

Ressource R2.02 : Développement d'application avec IHM

JavaFX - Séance TP 3 : les formulaires

Cette séance porte sur la conception et la création de formulaires. Un formulaire est destiné à éditer les informations d'un "objet" particulier du système d'information (dossier de candidature, bon de commande, profil utilisateur, contrat de location, etc.). Un formulaire doit permettre à l'utilisateur de modifier les données d'un "objet" existant mais aussi de saisir les données d'un nouvel "objet".

Pour chaque donnée, un formulaire proposera un "champ de saisie" qui, quelque soit sa forme (champ texte, série de boutons radio, cases à cocher, etc.) permettra à l'utilisateur de saisir/modifier la valeur de la donnée.

Ce sujet présente quelques règles sur la construction d'un formulaire et indique des exemples de mise en œuvre en JavaFX. Il se base entièrement sur l'exemple de ce formulaire "papier" :

BULLETIN D'ABONNEMENT	
Nom :	Prénom :
Adresse :	
Code postal :	Ville :
Téléphone (facultatif) :	
E-mail :	
(veuillez renseigner votre adresse e-mail pour la réception de votre journal numérique)	
<input type="radio"/> Je m'abonne à l'offre DUO 1 an pour un montant de 81,40 euros (journal papier + version numérique)	
<input type="radio"/> Je m'abonne à l'offre NUMERIQUE 1 an pour un montant de 76,40 euros (version numérique)	
<input type="radio"/> Je règle par prélèvement	
<input type="radio"/> Je règle par chèque	
<input type="checkbox"/> J'accepte de recevoir des informations sur mon journal	

Le sujet est divisé en deux parties. Nous créerons d'abord sur Scene Builder l'IHM correspondant au formulaire, puis nous programmerons son comportement sur Eclipse.

PARTIE 1 : IHM liée au formulaire (Scene Builder)

1. Organisation des données du formulaire (zoning)

L'organisation des informations au sein du formulaire doit en faciliter la lecture, la compréhension et l'utilisation (GUIDAGE).

1.1. Zones de regroupement

La plupart du temps il faut structurer l'ensemble des données en plusieurs zones ou "groupes de données"¹. Il s'agit bien sûr de regrouper les données qui vont ensemble.

Les différents groupes de données doivent être mis en évidence permettant ainsi une lecture à deux niveaux. Un formulaire contenant 20 données présentées "à plat" sera moins facile à mémoriser et à utiliser qu'un formulaire constitué de quatre groupes (1^{er} niveau) de cinq données (2^e niveau).

Proposition de regroupement des données du formulaire "papier"

The diagram shows a form titled "BULLETIN D'ABONNEMENT" with four distinct zones highlighted by red boxes and labeled on the right:

- zone "Identité"**: Contains fields for Nom, Prénom, Adresse, Code postal, Ville, Téléphone (facultatif), and E-mail (with a note: "veuillez renseigner votre adresse e-mail pour la réception de votre journal numérique").
- zone "Abonnement"**: Contains two radio button options: "Je m'abonne à l'offre DUO 1 an pour un montant de 81,40 euros (journal papier + version numérique)" and "Je m'abonne à l'offre NUMERIQUE 1 an pour un montant de 76,40 euros (version numérique)".
- zone "Règlement"**: Contains two radio button options: "Je règle par prélèvement" and "Je règle par chèque".
- zone "Acceptation"**: Contains a checkbox and the text "J'accepte de recevoir des informations sur mon journal".

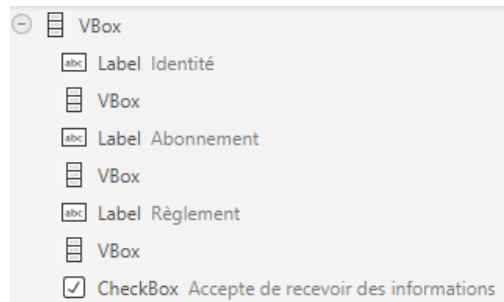
C'est ce "découpage" que nous allons adopter dans notre IHM.

Note : le texte d'entête "BULLETIN D'ABONNEMENT" pourra être inséré directement en haut de l'IHM ou bien il pourra constituer le titre de la future fenêtre.

¹ On estime qu'au-delà de 7 données, il est faut organiser les informations en les regroupant par "thème".

À FAIRE

Sur Scene Builder, créez une nouvelle IHM avec ce Scene graph :



Ajoutez les valeurs de "Padding", de "Spacing" et de "Margin" qui vous semblent nécessaires.

Note : la dernière zone n'est constitué que d'une case à cocher, inutile de créer une VBox.

1.2. Mise en évidence des zones

Les zones sont la première chose que l'utilisateur doit voir quand le formulaire apparaît à l'écran. Pour améliorer sa lisibilité, on peut mettre ces zones en évidence.

Exemple 1 : mise en évidence par l'ajout d'une bordure

Identité

Exemple 2 : mise en évidence par une couleur de fond

Identité

Note : retrouvez la liste des couleurs disponibles sur :

<https://docs.oracle.com/javase/8/javafx/api/javafx/scene/paint/Color.html>

À FAIRE

Faites en sorte que les zones du formulaire soient mises en évidence par le moyen de votre choix.

