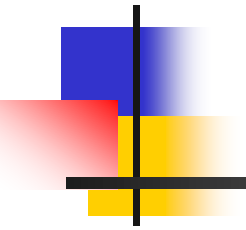
Flex

Lire les données de manière contrôlée



Plan

- Lier les données
- Stocker les données
- Valider les données

Gérer des données

- Lier des données
  - La notion de DataBinding est l'une des plus importantes du framework Flex
  - Son rôle est d'assurer la communication des données entre un objet source et un objet de destination
  - Ainsi, toute modification des données de l'objet source entraîne la modification des données de l'objet de destination

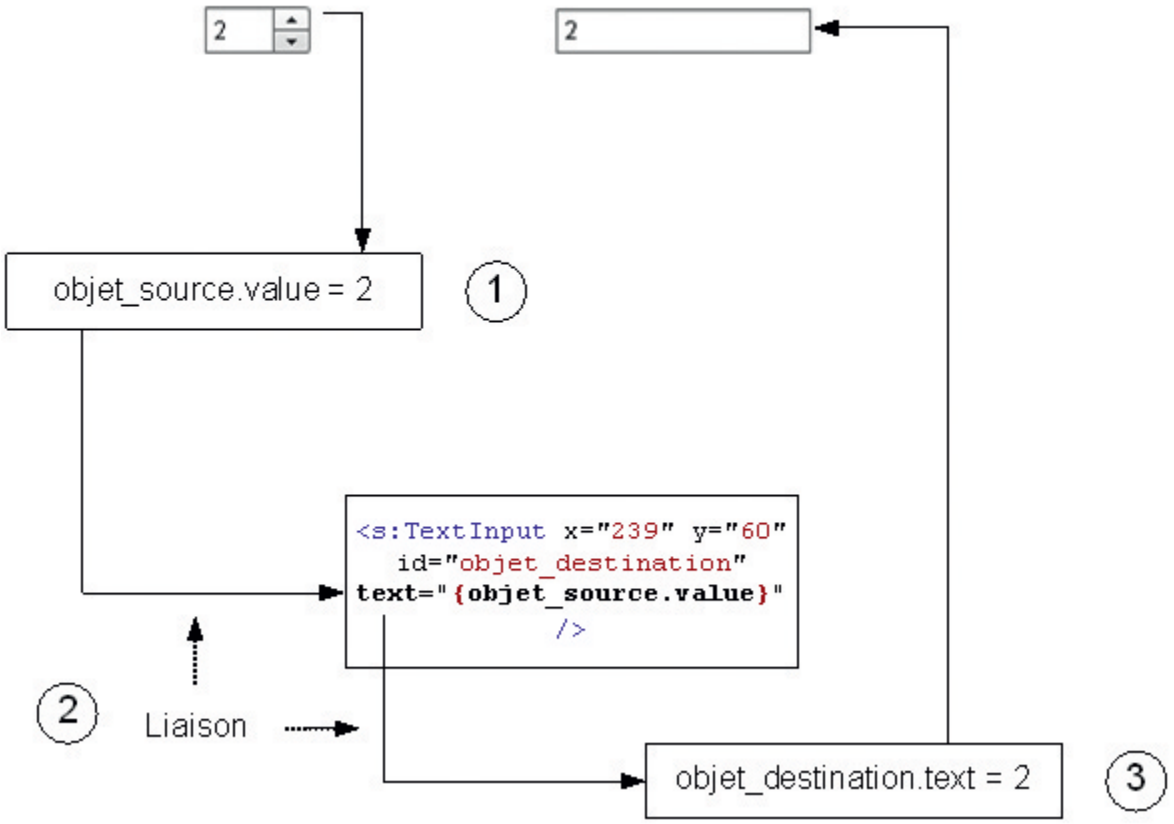
Gérer des données

- Lier des données : Exemple : DataBinding1.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   xmlns:mx="library://ns.adobe.com/flex/mx" minWidth="955"
    minHeight="600">
<fx:Declarations>
<!-- Placer ici les éléments non visuels (services et objets de
    valeur, par exemple). -->
</fx:Declarations>
<s:NumericStepper x="63" y="59" id="objet_source"/>
<s:TextInput x="142" y="60" id="objet_destination"
    text="{objet_source.value}"/>
</s:Application>
```

Lier des données : Exemple : DataBinding1.mxml

- Dans cette portion de code, deux composants sont présents :
  - le composant NumericStepper, qui représente l'objet source ;
  - le composant TextInput, qui correspond à l'objet de destination
  - La valeur mentionnée entre les accolades de la propriété text du composant TextInput précise la liaison de données



Lier des données

- DataBinding bi-directionnel
  - Permet de modifier l'élément source à partir de l'élément de destination
  - Pour activer cette fonctionnalité, il suffit d'ajouter un @ sur l'objet de destination :

