

Mini Projet

L'objectif de ce mini projet est de développer une application qui implémente une mini agenda personnelle, permettant à un utilisateur d'enregistrer les évènements importants.

L'utilisateur gère au maximum 10 événements par date.

Pour gérer l'agenda il sera nécessaire de :

- Ajouter un évènement à son agenda. Pour chaque événement il faut indiquer :
 - Date
 - Intitulé
 - Heure du début
 - Heure de fin
 - Description
 - Lieu
- Consulter les évènements
- Modifier un évènement
- Supprimer un évènement

L'agenda doit permettre d'afficher les jours de la semaine, en indiquant ceux qui correspondent aux week-ends.

Une interface utilisateur simple et facile à utiliser doit être développée

Il est proposé d'utiliser le langage Java et son *framework* **JavaSwing** pour l'interface utilisateur

ÉTAPE 1 : modélisation UML

Utilisez la notation UML pour modéliser l'agenda.

ÉTAPE 2 : Implémentation de la classe métier pour la gestion des Dates

Définissez et testez la classe Java pour représenter les dates (class **Date**).

- Les attributs
- Les constructeurs
 - Crée une Date à partir de la date d'aujourd'hui.
 - Crée une Date à partir d'un jour, un mois et une année donnés.
- Les méthodes
 - Accéder les champs de la classe
 - Modifier les champs de la classe.
- Écrire une méthode permettant de connaître le dernier jour d'un mois. Attention aux années bissextiles.

dernierJourDuMois (**int** parMois, **int** parAnnee)

- Écrire une méthode toString() qui retourne une chaîne de caractères représentant la date (ex : « Lundi 25 octobre 2021 »)

Java fournit quelques classes permettant le traitement de dates. L'une de ces classes est « **GregorianCalendar** » laquelle représente un calendrier solaire de 12 mois de 28 à 31 jours chacun. La classe contient des attributs représentant le jour, le mois, l'année, la semaine dans l'année, etc.

La classe fournit un constructeur qui crée une instance de la classe `GregorianCalendar` à partir de la date d'aujourd'hui.

Exemple : `GregorianCalendar aujourdHui = new GregorianCalendar();`

La classe fournit également des constructeurs pour créer un calendrier à partir d'une date donnée.

Parmi les méthodes de la classe se trouvent :

- `get(GregorianCalendar.MONTH)`
- `get(GregorianCalendar.DAY_OF_MONTH)`
- `get(GregorianCalendar.YEAR)`

ÉTAPE 3 : Implémentation de la classe métier pour la gestion des Évènements

Écrire une classe **Evenement** qui permet de représenter toutes les informations propres à un événement.

Cette classe doit contenir les différents constructeurs, ainsi que les méthodes permettant consulter et modifier les informations des événements.

ÉTAPE 4 : Implémentation de la classe métier pour la gestion de l'Agenda

Écrire une classe **Agenda** qui permet de regrouper l'ensemble d'évènements

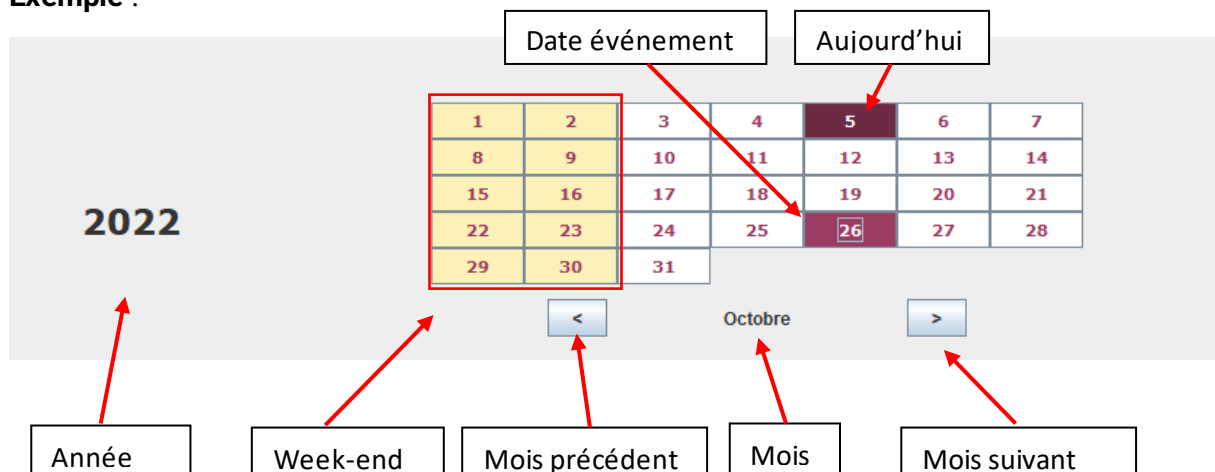
La classe **Agenda** contient des méthodes pour ajouter ou supprimer un événement




L'interface utilisateur

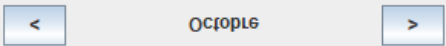


L'application à développer doit proposer une interface simple à utiliser et qui permette de réduire autant que possible la saisie manuelle des informations.

