



P2. INTERACCIÓN CON EL USUARIO

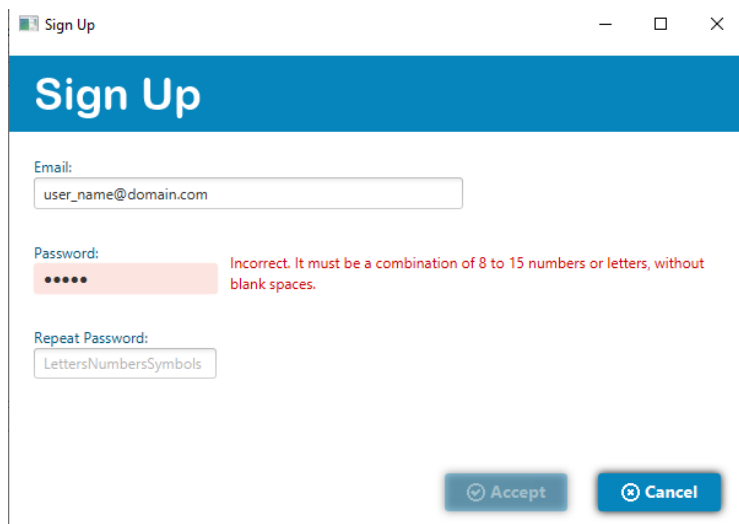
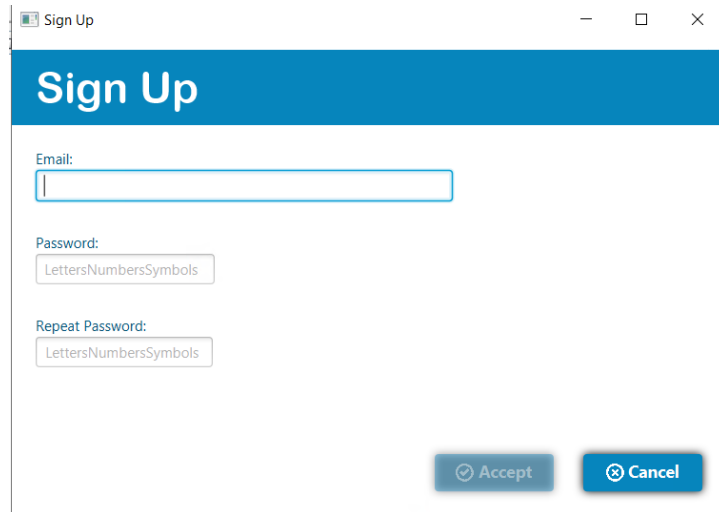
EJERCICIO: [FORMULARIO DE REGISTRO](#)

Interfaces Persona Computador

Depto. Sistemas Informáticos y Computación

UPV

Formulario de Registro



- Recoge la información de registro de un usuario: su email y contraseña.
- **Comprueba** que el **dato es válido** cuando el usuario sale del campo de edición
- Permite aceptar la información solo cuando se han introducido todos datos y son válidos
- El botón **aceptar** borra los campos inicializando los valores
- El botón **cancelar** cierra la ventana

Pasos a seguir:

1. Diseñar la interfaz
2. Implementar la validación de los datos cuando el usuario abandona el campo
3. Implementar el botón aceptar
4. Implementar el botón cancelar

1. Diseñar la interfaz

- Elegir contenedores adecuados
- Añadir los contenedores y controles
- Dar formato a los controles

Sign Up

Email: Incorrect Email

Password: Incorrect. It must be a combination of 8 to 15 numbers or letters, without blank spaces.

Repeat Password: Passwords don't match

1. Diseñar la interfaz: Elegir contenedores

1. BorderPane

The image shows a 'Sign Up' form with a blue header bar containing the text 'Sign Up'. Below the header is a large white area with rounded corners, outlined in yellow, containing three input fields: 'Email:' with the value 'yourEmail@domain.com' and a red error message 'Incorrect Email'; 'Password:' with the value 'LettersNumbersSymbols' and a red error message 'Incorrect. It must be a combination of 8 to 15 numbers or letters, without blank spaces.'; and 'Repeat Password:' with the value 'LettersNumbersSymbols' and a red error message 'Passwords don't match'. At the bottom of the form is a white footer bar, outlined in blue, containing two buttons: 'Accept' and 'Cancel'. To the right of the form, three labels with arrows point to these regions: '1. 1 Top' points to the header, '1. 2 Center' points to the main content area, and '1. 3 Bottom' points to the footer bar.