```
<s:TextInput x="20" y="59" id="objet_source"/>
<s:TextInput x="179" y="60" id="objet_destination"
  text="@{objet_source.text}"/>
```

  - De ce fait, lorsque le contenu de la zone de texte objet_source sera modifié, le contenu de la zone de texte objet_destination le sera également (et vice versa)

Lier des données

- La balise <fx:Binding>
  - a pour but de remplacer la notation par accolades vue précédemment
  - Elle permet également de remplir le rôle du contrôleur dans une application ayant un type d'architecture MVC en détachant la vue du modèle et en attribuant plusieurs objets de destination à un seul objet source (nous reviendrons sur ce point un peu plus loin dans ce chapitre)
  - Voyons comment utiliser cette balise dans l'exemple de code précédent :

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application …>
<s:NumericStepper x="10" y="22" width="78" id="objet_source"/>
<s:TextInput x="234" y="22" id="objet_destination"/>
<fx:Binding
source="String(objet_source.value)"
destination="objet_destination.text">
</fx:Binding>
</s:Application>
```

- Cette balise nous permet, par l'emploi des propriétés source et destination, de mieux comprendre la notion de DataBinding
- Une conversion de type numérique vers une chaîne a été nécessaire pour que la liaison soit effective
- Sans cela, il nous était impossible d'affecter une valeur de type numérique vers une propriété de type chaîne de caractères

Lier des données

- Activer le DataBinding bi-directionnel
  - En ajoutant l'attribut twoWay affecté de la valeur true à la balise <fx :Binding>, nous activons le DataBinding bi-directionnel
  - Exemple

```
<s:TextInput x="10" y="22" width="110"
   id="objet_source"/>
<s:TextInput x="146" y="22" id="objet_destination"/>
<fx:Binding source="objet_source.text"
   destination="objet_destination.text" twoWay="true">
</fx:Binding>
```

Lier des données

- Le DataBinding en ActionScript
  - En AS, il est possible de programmer les liaisons, et de définir des variables en tant que sources de données
  - **Programmation de la liaison entre objets**
    - Utilise la classe BindingUtils
    - dans le package mx.binding.utils, met à disposition la méthode BindingUtils.bindProperty de paramètres :
      - identifiant de l'objet de destination,
      - propriété de l'objet de destination : obj de la modification
      - identifiant de l'objet source,
      - propriété de l'objet source : obj de la synchronisation

- Adaptation de l’exemple précdent : DataBindingAs

```
<s:Application xmlns:fx=”http://ns.adobe.com/mxml/2009”
xmlns:s=”library://ns.adobe.com/flex/spark”
xmlns:mx=”library://ns.adobe.com/flex/mx” minWidth=”955” minHeight=”600”
creationComplete=”creationLiaison()”>
<fx:Script>
<![CDATA[
import mx.binding.utils.BindingUtils;
public function creationLiaison():void{
BindingUtils.bindProperty(objet_destination, "text", objet_source, "value");
}
]]>
</fx:Script>
<s:NumericStepper x="10" y="22" width="78" id="objet_source"/>
<s:TextInput x="234" y="22" id="objet_destination"/>
</s:Application>
```

- Remarque
  - La programmation des liaisons est assez simple
  - Pour obtenir un résultat concluant, il suffit
    - d'importer la classe nécessaire
    - et de créer une procédure utilisant la méthode bindProperty, dont les paramètres correspondent aux identifiants et propriétés de dess objets
  - A ne pas oublier
    - d'exécuter cette fonction à la fin de l'initialisation ou de la création de l'application (creationComplete)
    - sans quoi l'application ne prendra pas la liaison en compte