1.3. Découper un formulaire en plusieurs parties

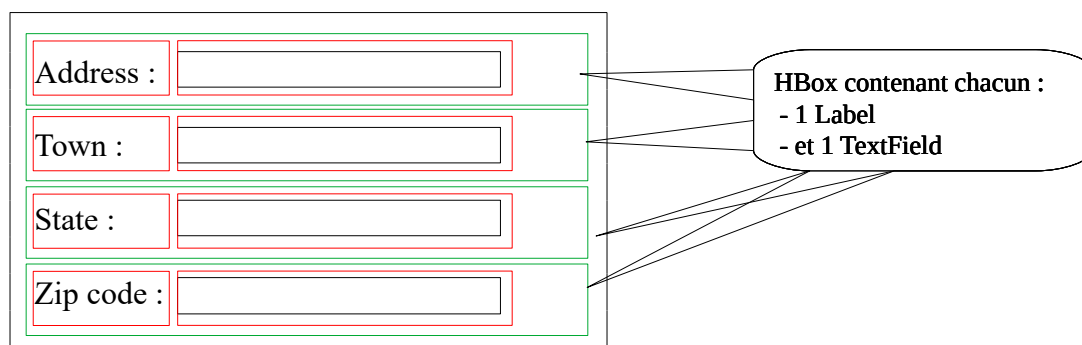
Lorsque le formulaire est très dense², il peut être intéressant de le découper en plusieurs parties et ne présenter à l'utilisateur qu'une partie à la fois. Les onglets se prêtent très bien à ce découpage. Pour cela, consultez la documentation JavaFX sur la classe **TabPane**.

1.4. Aligner les champs de saisie

1.4.1. Alignement des composants à gauche

Il est d'usage d'aligner les champs de saisie à gauche. En JavaFX, certains Layout Pane s'y prêtent plus facilement que d'autres.

Exemple avec des HBox



Pour créer un alignement à gauche des TextField, il faut donner la même taille aux Label.

Exemple avec un GridPane

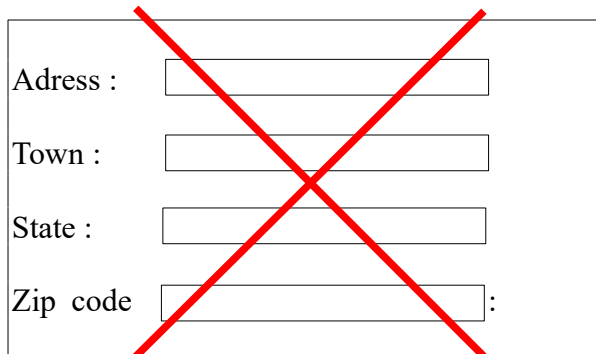
Address :	<input type="text"/>
Town :	<input type="text"/>
State :	<input type="text"/>
Zip code :	<input type="text"/>

Avec un GridPane les composants sont naturellement alignés à gauche (par défaut).

² On estime qu'au delà de 30 données, il est judicieux de décomposer le formulaire en plusieurs parties. Ce n'est pas le cas avec le formulaire d'abonnement.

1.4.2. Alignement des composants à droite

Pour éviter un "effet d'escalier", certains formulaires alignent aussi les champs texte à droite, ce qui revient finalement à leur donner à tous la même taille. Par exemple :



Adress :

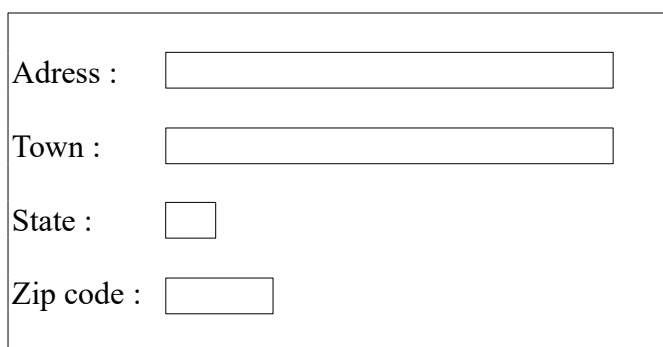
Town :

State :

Zip code :

The image shows a form with four rows. Each row has a label on the left and a text input box on the right. The labels are 'Adress', 'Town', 'State', and 'Zip code'. The input boxes have different widths: 'Adress' is the widest, followed by 'Town', then 'State', and 'Zip code' is the narrowest. A large red 'X' is drawn over the entire form, indicating that this layout is incorrect.

Or la taille d'un champ texte doit fournir une indication sur la taille maximale autorisée. Dans l'exemple précédent, difficile de deviner que le champ *State* n'accepte que deux lettres et que le champ *Zip code* n'accepte que cinq caractères. On préférera ce deuxième exemple qui offre un meilleur GUIDAGE :



Adress :

Town :

State :

Zip code :

The image shows a form with four rows. Each row has a label on the left and a text input box on the right. The labels are 'Adress', 'Town', 'State', and 'Zip code'. The input boxes have different widths: 'Adress' is the widest, followed by 'Town', then 'State', and 'Zip code' is the narrowest. This layout is presented as a better example for providing guidance.