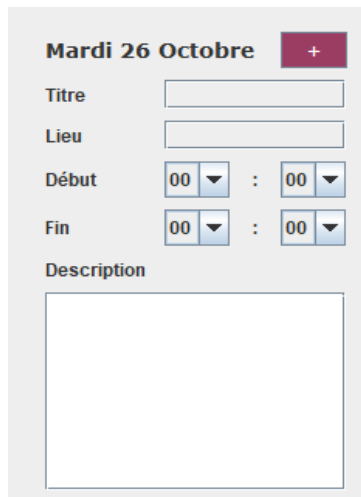
Le calendrier peut être affiché sous forme d'une grille avec tous les jours du mois choisi.

Exemple :



Dans l'exemple, la journée courante et la journée de l'évènement à enregistrer sont indiquées avec des couleurs différentes (dans l'exemple la date en cours est le 5 octobre 2022,  la date choisie pour l'évènement est le 26 octobre 2022 ). Les jours correspondants aux week-ends sont également indiqués avec une autre couleur  . Le nom du mois est aussi affiché. L'utilisateur pourra naviguer dans le calendrier afin de changer de mois.


 , grâce à des boutons précédant  et suivant  .

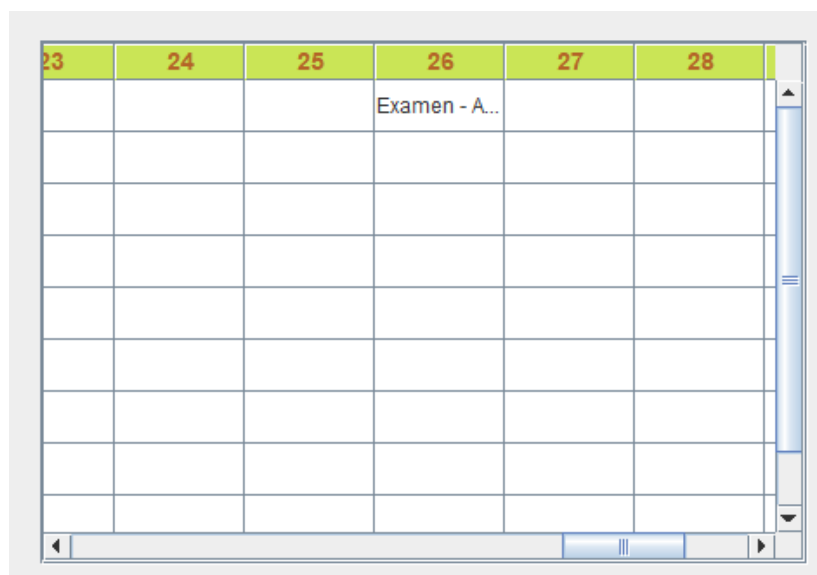


Une fois la date sélectionnée, elle est affichée dans le panel permettant la saisie de différentes informations, ensuite l'utilisateur pourra, pour l'évènement, indiquer :

- Un titre
- Le lieu
- L'heure du début
- L'heure de fin
- Optionnellement, une description

Afin de limiter les erreurs de saisie utilisez, dès que possible des listes déroulantes, tel qu'indiqué dans l'exemple ci-à-côté.