Lier des données

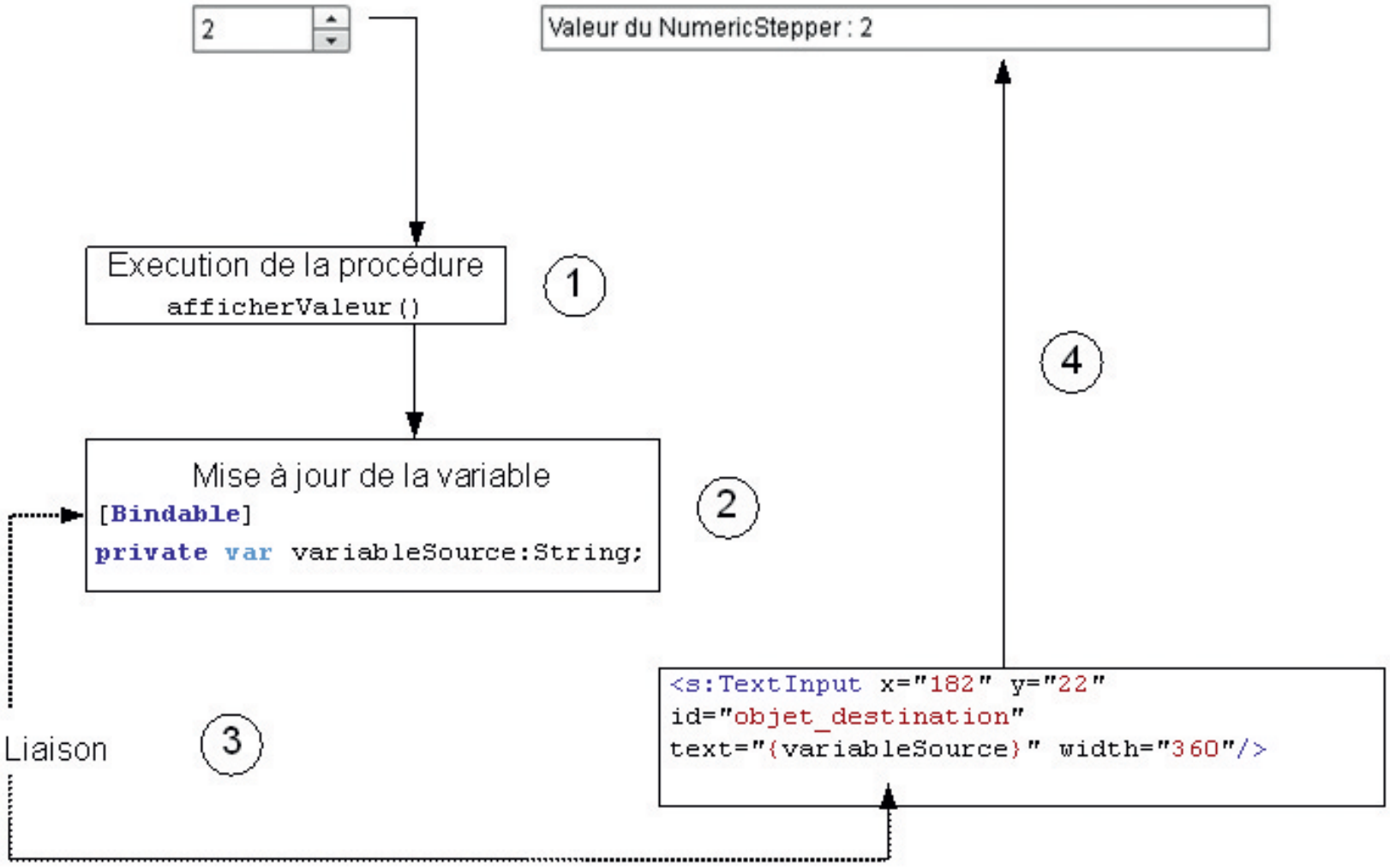
- Utilisation de l'étiquette [Bindable]
  - Jusqu'à présent, la liaison de données a été établie entre deux objets MXML
  - [Bindable] permet de déclarer des variables ActionScript en tant que sources de données pour un objet MXML
    - Ce qui est intéressant quand on combine les deux langages dans la même application
  - Donc,
    - Pour permettre à une variable d'être source de données dans une liaison, sa déclaration doit être précédée de l'étiquette [Bindable]

- Exemple : BindableAs.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application …>
<fx:Script>
<![CDATA[
[Bindable]
private var variableSource:String;
public function afficherValeur():void{
    variableSource = "Valeur du NumericStepper : "+
    String(numericStepper.value);
}
]]>
</fx:Script>
<s:NumericStepper x="10" y="22" width="78" id="numericStepper"
    change="afficherValeur()"/>
<s:TextInput x="182" y="22" id="objet_destination" text="{variableSource}"
  width="360"/>
</s:Application>
```

- Remarques
  - variableSource
    - est considérée comme source de données du fait que sa déclaration est précédée de l'étiquette [Bindable]
  - afficherValeur()
    - est appelée à chaque modification de la valeur du NumericStepper et a pour fonction de modifier la variable source
  - La liaison de données
    - est effectuée sur la propriété text du composant TextInput, ce qui signifie qu'à chaque modification de la variable source, la valeur de la propriété text sera modifiée en conséquence

Schéma d'explication



Lier des données

- **Spécification d'un événement**
  - Par défaut, la synchronisation s'effectue sur la modification de la valeur source
  - Mais il est possible de modifier ce comportement en permettant la synchronisation sur le déclenchement d'un événement précis.
  - Pour ce faire
    - nous devons définir un événement de synchronisation et l'affecter à l'étiquette [Bindable] précédant la variable source
    - Lorsque le dispatch de l'événement sera exécuté, l'étiquette [Bindable] concernée en sera avertie et procédera à la synchronisation des données

- Exemple : BindableAS2.mxml
  - Ici la synchronisation des données ne s'effectue que si la valeur du composant NumericStepper est supérieure ou égale à 5