1.5. Cheminement en "Z"

Traditionnellement dans un formulaire, l'utilisateur doit pouvoir passer d'un champ de saisie à un autre en utilisant la touche Tab. Il faut alors veiller à ce que ce cheminement respecte le sens de lecture : de gauche à droite puis de haut en bas. On parle de "cheminement en Z".

En JavaFX, le cheminement dépend de l'ordre dans lequel les composants ont été ajoutés à leur "parent" dans le *Scene graph*.

1.6. Compatibilité

Le critère ergonomique de COMPATIBILITÉ nous invite à respecter la représentation mentale qu'a l'utilisateur du "document". À l'écran, le formulaire doit se rapprocher le plus possible de l'image que l'utilisateur a du document (cf. image opérative).

Par exemple si le formulaire doit représenter un objet du monde réel (dossier de candidature, bon de commande, bulletin d'abonnement, etc.) alors sa forme à l'écran doit être le plus fidèle possible à cet objet .

À FAIRE

Créez le contenu de la première zone "Identité" par une série de HBox :

Identité

Nom :	<input type="text"/>	Prénom :	<input type="text"/>
Adresse :	<input type="text"/>		
	<input type="text"/>		
Code postal :	<input type="text"/>	Ville :	<input type="text"/>
Téléphone :	<input type="text"/>		
E-mail	<input type="text"/>		

Adaptez la largeur des différents éléments.

2. Aide à la saisie

Un des objectifs quand on conçoit un formulaire est de le rendre efficace, c'est-à-dire facile et rapide à comprendre et à utiliser. Tout doit être mis en œuvre pour que l'utilisateur comprenne rapidement le rôle exact de chaque donnée (GUIDAGE) et puisse la saisir/modifier le plus facilement et rapidement possible (CONCISION).

2.1. Précisions sur chaque donnée

Chaque champ de saisie doit être accompagné d'un libellé qui indique le rôle de cette donnée. Le libellé peut être positionné au-dessus ou à gauche du champ de saisie (sauf pour les cases à cocher et les boutons radio).

Nom :	<input type="text"/>	Nom :	<input type="text"/>
-------	----------------------	-------	----------------------

Libellé et champ peuvent être disposés horizontalement ou verticalement.

Le libellé peut parfois se trouver dans le champ de saisie (il est alors remplacé par la frappe de l'utilisateur) :

Je crée un compte pour m'abonner	
Prénom	<input type="text"/>
Nom	<input type="text"/>
Téléphone	<input type="text"/>

Mais il est parfois nécessaire de fournir des précisions supplémentaires à l'utilisateur : nature exacte de la donnée, caractère obligatoire ou facultatif, longueur maximale, format de la valeur à saisir, unité dans laquelle la valeur est exprimée, etc. C'est ce que montrent les paragraphes qui suivent.

2.1.1. Insérer des commentaires dans le formulaire

Il ne faut pas hésiter à insérer des commentaires dans la fenêtre, à proximité de la donnée, comme sur ces quelques exemples :

Indice :	<input type="text"/>
Voir sur votre bulletin de salaire	

pour aider à retrouver la valeur

Code PDT :	<input type="text"/>
Numéro à 5 chiffres	

pour préciser la longueur

Date :	<input type="text"/>
(jj/mm/aaaa)	

pour préciser le format attendu

Date :	<input type="text"/>
(ex : 02/05/2023)	

pour préciser le format attendu

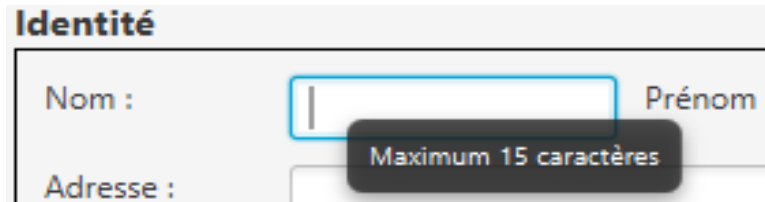
Poids du produit :	<input type="text"/>	Kg
--------------------	----------------------	----

pour préciser l'unité

2.1.2. Les bulles d'aide

Une bulle d'aide contient un message explicatif qui apparaît quand la souris survole le champ de saisie. L'avantage d'une bulle d'aide est de ne pas surcharger le formulaire car l'explication n'apparaît que sur sollicitation de l'utilisateur.

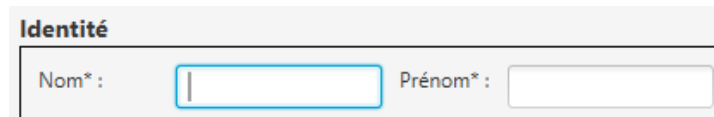
Exemple :

Un formulaire intitulé "Identité" avec des champs pour "Nom :", "Prénom :", et "Adresse :". Le champ "Nom :" est sélectionné, et une bulle d'aide noire avec le texte "Maximum 15 caractères" apparaît au-dessus de lui.

