ActionListener {

// Composants

JButton rechercher ;

JTextField saisieHeure ;

JTextField saisieTitre ;

InterfaceModeleSeances seances ;

// Constructeur

FormulaireRecherche (InterfaceModeleSeances seances) {

**this**.seances = seances ;

setLayout (**new** BorderLayout());

add (BorderLayout.*NORTH*, **new** JLabel ("Recherche d'une séance"));

JPanel saisie = **new** JPanel ();

saisie.setLayout(**new** GridLayout(2,2));

saisie.add (**new** JLabel ("Heure"));

saisieHeure = **new** JTextField (5);

saisie.add (saisieHeure);

saisie.add (**new** JLabel ("Titre"));

saisieTitre = **new** JTextField (10);

saisie.add (saisieTitre);

add (BorderLayout.*CENTER*, saisie);

rechercher = **new** JButton ("Rechercher");

add (BorderLayout.*SOUTH*, rechercher);

// Ajouter les écouteurs

rechercher.addActionListener (**this**);

saisieHeure.addActionListener (**this**);

saisieTitre.addActionListener (**this**);

}

**public** **static** **void** main(String[] args) {

// Creer une fenetre

JFrame fenetre = **new** JFrame ("Recherche");

fenetre.setSize (300, 300);

fenetre.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

// Creer un formulaire de recherche

ModeleSeances seances = **new** ModeleSeances();

FormulaireRecherche f = **new** FormulaireRecherche (seances);

// Ajouter le label dans un des conteneurs de la fenetre

fenetre.getContentPane().add(f);

// Afficher la fenetre

fenetre.setVisible(**true**);

}

**public** **void** actionPerformed(ActionEvent arg0) {

// Lancer la recherche

**if** (arg0.getSource () == saisieTitre) {

String titre = saisieTitre.getText ();

ArrayList <ModeleSeance> res = seances.rechercheSeance (titre);

}

// Faire de même pour le champ de texte

}

}

12) On vous donne en annexe un exemple d’utilisation de JRadioButton. Ajouter maintenant dans l’affichage du panneau de recherche un bouton radio pour que l’utilisateur puisse indiquer s’il souhaite visualiser les résultats triés par titre de film ou par heure de séance. Expliquez ce que vous ajouter comme méthode ou attributs et dans quelles classes et interfaces.

=> Ajouter dans le constructeur de la classe FormulaireRecherche :

// Ajout des boutons radio pour sélection affichage par titre ou par heure

ButtonGroup group = **new** ButtonGroup();

JRadioButton boutonTitre = **new** JRadioButton("titre");

JRadioButton boutonHeure = **new** JRadioButton("heure");

group.add(boutonTitre);

group.add(boutonHeure);

JPanel radioPanel = **new** JPanel(**new** GridLayout(1, 2));

radioPanel.add(boutonTitre);

radioPanel.add(boutonHeure);

add (BorderLayout.*NORTH*, radioPanel);

## Exercice 3 (2 pts)

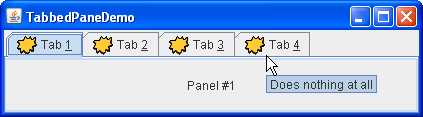
Java est un langage de programmation qui, par plusieurs aspects, facilite la ré-utilisation d’un programme développé par un programmeur par d’autres programmeurs. Nommez ces notions et expliquez comment elles permettent la ré-utilisabilité.

=> L’héritage facilite la réutilisation : un programmeur peut étendre une classe définie par un autre programmeur.

=> Les interfaces aussi. Un programmeur peut définir des interfaces qui sont ensuite implémentées par un autre programmeur.

=> La structuration en package permet aussi de regrouper des classes définies par un même programmeur et qui peuvent être utilisées sous réserve de visibilité (importées par exemple) par un autre programmeur.

## Annexe 1 : Onglets



The following code from [TabbedPaneDemo.java](http://docs.oracle.com/javase/tutorial/uiswing/examples/components/TabbedPaneDemoProject/src/components/TabbedPaneDemo.java) creates the tabbed pane in the previous example. Note that no event-handling code is necessary. The JTabbedPane object takes care of mouse and keyboard events for you.

JTabbedPane tabbedPane = new JTabbedPane();

ImageIcon icon = createImageIcon("images/middle.gif");

JComponent panel1 = makeTextPanel("Panel #1");

**tabbedPane.addTab("Tab 1", icon, panel1,**

**"Does nothing");**

tabbedPane.setMnemonicAt(0, KeyEvent.VK\_1);

JComponent panel2 = makeTextPanel("Panel #2");

**tabbedPane.addTab("Tab 2", icon, panel2,**

**"Does twice as much nothing");**

tabbedPane.setMnemonicAt(1, KeyEvent.VK\_2);

JComponent panel3 = makeTextPanel("Panel #3");

**tabbedPane.addTab("Tab 3", icon, panel3,**

**"Still does nothing");**

tabbedPane.setMnemonicAt(2, KeyEvent.VK\_3);

JComponent panel4 = makeTextPanel(

"Panel #4 (has a preferred size of 410 x 50).");

panel4.setPreferredSize(new Dimension(410, 50));

**tabbedPane.addTab("Tab 4", icon, panel4,**

**"Does nothing at all");**

tabbedPane.setMnemonicAt(3, KeyEvent.VK\_4);

As the previous code shows, the addTab method handles the bulk of the work in setting up a tab in a tabbed pane. The addTab method has several forms, but they all use both a string title and the component to be displayed by the tab. Optionally, you can specify an icon and [tool tip](http://docs.oracle.com/javase/tutorial/uiswing/components/tooltip.html) string. The text or icon (or both) can be null. Another way to create a tab is to use the insertTab method, which lets you specify the index of the tab you're adding. Note that the addTab method does not allow index specification in this step.