```
<fx:Script> <![CDATA[
[Bindable(event="siSupOuEgalA5")]
private var variableSource:String;
public function afficherValeur():void{
    if (numericStepper.value >= 5){
      variableSource = "Valeur du NumericStepper :"+
      String(numericStepper.value);
      // Déclaration de l'événement
      var evenement:Event = new Event("siSupOuEgalA5");
       dispatchEvent(evenement);} // Action de dispatch
}
]]>
</fx:Script>
<s:NumericStepper x="10" y="22" width="78" id="numericStepper"
   change="afficherValeur()"/>
<s:TextInput x="182" y="22" id="objet_destination" text="{variableSource}"
  width="360"/>
</s:Application>
```

Remarques

- [Bindable(event=”siSupOuEgalA5”)]
  - précise que l’action de synchronisation ne peut se produire que lorsque l’événement siSupOuEgalA5 est déclenché.
- Par ailleurs,
  - cet événement n’est déclaré dans la procédure afficherValeur() que lorsque la valeur du NumericStepper répond aux conditions fixées
- Vient ensuite
  - l’envoi de l’ordre de mise à jour de la variable à l’étiquette [Bindable], par l’exécution du dispatch de l’événement
  - La valeur de variable est alors immédiatement mise à jour et modifiée.

Gérer des données

- Stocker des données grâce au modèle
  - On vient de voir comment
    - <fx:Binding> peut jouer le rôle du contrôleur
  - Nous allons voir comment le modèle peut être implémenté en combinant MXML et AS

Gérer des données

- <fx:Model>
  - Imaginons une petite application dont le but est d'enregistrer les nouvelles admissions dans un centre hospitalier
  - Une admission consiste à enregistrer les données caractéristiques d'un patient, à savoir son identifiant, son prénom et son nom
  - La première étape va donc consister à créer un modèle capable de stocker ces informations

Gérer des données

- Exemple : Model1.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application …>
<fx:Declarations>
<!-- LE MODÈLE -->
<fx:Model id="ModelePatient">
    <patient>
      <id/>
      <prenom/>
      <nom/>
    </patient>
</fx:Model>
</fx:Declarations>
</s:Application>
```

Gérer des données

- Remarques
  - Un modèle se caractérise par un identifiant
  - À l'intérieur de ce modèle, nous spécifions ce qui peut être considéré comme la classe patient et ses trois attributs :
    - id, prenom et nom
  - Il s'agit d'un élément non visuel
    - Il doit donc impérativement être déclaré à l'intérieur des balises <fx :Declaration/>

Remarques (suite)

- Afin d'étudier l'utilisation du modèle, nous devons ajouter un formulaire de saisie, qui nous permettra d'alimenter le modèle en données :

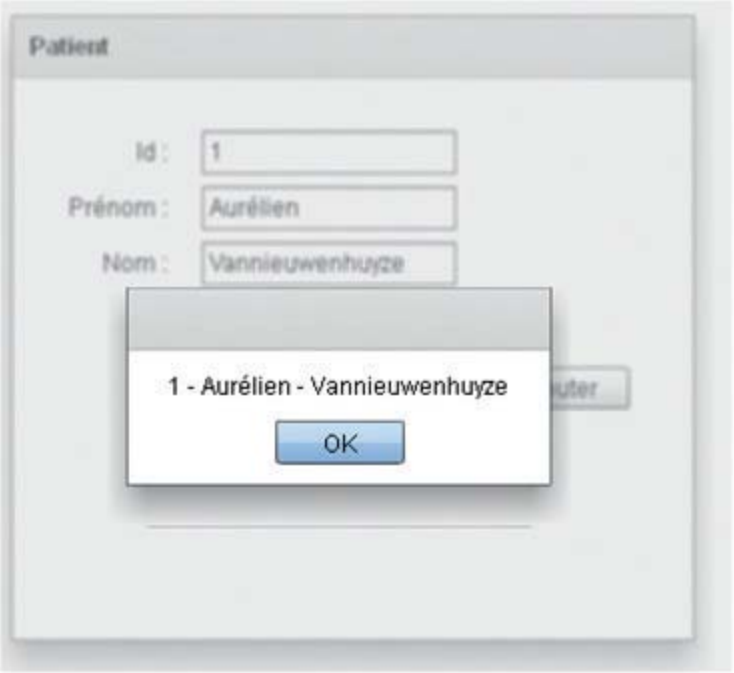
```
<s:Panel x="10" y="10" width="344" height="315" title="Patient">
<mx:Form x="10" y="10" height="115" width="295">
<mx:FormItem label="Id :"><s:TextInput
  id="id_txt"/></mx:FormItem>
<mx:FormItem label="Prénom :"><s:TextInput id="prenom_txt"/>
</mx:FormItem>
<mx:FormItem label="Nom :"><s:TextInput id="nom_txt"/>
</mx:FormItem>
</mx:Form>
<s:Button x="240" y="145" label="Ajouter"
  click="ajouterPatient()"/>
</s:Panel>
```