Nous verrons dans la partie 2 comment ajouter en JavaFX des bulles d'aide sur des composants.

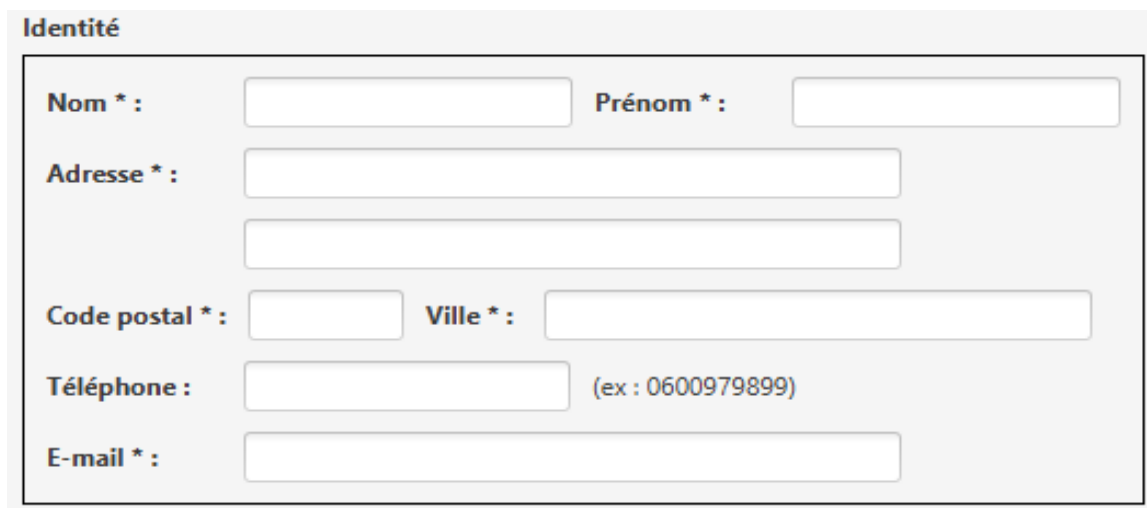
2.1.3. Signaler les données obligatoires

Dans la grande majorité des interfaces, le caractère obligatoire d'une donnée est signalé par un **astérisque**. Pour des raisons d'homogénéité, il est conseillé de reprendre ce principe.

Un formulaire intitulé "Identité" avec des champs pour "Nom*" et "Prénom*". Les champs sont vides et ont une bordure bleue.

À FAIRE

*Complétez l'IHM en ajoutant * aux champs obligatoires (tous sauf le téléphone) et indiquez de quelle façon le numéro de téléphone doit être saisi :*

Un formulaire intitulé "Identité" avec des champs pour "Nom *", "Prénom *", "Adresse *", "Code postal *", "Ville *", "Téléphone", "E-mail *". Les champs "Nom *", "Prénom *", "Adresse *", "Code postal *", "Ville *", et "E-mail *" ont une bordure grise. Le champ "Téléphone" a une bordure grise et un exemple de numéro "(ex : 0600979899)".



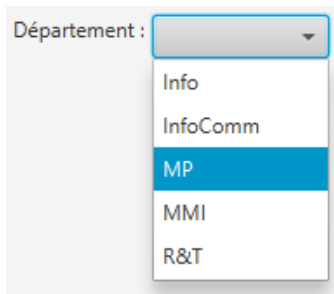
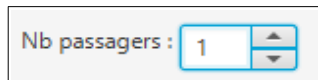
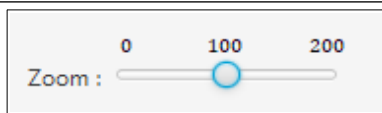
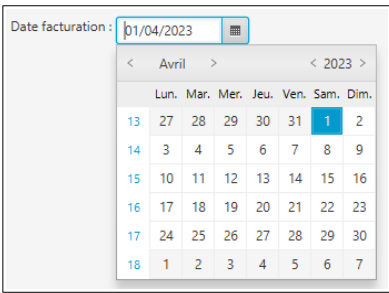
Vous vérifierez que le "cheminement en Z" est cohérent.

2.2. Faciliter la saisie

L'idée générale ici est de minimiser les saisies clavier. Non seulement l'utilisateur gagnera du temps mais il limitera aussi les erreurs de frappe. Donc, à chaque fois que c'est possible, il faudra délaissier le champ texte (*TextField*) au profit d'autres composants plus adaptés.

2.2.1. Choix du composant

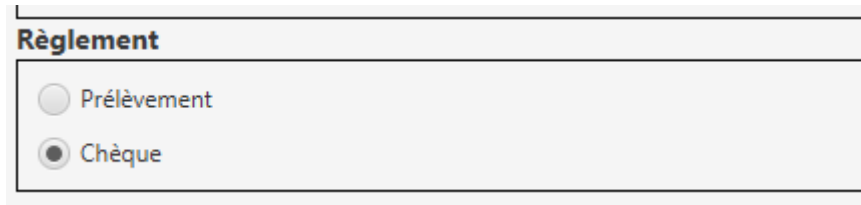
Ce tableau présente quelques exemples de composant possible en fonction du type de la donnée, comme alternative au champ texte.