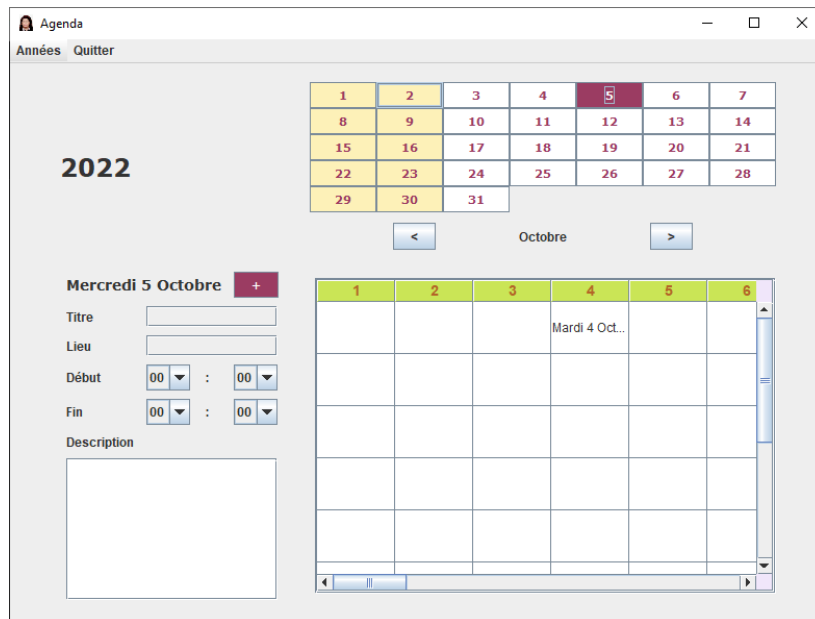
Le bouton  permettra d'ajouter l'évènement à l'agenda.



23	24	25	26	27	28
			Examen - A...		

Une fois l'évènement ajouté, les informations correspondantes à l'évènement sont affichées sur le tableau qui représente l'agenda du mois. Pour chaque mois ces évènements sont organisés par jour. L'agenda du mois est rafraîchi si l'utilisateur change de mois et choisie une nouvelle date pour un autre événement.

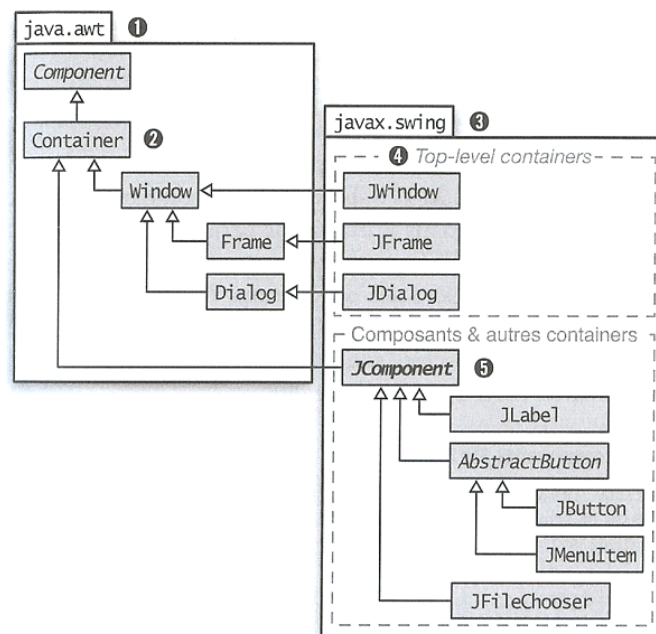
L'image ci-dessous montre un exemple de l'interface avec tous les éléments graphiques intégrés.



Java et les interfaces graphiques.

Pour l'implémentation de l'interface graphique, la bibliothèque *Swing* sera utilisée. Swing propose de nombreux composants qui permettent l'implémentation des trois aspects principaux de toute interface graphique :

- **Éléments de l'interface utilisateur** : ce sont les éléments graphiques de base que l'utilisateur voit et avec lesquels il interagit.



- **Mises en page** : ce sont des gestionnaires qui définissent la façon dont les éléments d'interface utilisateur seront organisés à l'écran et fournissent une apparence finale à l'interface. Parmi les gestionnaires disponibles dans *Swing*, se trouvent :
 - **BoxLayout** : organise les composants verticalement ou horizontalement.
 - **GroupLayout** : regroupe ses composants et les place hiérarchiquement dans un conteneur.

- **SpringLayout** : organise les éléments de son conteneur associé selon un ensemble de contraintes. Les contraintes ne sont rien d'autre qu'une distance horizontale et verticale entre deux composant.
- **CardLayout** : un seul composant est visible à la fois. Chaque composant est traité comme une carte.
- **Comportement** : il s'agit de la réponse aux événements qui se produisent dans le système.
 - **Événement** : les événements sont des actions ou des occurrences détectées par les objets (exemple : bouton, liste déroulante, etc.).

L'objet qui détecte l'événement (**Listener**) doit envoyer une notification à l'objet en charge du traitement de l'événement afin de pouvoir y répondre d'une manière ou d'une autre.

Pour recevoir des événements un objet doit s'inscrire comme écouteur auprès d'un objet source d'événement

Exemple :

```
boutonPrecedent.addActionListener(this);
```

Dans cet exemple l'objet `boutonPrecedent` est une instance de la classe `JButton` et s'inscrit comme écouteur des événements d'action sur les boutons (cliquer sur le bouton, ..).