- Remarques (suite 2)
  - Il convient ensuite d'implémenter une procédure qui va utiliser notre modèle pour stocker les informations
  - Cette procédure sera appelée lorsque l'utilisateur cliquera sur le bouton Ajouter

```
<fx:Script>
<![CDATA[
import mx.controls.Alert;
public function ajouterPatient():void {
   ModelePatient.id = id_txt.text;
   ModelePatient.prenom = prenom_txt.text;
   ModelePatient.nom = nom_txt.text;
   Alert.show (ModelePatient.id +" - "+ModelePatient.prenom+"
   - "+ModelePatient.nom);
}
]]>
</fx:Script>
```

Remarques (suite 3)

- La procédure affecte les attributs du modèle à partir des champs du formulaire pour ensuite afficher la valeur de ces attributs dans une fenêtre d'alerte



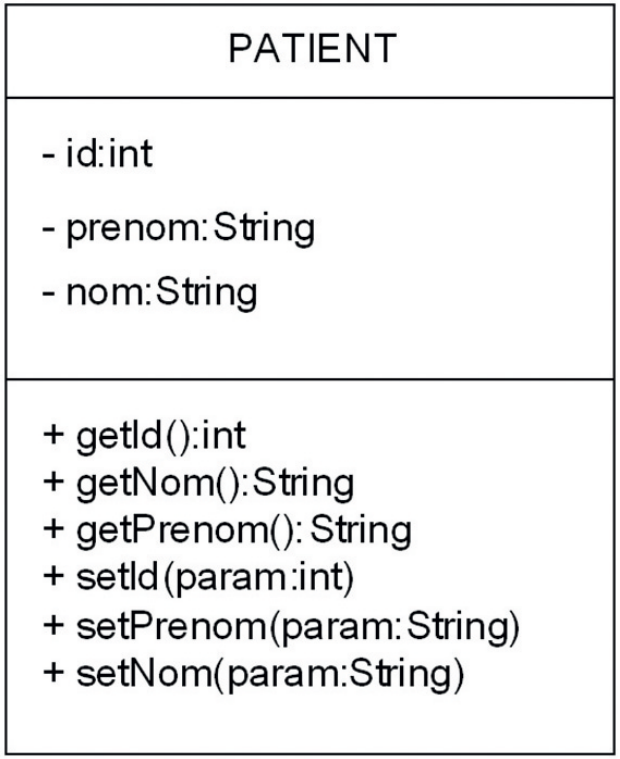
Remarques (suite 4)

- Il aurait également été possible d'utiliser le DataBinding
- L'objet source de la liaison aurait alors été la zone de saisie et l'objet de destination, l'attribut du modèle correspondant à cette zone de saisie

```
<fx:Declarations>
<!-- LE MODÈLE -->
<fx:Model id="ModelePatient">
   <patient>
   <id>{id_txt.text} </id>
   <prenom>{prenom_txt.text} </prenom>
   <nom>{nom_txt.text} </nom>
   </patient>
</fx:Model>
</fx:Declarations>
```

Liaison de données

- Les modèles et ActionScript
  - Déclaration du modèle en AS : alternative de <fx:Model>
  - On crée une classe Patient :



```
package pkg{
public class Patient{ //Déclaration
    des attributs
    private var _id:int;
    private var _nom:String;
    private var _prenom:String;
    //Constructeur
    public function Patient(){}
    //Getter & Setter
    public function get
    prenom():String{
        return _prenom;
    }
    public function set
    prenom(value:String):void{
        prenom = value;
    }
public function get nom():String{
        return _nom;
}
public function set
nom(value:String):void{
        _nom = value;
}
public function get id():int{
        return _id;
}
public function set id(value:int):void{
        _id = value;
}
}
}
```

L’application

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application ...xmlns:Patient="pkg.*">
<fx:Declarations>
<!-- LE MODÈLE -->
<Patient:Patient id="ModelePatient">
</Patient:Patient>
</fx:Declarations>
<fx:Script>
<![CDATA[
import mx.controls.Alert;
public function ajouterPatient():void{
    ModelePatient.id = int(id_txt.text);
    ModelePatient.prenom = prenom_txt.text;
    ModelePatient.nom = nom_txt.text;
    Alert.show (ModelePatient.id +" - "+ModelePatient.prenom+" -
    "+ModelePatient.nom);
}
]]>
</fx:Script>
```

L’application (suite)

```
<!-- LE FORMULAIRE -->
<s:Panel x="10" y="10" width="344" height="315" title="Patient">
<mx:Form x="10" y="10" height="115" width="295">
<mx:FormItem label="Id :">
<s:TextInput id="id_txt"/>
</mx:FormItem>
<mx:FormItem label="Prénom :">
<s:TextInput id="prenom_txt"/>
</mx:FormItem>
<mx:FormItem label="Nom :">
<s:TextInput id="nom_txt"/>
</mx:FormItem>
</mx:Form>
<s:Button x="240" y="145" label="Ajouter" click="ajouterPatient()"/>
</s:Panel>