Type de la donnée	Composant possible	Exemple
booléen	case à cocher (<i>CheckBox</i> ³)	
énuméré (pas plus de 4 valeurs)	série de boutons radio (<i>RadioButton</i> ³)	
énuméré (au-delà de 4 valeurs)	liste déroulante (<i>ComboBox</i> ³)	
énuméré (valeurs numériques et nombre de valeurs restreint)	toupie (<i>Spinner</i> ³)	
énuméré (valeurs numériques et nombre de valeurs restreint)	curseur (<i>Slider</i> ³)	
date	Calendrier (<i>DatePicker</i> ³)	

³ Consultez la documentation sur ces classes.

2.2.2. Valeurs par défaut

À chaque fois que c'est possible, il faut proposer une valeur par défaut à l'utilisateur. Cette valeur par défaut doit être la valeur qui a la plus grande probabilité d'être saisie/choisie. Dans l'exemple du formulaire d'abonnement, s'il s'avère que la majorité des clients choisissent de payer par chèque, c'est ce mode de règlement qui doit être proposé par défaut :

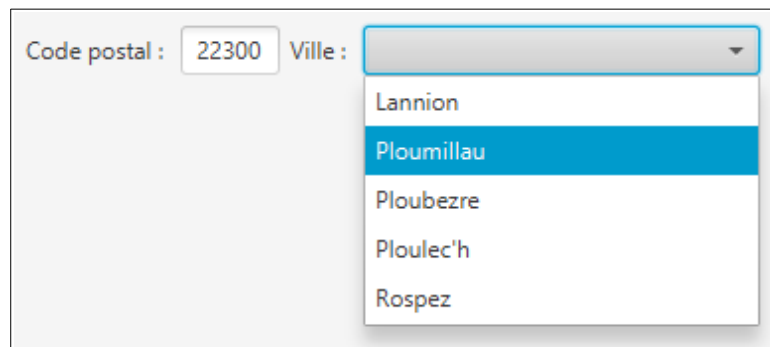


Règlement

☐ Prélèvement

☒ Chèque

Autre exemple : une fois que l'utilisateur a saisi le code postal, il serait judicieux de lui proposer la liste des quelques villes associées. Cela suppose bien sûr de connaître, pour chaque code postal, la liste des communes correspondantes⁴.



Code postal : 22300 Ville :

- Lannion
- Ploumillau
- Ploubezre
- Ploulec'h
- Rospez

⁴ Nous ne mettrons pas ce dispositif en œuvre ici.

À FAIRE

Complétez votre IHM par le contenu des zones Abonnement et Règlement :

Identité			
Nom * :	<input type="text"/>	Prénom * :	<input type="text"/>
Adresse * :	<input type="text"/>		
	<input type="text"/>		
Code postal * :	<input type="text"/>	Ville * :	<input type="text"/>
Téléphone :	<input type="text"/>	(ex : 0600979899)	
E-mail * :	<input type="text"/>		
Abonnement			
<input type="radio"/> Formule DUO (81,40 euros)			
<input checked="" type="radio"/> Formule NUMERIQUE (76,40 euros)			
Règlement			
<input type="radio"/> Prélèvement			
<input checked="" type="radio"/> Chèque			
<input type="checkbox"/> Accepte de recevoir des informations			

3. Les boutons du formulaire

En règle générale, l'apparition d'un formulaire à l'écran fait suite au lancement d'une action de l'utilisateur (créer nouvelle inscription, modifier profil, éditer dossier, etc.). Le formulaire, qui représente la réponse à cette demande d'action, doit fournir à l'utilisateur la possibilité de valider l'action ou de l'annuler. C'est pourquoi un formulaire contiendra au moins deux boutons :

- l'un pour valider l'action en cours (exemples : "OK", "Enregistrer", "Valider", etc.),
- l'autre pour l'annuler (par exemple "Annuler" ou parfois "Abandonner").

Ces boutons, qui sont utilisés en fin de saisie ou modification, doivent se trouver "à la fin" du formulaire, c'est-à-dire en bas et/ou à droite de la fenêtre.

3.1. Bouton par défaut

Quand une fenêtre contient plusieurs boutons, l'un d'eux peut être choisi comme **bouton par défaut**. Cela signifie que le bouton peut être activé par l'appui sur la touche *Entrée*.

Note : il ne faut jamais choisir une action lourde de conséquences comme action par défaut. Par exemple, on évitera de choisir "Supprimer" ou "Réinitialiser" comme bouton par défaut.

3.2. Bouton d'annulation

Quand une fenêtre contient plusieurs boutons, l'un d'eux peut être choisi comme **bouton d'annulation**, ce qui signifie qu'il pourra simplement être activé par l'appui sur la touche *Echap*. Dans un formulaire, c'est bien sûr le bouton Annuler qui aura cette caractéristique.