- **Traitement** : pour traiter un événement l'objet doit implémenter les méthodes requises selon le type d'événement (Exemple : implémenter l'interface **ActionListener**).

Exemple :

```
public class PanelCalendrier implements ActionListener
```

Dans cet exemple `ActionListener` est une interface, la classe `PanelCalendrier` implémente les méthodes de l'interface.

Le tableau ci-dessous montre les différents types d'événements ainsi que les gestionnaires fournis par `Swing`.

Classes Events	Interfaces Listeners	Description
ActionEvent	ActionListener	Cette interface est utilisée pour recevoir les événements d'action.
MouseEvent	MouseListener et MouseMotionListener	Cette interface est utilisée pour recevoir les événements de souris.
KeyEvent	KeyListener	Cette interface est utilisée pour recevoir les événements des touches.
ItemEvent	ItemListener	Cette interface est utilisée pour recevoir les événements d'élément.
TextEvent	TextListener	Cette interface est utilisée pour recevoir les événements de texte.
AdjustmentEvent	AdjustmentListener	Cette interface est utilisée pour recevoir les événements d'ajustement.
WindowEvent	WindowListener	Cette interface est utilisée pour recevoir les événements de l'objet window.
ComponentEvent	ComponentListener	Cette interface est utilisée pour recevoir les événements des composants.
ContainerEvent	ContainerListener	Cette interface est utilisée pour recevoir les événements de conteneur.

ÉTAPE 1 : implémentation de l'élément graphique qui permettra de représenter le calendrier

L'objectif de cette étape est de réaliser les éléments graphiques permettant de représenter le calendrier. Il est proposé d'utiliser la métaphore d'un diaporama et de permettre la navigation. Le diaporama affichera les mois de l'année.



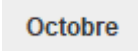
1. Écrire une classe **BoutonDate**. Cette classe est l'aspect « View » d'une date, La classe **BoutonDate** possède un champ du type **Date** et 2 champs booléens (**estDateCourante**, **estDateWeekEnd**). Ces valeurs seront utilisées pour changer la couleur des boutons
 - a. **estDateCourante** prend la valeur **true**, si la date du bouton corresponde à la date courante.
 - b. **estDateWeekEnd** prend la valeur **true**, si la date du bouton corresponde à un samedi ou à un dimanche.

Challenge : réalisez les modifications afin de permettre l'indication des jours fériés.

2. Pour implémenter le diaporama il est proposé d'utiliser la classe **JPanel**. Écrire une classe **PanelCalendrier** laquelle est une sous classe de **JPanel**.

La disposition des éléments de la classe **PanelCalendrier** sera gérée par un gestionnaire de positionnement du type **BorderLayout**.

Le **PanelCalendrier** doit contenir les éléments suivants :

- a. Au sud, un **JPanel** appelé **panelSud** contient un bouton précédent  (classe **JButton**), une étiquette (classe **JLabel**) et un bouton suivant  et le nom du mois  qui est sélectionné. Ces boutons doivent écouter les événements de clique.
- b. Au centre, un **JPanel** appelé **panelCentre** est géré par un gestionnaire de positionnement de type **CardLayout** et contient autant de panels **panelsMois** (**JPanel**) empilés que des mois dans l'année. Chaque **panelMois** est géré par un **GridLayout** et contient autant de boutons **BoutonDate** qu'il y a de jours dans le mois affiché. La grille contient 7 colonnes et autant de lignes que nécessaire (il faut instancier le **GridLayout** avec 0 pour nombre de lignes et 7 pour nombre de colonnes, sinon on ne parvient pas à forcer l'affichage avec 7 colonnes. Ex : **setLayout(new GridLayout(0,7))**).

Au lancement, le panneau du mois courant est affiché, utiliser pour cela la méthode **show ()** de la classe **CardLayout**.

Lorsque l'utilisateur clique sur un bouton, il change de couleur. Veillez à ce que le bouton qui était précédemment cliqué retrouve sa couleur d'origine.

- c. La classe **PanelCalendrier** doit contenir un tableau pour les jours de chaque mois.
- d. Associer aux boutons des jours un **ActionListener** pour qu'ils puissent réagir aux événements des cliques de l'utilisateur. À la clique sur l'un des boutons date, la chaîne correspondant à cette date doit être affichée (ex : Mercredi 5 octobre 2022).

ÉTAPE 2 : Création du Panel qui contiendra les données de chaque événement


L'objectif de cette étape est de construire un panel permettant d'afficher un formulaire pour la saisie des informations concernant un événement.