</s:Application>
```

Remarques

- Grâce aux classes, nous pouvons implémenter la logique métier en y développant des méthodes spécifiques, ce qui était jusque-là impossible à l'aide des méthodes traditionnelles d'implémentation de modèle
- Dans notre exemple, nous aurions pu développer une méthode calculant le nombre de lettres contenues dans le nom de famille saisi, lequel aurait alors été concaténé à ce même nom

Gérer des données

- Valider des données
  - Il s'agit d'en vérifier le format
  - Flex permet de vérifier le format des éléments suivants :
    - carte de crédit (en fonction du type : Visa, MasterCard…) ;
    - valeur monétaire ;
    - date ;
    - adresse e-mail ;
    - nombre ;
    - numéro de téléphone ;
    - numéro de Sécurité sociale ;
    - code postal ;
    - chaîne de caractères ;
    - expression régulière.

Gérer des données

- Exemple : valider du format de l'adresse email
  - Pour commencer, créons un formulaire contenant une zone de texte dédiée à la saisie de l'adresse e-mail :

```
<s:Panel x="10" y="10" width="344" height="159"
  title="E-mail">
  <mx:Form x="10" y="10" height="62" width="295">
      <mx:FormItem label="Votre e-mail :">
          <s:TextInput id="mail_txt"/>
      </mx:FormItem>
  </mx:Form>
  <s:Button x="235" y="80" label="Valider"
  id="btn_valider" />
</s:Panel>
```

Gérer des données

Valider du format de l'adresse email

- Ensuite, rendons la saisie obligatoire
  - Pour cet exemple, le champ mail_txt doit obligatoirement être renseigné
  - Il suffit de lui préciser que ce champ est obligatoire grâce à la propriété required du composant FormItem
    - <mx:FormItem label="Votre e-mail :" required="true">

➔ Un astérisque apparaît alors après le libellé du champ

- Il ne nous reste plus qu'à mettre en place le validateur comme suit : validateur1.mxml

```
<fx:Declarations>
<!-- VALIDATION -->
<mx:Validator id="validation_saisie"
   source="{mail_txt}" property="text">
</mx:Validator>
</fx:Declarations>
<s:Panel x="10" y="10" width="344" height="159" title="E-
   mail">
<mx:Form x="10" y="10" height="62" width="295">
<mx:FormItem label="Votre e-mail :" required="true">
<s:TextInput id="mail_txt"/>
</mx:FormItem>
</mx:Form>
<s:Button x="235" y="80" label="Valider" id="btn_valider" />
</s:Panel>
```

- Remarque
  - Le fonctionnement de la validation est basé sur la notion de DataBinding
  - La source du Validator est le champ mail_txt
  - Lorsque sa propriété text est modifiée, l'action de validation est alors déclenchée

Gérer des données

Valider du format de l'adresse email

- Vérifier le format
  - Utiliser EmailValidator qui vérifie la présence des 3 éléments de l'adresse : nom, arobase et domaine
  - Remplacer le composant de validation par celui-ci

```
<mx:EmailValidator id="validation_email"
   source="{mail_txt}" property="text">
</mx:EmailValidator>
```

  - Ce composant vérifie à la présence oblgatoire du champ et son format

Gérer des données
Valider du format de l'adresse email

- Personnaliser les messages d'erreur
  - Chaque composant de validation possède sa propre bibliothèque de textes à afficher en fonction du type d'erreur rencontré
  - Il est possible de personnaliser ces messages
  - Voyons comment procéder pour notre composant

```
<mx:EmailValidator id="validation_email" source="{mail_txt}"
    property="text" requiredFieldError="Ce champ est
    obligatoire" missingAtSignError="Votre adresse e-mail ne
    comporte pas d'arobase" missingUsernameError="Le nom
    d'utilisateur n'a pas été spécifié"
    missingPeriodInDomainError="Le nom de domaine n'a pas
    été spécifié">
</mx:EmailValidator>
```

Gérer des données

Valider du format de l'adresse email

- Les événements de validation
  - Par défaut, l'événement de validation de format correspond à la notification de changement de la valeur source du composant de validation
  - On peut modifier ce comportement à l'aide des propriétés trigger et triggerEvent du composant de validation
  - La propriété trigger
    - sert à spécifier qu'un composant de l'interface déclenchera l'action de validation
  - La propriété triggerEvent
    - précise l'action qui devra être réalisée sur le composant.

Gérer des données

Valider du format de l'adresse email

- Les événements de validation (suite)
  - Ainsi, si nous souhaitons déclencher la validation d'un champ lorsque l'utilisateur clique sur le bouton btn_verifier de l'interface
    - la propriété trigger de l'élément de validation devra porter la valeur btn_verifier et il faudra affecter la propriété triggerEvent de la valeur click

Gérer des données

Valider du format de l'adresse email

- Les événements de validation (suite)
  - Appliquons ceci à notre exemple en définissant la règle suivante : « L'action de validation doit être effectuée à chaque fois que l'utilisateur clique sur le bouton Valider »