À FAIRE

Ajoutez en fin de formulaire une nouvelle HBox qui contient les deux boutons. Le bouton OK est le bouton par défaut et le bouton Annuler est le bouton d'annulation.

PARTIE 2 : comportement du formulaire (JavaFX)

À FAIRE

Sur Eclipse, créez une nouvelle classe qui crée une fenêtre correspondant à l'IHM de la partie 1. Son titre sera "Bulletin d'abonnement".
Créez ensuite une autre classe qui sera le contrôleur de la fenêtre.

4. Caractéristiques de la fenêtre

4.1. Fenêtre non redimensionnable

Un formulaire est assez rarement redimensionnable ; il faut explicitement l'indiquer :

Code JavaFX : `.setResizable(false);`

4.2. Modalité

Une fenêtre est **modale** quand elle empêche l'utilisateur d'accéder à d'autres parties de l'application tant qu'il n'a pas fermé cette fenêtre. L'utilisateur doit impérativement répondre à la sollicitation avant de pouvoir poursuivre. Par exemple, les messages d'erreur apparaissent dans des fenêtres modales.

Les formulaires sont le plus souvent des fenêtres modales. En effet, il serait peu judicieux de permettre à l'utilisateur de commencer à saisir un nouveau dossier et de le laisser faire autre chose au risque qu'il oublie de compléter le dossier.

Code JavaFX : `this.initModality(Modality.APPLICATION_MODAL);`

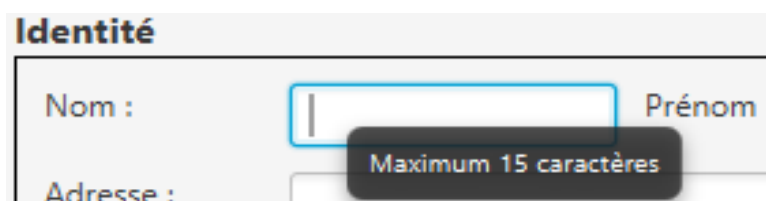
À FAIRE

Faites en sorte que la fenêtre "Bulletin d'abonnement" soit modale et non redimensionnable.

5. Bulles d'aide (Tooltip)

Rappel : une bulle d'aide contient un message explicatif qui apparaît quand la souris survole le champ de saisie. L'avantage d'une bulle d'aide est de ne pas surcharger le formulaire car l'explication n'apparaît que sur sollicitation de l'utilisateur.

Exemple :



Code JavaFX : `txtNom.setTooltip(new Tooltip("Maximum 15 caractères"));`

À FAIRE

Dans le contrôleur de la fenêtre, ajoutez la méthode `initialize()` pour affecter des bulles d'aide à quelques champs du formulaire avec des messages pertinents.

6. Prévention des erreurs

La prévention des erreurs consiste à empêcher l'utilisateur, autant que possible, de faire des erreurs de saisie (frappe d'un caractère non autorisé, frappe d'un caractère alors que le nombre maximum est atteint, etc.).

Exemple 1

Supposons que le nom de l'abonné soit limité à 15 caractères. L'IHM ne doit donc plus accepter la frappe de caractère au-delà de ce maximum.

Code JavaFX :

```
if (txtNom.getText().length() >= 16){  
    txtNom.deletePreviousChar();  
}
```

Efface le caractère
qui vient d'être frappé

Exemple 2

Certains champs de saisie sont obligatoires. Il faut donc empêcher l'utilisateur de valider le formulaire tant que toutes les données obligatoires n'ont pas été saisies : le bouton OK doit rester désactivé (grisé) tant que cette condition n'est pas satisfaite.

Code JavaFX :

```
bnOK.setDisable(true);
```

Note : nous verrons dans une prochaine séance comment rendre le bouton OK à nouveau actif.

À FAIRE

Dans le contrôleur de la fenêtre, faites en sorte que la saisie du nom et du prénom soit limitée à 15 caractères et désactivez le bouton OK.

7. Traitement des erreurs

Si l'IHM doit empêcher au maximum l'utilisateur de faire des erreurs, il faut néanmoins savoir en traiter certaines. Pour rappel, le critère ergonomique de GESTION DES ERREURS préconise que, pour chaque erreur détectée :

- la nature de l'erreur soit indiquée,
- la cause de l'erreur soit indiquée,
- un moyen d'y remédier soit proposé.

D'autre part, l'erreur doit être signalée le plus tôt possible et le plus près possible de l'endroit où elle a été détectée.

7.1. La classe **Alert**

Une première solution pour signaler une erreur à l'utilisateur est d'afficher une fenêtre popup contenant le message d'erreur. La classe **Alert** convient parfaitement pour créer ce genre de boîte à message.