1. Écrire une classe **PanelEvenement** laquelle possède deux champs de texte (**JTextField**) correspondant à l'intitulé et au lieu de l'évènement.

Le panel contiendra également les champs nécessaires pour la saisie de l'heure de début et l'heure de fin de l'évènement (classe **JComboBox**).

Finalement une aire de texte permettra (classe **JTextArea**) la saisie d'une courte description de l'évènement.

Associer un gestionnaire de positionnement du type **GridBagLayout**.

2. Ajouter au panel un bouton  (classe **JButton**) pour ajouter les informations à l'agenda. Associer au bouton un **ActionListener** pour que le bouton puisse écouter les évènements des cliques de l'utilisateur.

ÉTAPE 3 : Création du Panel qui contiendra l'agenda du mois

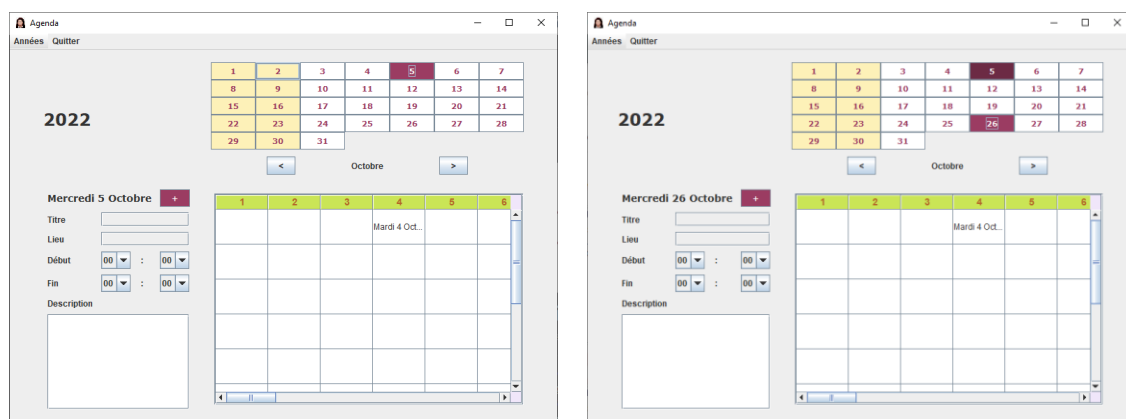
1. Écrire une classe **TableDunMois** (classe **JTable**) pour afficher les évènements du mois. L'objet **TableDunMois** sera ajouté à un **JScrollPane**.

Le composant **JTable** permet d'afficher des tables de données, éventuellement l'édition de ces données est possible. Une instance de la classe **JTable** ne contient pas les données mais les obtient à partir d'un tableau d'objets à 2 dimensions, ou à partir d'un modèle de données (classe **DefaultTableModel**). Le rendu et le mode d'édition des cellules de la table peuvent être modifiés.



2. Écrire une classe **ModeleTableDunMois** laquelle est une sous classe de **DefaultTableModel**, cette classe sera le modèle pour stocker les évènements de l'agenda, le modèle permet de stocker les informations pour un mois (maximum 10 évènements par jour). La ligne d'entête contient les différents jours du mois (de 1 au dernier jour du mois). Le modèle contiendra 10 lignes, chaque ligne correspondra à un évènement.

ÉTAPE 4 : Traitement des événements

L'objectif de cette étape est d'implémenter les éléments nécessaires à la gestion des événements de l'interface. (Ex. changer le couleur de la date sélectionnée pour un événement).





Pour gérer les événements les classes doivent implémenter l'interface correspondantes. Écrire le code nécessaire afin de traiter les différents événements.

1. Modifier la classe **PanelCalendrier** afin qu'elle puisse traiter les événements de cliques sur les boutons   . Le traitement (**ActionPerformed**) sera d'afficher le mois suivant ou le mois précédent.
2. Modifier la classe **PanelCalendrier** afin qu'elle puisse traiter les événements de cliques sur les boutons des dates. Le traitement (**ActionPerformed**) sera de changer la couleur de la date sélectionnée et d'actualiser la date sur le panel événement.

Exemple :

Si l'utilisateur clique sur le 17 Octobre  la couleur doit changer  et le panel événement affichera :

 Lundi 17 Octobre 

3. Modifier la classe **PanelEvenement** afin qu'elle puisse traiter l'événement de clique sur le bouton  . Le traitement (**ActionPerformed**) de l'évènement sera l'ajout des informations saisies à la structure de données qui représente l'agenda.