```
<mx:EmailValidator id="validation_email"
   source="{mail_txt}" property="text"
   trigger="{btn_valider}" triggerEvent="click">
</mx:EmailValidator>
```

Gérer des données

Valider du format de l'adresse email

- Gérer la validation en ActionScript
  - Ceci sera important pour l'interaction entre la validation (MXML) et les traitements ActionScript
  - En effet, il serait bien regrettable que le traitement d'enregistrement ait lieu malgré l'erreur de format détectée par le composant de validation
  - Pour cela, nous avons recours à la méthode validate() du composant de validation

Gérer des données

Valider du format de l'adresse email

- Gérer la validation en ActionScript

```
<fx:Script>
<![CDATA[
import mx.controls.Alert;
import mx.events.ValidationResultEvent;
public function ajouterAdresseEmail():void{
    // Résultat de la validation
    var resultatValidation:ValidationResultEvent =
    validation_email.validate()
    if (resultatValidation.type ==
    ValidationResultEvent.VALID){
     Alert.show ("Enregistrement effectué")}
}
]]>
</fx:Script>
```

Gérer des données

Valider du format de l'adresse email

- Remarque 1
  - Le type de résultat retourné par la méthode validate() est VALID si la validation s'est correctement déroulée, et INVALID si la validation a échoué
  - Un simple test sur la valeur de la variable resultatValidation permet d'exécuter ou non le traitement d'ajout

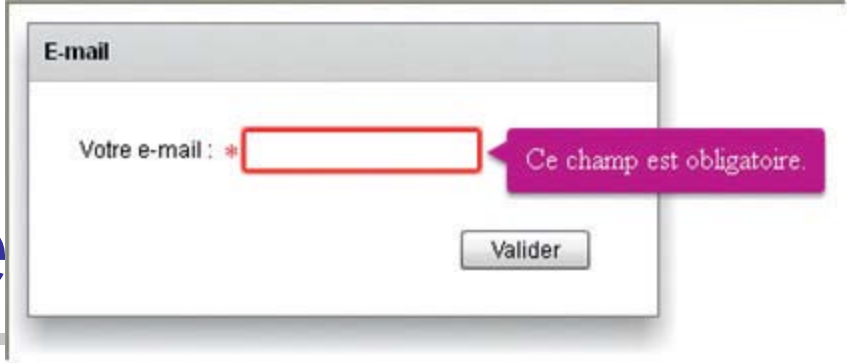
Gérer des données

Valider du format de l'adresse email

- Remarque 2 : Validateur2.mxml
  - **Validation complète d'un formulaire**
  - Si un formulaire comporte de nombreux champs, la méthode que nous venons de voir deviendra rapidement fastidieuse à implémenter
  - La classe Validator pallie ce problème grâce à sa méthode validateAll([*composants de validation séparés par une virgule]*)

```
<fx:Script>
<![CDATA[
import mx.validators.Validator;
import mx.controls.Alert
public function ajouterAdresseEmail():void{
    // Stockage des erreurs de validation dans un tableau
    var tableau_validation:Array =
    Validator.validateAll([validation_email]);
    // Si le tableau ne contient aucun élément, le
    traitement peut être effectué
    if(tableau_validation.length == 0) {
            Alert.show (“Enregistrement effectué”)
    }
}
]]>
</fx:Script>
```

Gérer les donnée

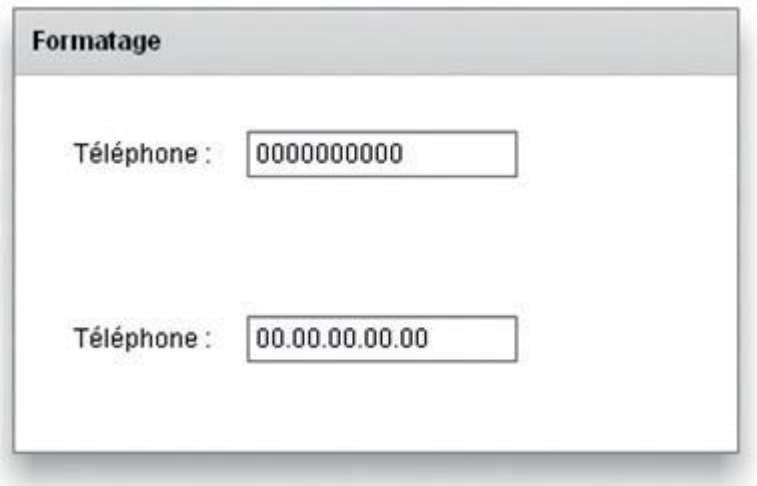


- Mettre la validation en forme
  - Grâce à la balise <fx:Style>, nous pouvons modifier le style CSS de l'application afin de faire correspondre l'affichage des messages d'erreur à la charte graphique de l'application
  - L'exemple ci-dessous permet de modifier la couleur du message d'erreur affiché