Pour créer une telle popup, il faut créer et configurer une instance de la classe *Alert*, en indiquant le type de popup (**ERROR** ici), le message et les boutons qui doivent être présents sur la popup (bouton **OK** ici).

```
Alert erreur = new Alert(  
    AlertType.ERROR,  
    "Le code postal doit être sur 5 caractères. Veuillez modifier ce champ.",  
    ButtonType.OK);
```

La nature de l'erreur peut figurer dans le titre de la popup :

```
erreur.setTitle("Code postal : format incorrect");
```

Enfin, il faut afficher la popup :

```
erreur.showAndWait();
```

Il ne reste plus qu'à associer la callback au champ "Code postal" :

```
if (txtCodePostal .getText().length() != 5) {  
    e.consume();  
    Alert erreur = new Alert(AlertType.ERROR, "Le code postal doit être  
sur 5 caractères. Veuillez modifier ce champ.", ButtonType.OK);  
    erreur.setTitle("Code postal : format incorrect");  
    erreur.showAndWait();  
}
```

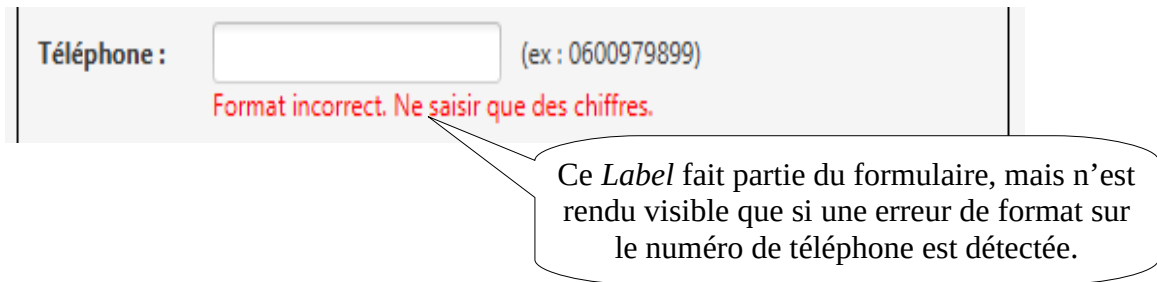
À FAIRE

Ajoutez à votre formulaire ce traitement d'erreur (dépassement de la taille max) pour quelques champs de saisie :

- les deux champs d'adresse : taille max de 38 caractères
- champ "Ville" : taille max de 32 caractères

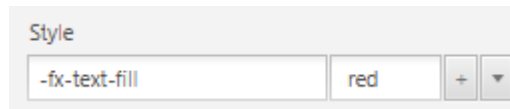
5.2 Un message d'erreur adapté pour chaque champ

Une deuxième solution consiste à prévoir les messages en insérant des messages au sein même du formulaire près de chaque champ de saisie, et à les rendre visibles quand nécessaire.



À FAIRE

Sur Scene Builder, ajoutez un Label sous le champ "Téléphone" et appliquez-lui ce style css :



Dans la callback associée à la frappe d'une touche (on Key Released) programmez ceci :

```
if (estValide(txtTelephone.getText()) ) {  
    lblErreurTelephone.setVisible(false);  
} else {  
    lblErreurTelephone.setVisible(true);  
}
```

Pour tester la validité d'un numéro de téléphone, ajoutez cette méthode à votre code JavaFX :

```
private boolean estValide(String str) {  
    return str.matches("\\d*");  
}
```

Note : évidemment, par souci d'homogénéité, le traitement des erreurs devra être uniforme dans tout le formulaire.

8. Annulation

Comme vu précédemment, tout formulaire possède au moins un bouton d'annulation. Cliquer sur le bouton *Annuler* doit simplement entraîner la fermeture de la fenêtre (dans certains cas, assez rares, une confirmation peut être demandée à l'utilisateur).

La fermeture de la fenêtre nous oblige à modifier légèrement la classe *Formulaire*. Nous allons y créer une variable de classe de type *Stage* qui représentera la fenêtre :

```
static private Stage f;
```

Dans la méthode *start()*, c'est cette variable qui est initialisée et modifiée :

```
f = new Stage();  
f.setTitle("BULLETIN D'ABONNEMENT");  
f.setResizable(false);  
f.initModality(Modality.APPLICATION_MODAL);  
etc.
```

Puis nous créerons une méthode de classe responsable de la fermeture de la fenêtre :

```
static public void fermer() {  
    f.close();  
}
```

Et dans le contrôleur, un clic sur le bouton *Annuler* déclenchera l'appel à cette méthode de classe :

```
@FXML void annuler() {  
    Formulaire.fermer();  
}
```

À FAIRE

Programmez comme indiqué le clic sur le bouton Annuler.

9. Validation du formulaire

Lors de la validation du formulaire, lors d'un clic sur le bouton *OK*, il faut :

- vérifier que les valeurs présentes dans le formulaire sont toutes valides, que la saisie est "cohérente",
- éventuellement convertir certaines valeurs du formulaire : en effet, les valeurs des composants graphiques ne coïncident pas toujours aux valeurs des attributs de l'objet⁵, et puis il faut gérer le cas des champs vides pour les données non obligatoires,
- et entraîner l'ajout ou la modification de l'objet correspondant.