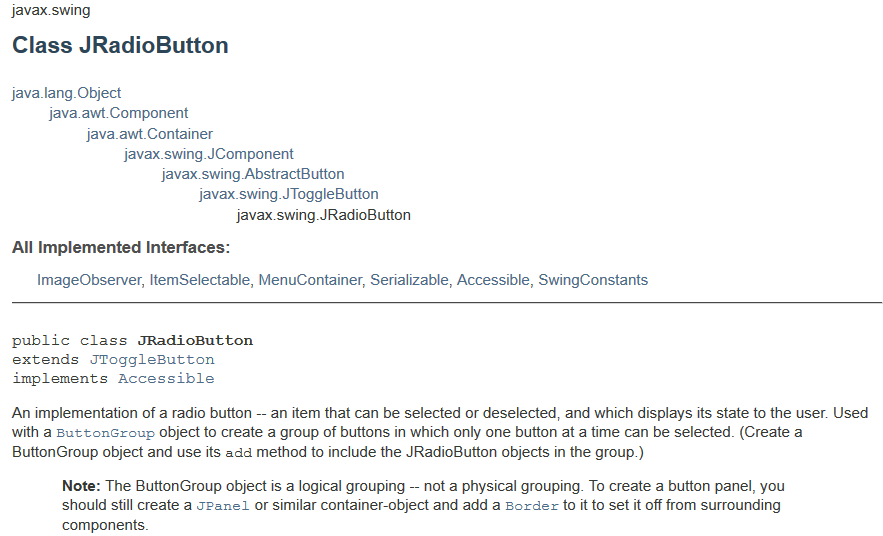
## To Switch to Specific Tabs

There are three ways to switch to specific tabs using GUI.

1. **Using a mouse.** To switch to a specific tab, the user clicks it with the mouse.
2. **Using keyboard arrows.** When the JTabbedPane object has the focus, the keyboard arrows can be used to switch from tab to tab.
3. **Using key mnemonics.** The setMnemonicAt method allows the user to switch to a specific tab using the keyboard. For example, setMnemonicAt(3, KeyEvent.VK\_4) makes '4' the mnemonic for the fourth tab (which is at index 3, since the indices start with 0); pressing Alt-4 makes the fourth tab's component appear. Often, a mnemonic uses a character in the tab's title that is then automatically underlined.

(extrait de <http://docs.oracle.com/javase/tutorial/uiswing/components/tabbedpane.html>)

## Annexe 2 : Boutons radio



… (extrait de <http://docs.oracle.com/javase/tutorial/displayCode.html?code=http://docs.oracle.com/javase/tutorial/uiswing/examples/components/RadioButtonDemoProject/src/components/RadioButtonDemo.java>)

    public RadioButtonDemo() {

        super(new BorderLayout());

        //Create the radio buttons.

        JRadioButton birdButton = new JRadioButton(birdString);

        birdButton.setActionCommand(birdString);

        birdButton.setSelected(true);

        JRadioButton catButton = new JRadioButton(catString);

        catButton.setActionCommand(catString);

        JRadioButton dogButton = new JRadioButton(dogString);

        dogButton.setActionCommand(dogString);

        JRadioButton rabbitButton = new JRadioButton(rabbitString);

        rabbitButton.setActionCommand(rabbitString);

        JRadioButton pigButton = new JRadioButton(pigString);

        pigButton.setActionCommand(pigString);

        //Group the radio buttons.

        ButtonGroup group = new ButtonGroup();

        group.add(birdButton);

        group.add(catButton);

        group.add(dogButton);

        group.add(rabbitButton);

        group.add(pigButton);

        //Register a listener for the radio buttons.

        birdButton.addActionListener(this);

        catButton.addActionListener(this);

        dogButton.addActionListener(this);

        rabbitButton.addActionListener(this);

        pigButton.addActionListener(this);

        //Set up the picture label.

        picture = new JLabel(createImageIcon("images/"+ birdString + ".gif"));

        //Put the radio buttons in a column in a panel.

        JPanel radioPanel = new JPanel(new GridLayout(0, 1));

        radioPanel.add(birdButton);

        radioPanel.add(catButton);

        radioPanel.add(dogButton);

        radioPanel.add(rabbitButton);

        radioPanel.add(pigButton);

        add(radioPanel, BorderLayout.LINE\_START);

        add(picture, BorderLayout.CENTER);

        setBorder(BorderFactory.createEmptyBorder(20,20,20,20));

    }

    /\*\* Listens to the radio buttons. \*/

    public void actionPerformed(ActionEvent e) {

        picture.setIcon(createImageIcon("images/"+ e.getActionCommand()+ ".gif"));

    }