```
<fx:Style>
.errorTip {borderColor: #BD007B;
    color: #FFFFFF;
    fontFamily: Base02Embedded;
    fontSize: 16;
    fontWeight: normal;}
</fx:Style>
```

Formater des données

- Objectif
  - Cette opération consiste à transformer une donnée en une chaîne dont le format aura préalablement été précisé
  - L'application la plus commune est le formatage du numéro de téléphone, ce que nous allons voir à travers l'exemple suivant



Exemple : Formatage.mxml

```
<fx:Declarations>
<!-- SPÉCIFICATION DU FORMAT -->
<mx:PhoneFormatter
    id="formatTelephone"
    formatString="##.##.##.##.##">
</mx:PhoneFormatter>
</fx:Declarations>
<!-- LE FORMULAIRE -->
<s:Panel x="10" y="10" width="344" height="210" title="Formatage">
    <mx:Form x="10" y="10" height="62" width="295">
      <mx:FormItem label="Téléphone :" >
                <s:TextInput id="tel_txt"/>
      </mx:FormItem>
    </mx:Form>
    <mx:Form x="10" y="97" width="295" height="63">
      <mx:FormItem label="Téléphone : ">
                <s:TextInput id="telFormate_txt"
                text="{formatTelephone.format (tel_txt.text)}"/>
      </mx:FormItem>
    </mx:Form>
</s:Panel>
```

Exemple : commentaires

- Nous commençons par la spécification d'un composant de formatage propre aux données téléphoniques (à nous)
  - Nous affectons ainsi la propriété formatString de la valeur ##.##.##.##.##
- la propriété text de la seconde zone de texte :
  - text="{formatTelephone.format(tel_txt.text)}"
- Ceci signifieque le formatage est exécuté sur la propriété text de la zone tel_txt lorsque celle-ci est mise à jour (nous retrouvons ici la notion de DataBinding)

Formater lesdonnées

- Les composants de formatage proposés par Flex sont les suivants :
  - Composant de formatage monétaire
    <mx:CurrencyFormatter>
  - Composant de formatage de date
    <mx:DateFormatter>
  - Composant de formatage de numéro de téléphone
    <mx:PhoneFormatter>
  - Composant de formatage de nombre
    <mx:NumberFormatter>
  - Composant de formatage de code postal
    <mx:ZipCodeFormatter>

Formater lesdonnées

- Formatage des différents éléments d'une date

Lettre	Description
Y	Formate l'année. Par exemple : YY = 10 YYYY = 2010
M	Formate le mois. Par exemple : M = 10 MM = 010 MMM = oct MMMM =october
D	Formate le jour. Par exemple : D = 16 DD = 016
E	Formate le jour de la semaine. Par exemple : E = 7 EE = 07 EEE = sat EEEE = saturday
A	Ajoute l'indicateur Matin/Après midi.

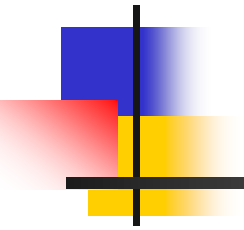
Lettre	Description
J	Formate l'heure de 0 à 23.
H	Formate l'heure de 1 à 24.
K	Formate l'heure au format AM/PM (0-1).
L	Formate l'heure au format AM/PM (1-12).
N	Formate les minutes.
S	Formate les secondes.

Exemples de formatage

Formatage	Résultat obtenu
YYYY.MM.DD	2008.05.15
H:NN	9:05 AM
EEE, DD MMMM YYYY	Thu, 15 May 2008

- Exemple : FormatageDate.mxml
  - L'exemple de code suivant montre comment formater la date du jour à l'aide de l'expression EEE, DD MMMM YYYY

```
<fx:Declarations>
<!-- SPÉCIFICATION DU FORMAT -->
<mx:DateFormatter id="formatDate" formatString="EEE, DD
    MMMM YYYY"></mx:DateFormatter>
</fx:Declarations>
<fx:Script>
<![CDATA[
[Bindable]
public var date:Date = new Date(); //new Date() = date du
    jour
]]>
</fx:Script>
<s:TextInput id="date_txt" text="{formatDate.format(date)}"
    x="10" y="50"/>
```



TD

- Énoncé
  - Reprendre l’exercice sur l’annuaire
  - Utiliser un fichier XML pour sauvegarder les contacts
  - Le formulaire d’ajout peut être ajouté à l’interface ou ajouté comme un popup