À FAIRE

*Considérons ici que le formulaire soit utilisé en mode "ajout", c'est-à-dire en mode "saisie d'un nouvel abonnement". En utilisant la classe `Abonnement`, programmez le clic sur le bouton *OK* qui doit provoquer :*

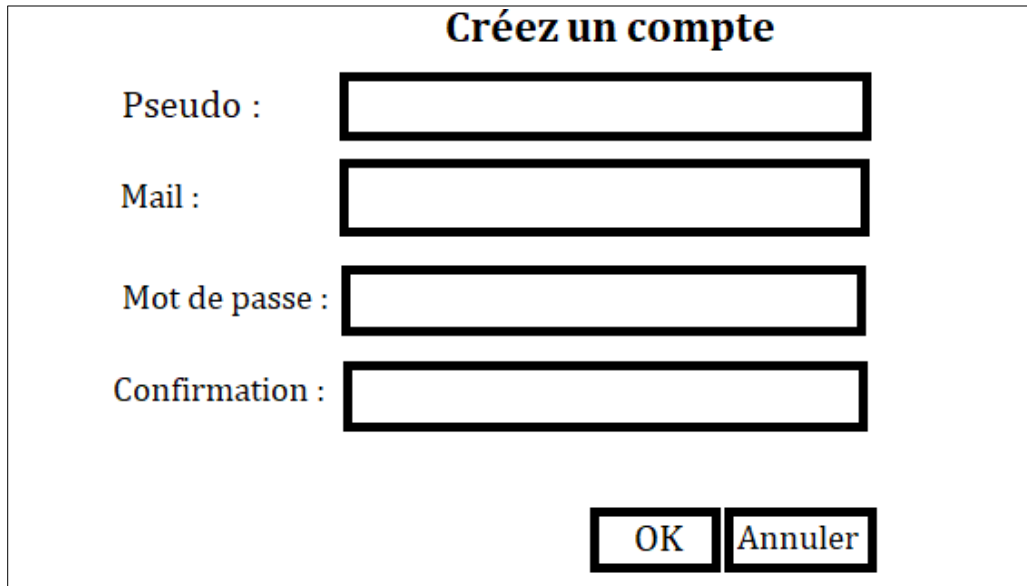
- *la vérification du champ E-mail (vérifiez que la valeur saisie contient bien un @), avec l'affichage d'une popup d'erreur si besoin,*
- *la création d'une nouvelle instance de la classe `Abonnement` (la numérotation des instances est automatique),*
- *l'affichage dans la console de cette nouvelle instance (cf. méthode `afficher()` de la classe `Abonnement`),*
- *la fermeture de la fenêtre.*

⁵ Par exemple, une variable de type `String` dans une classe peut apparaître sous la forme de plusieurs boutons radio dans le formulaire.

10. À vous de jouer...

10.1. Formulaire de création de compte

Dans cet exercice, vous devez implémenter un formulaire de création de compte dont voici une maquette :



Créez un compte

Pseudo :

Mail :

Mot de passe :

Confirmation :

Remarques

- Pour les deux derniers champs, vous pourrez utiliser le composant *PasswordField*.
- Lors de la validation du formulaire, il faudra vérifier que les deux mots de passe saisis sont identiques.
- Le bouton OK sera désactivé (grisé) tant que tous les champs n'auront pas été remplis. Trouvez un moyen de le rendre actif quand tous les champs sont remplis.

10.2. Formulaire de saisie d'une candidature

Vous êtes chargés de réaliser un formulaire de saisie d'une candidature à l'IUT. Programmez ce formulaire en tenant compte de la liste des données qui doivent y figurer.

Liste des données

Attribut	Obligatoire	Domaine de définition et commentaires
n° inscription	X	entier strictement positif, attribué automatiquement (de manière incrémentale) à chaque nouvelle inscription. Non modifiable par l'utilisateur.
nom	X	chaîne de 20 caractères maximum
prénom	X	chaîne de 20 caractères maximum
date naissance	X	date
adresse ligne1		chaîne de 38 caractères maximum
adresse ligne2		chaîne de 38 caractères maximum
code postal		chaîne de 5 caractères exactement
commune		chaîne de 32 caractères maximum
tél portable		chaîne de 10 caractères maximum
spécialité 1	X	dans {Histoire, Humanités, Langues, Maths, Physique/Chimie, SVT, SES, Bio-Eco, NSI, Arts, Littérature, Sport}
spécialité 2	X	dans {Histoire, Humanités, Langues, Maths, Physique/Chimie, SVT, SES, Bio-Eco, NSI, Arts, Littérature, Sport}
boursier	X	booléen
diplôme préparé	X	dans {BUT INFO, BUT MP, BUT R&T, BUT MMI, BUT InfoComm}
montant de l'inscription	X	valeur par défaut = 265 euros (ou bien 0 euro si boursier)
date règlement	X	date du jour par défaut

Question 1

Dessinez sur papier une maquette aussi détaillée que possible.

Question 2

Programmez ce formulaire avec Scene Builder et JavaFX.