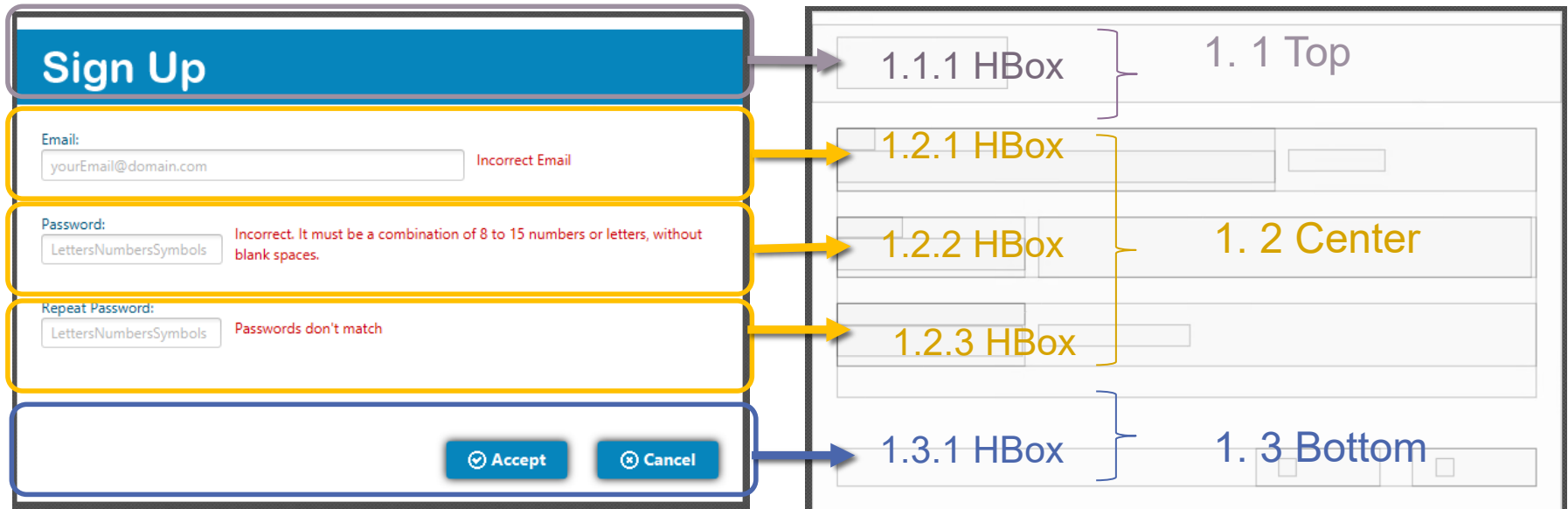
1. 1 Top

1. 2 Center

1. 3 Bottom

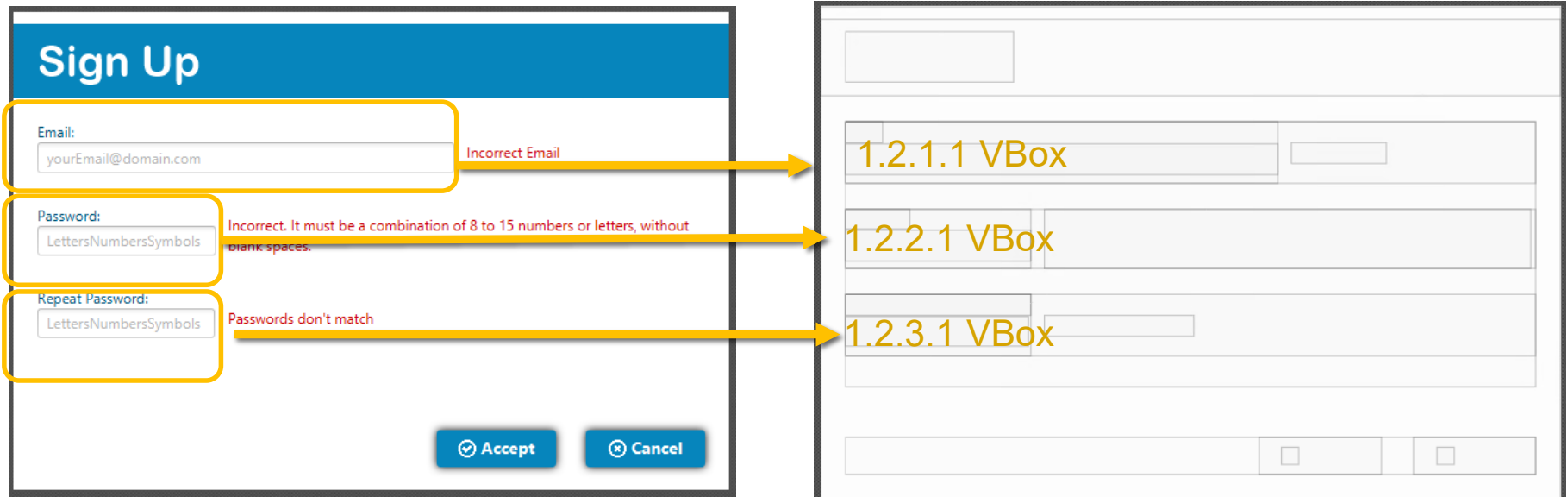
1. Diseñar la interfaz: Elegir contenedores

1. BorderPane

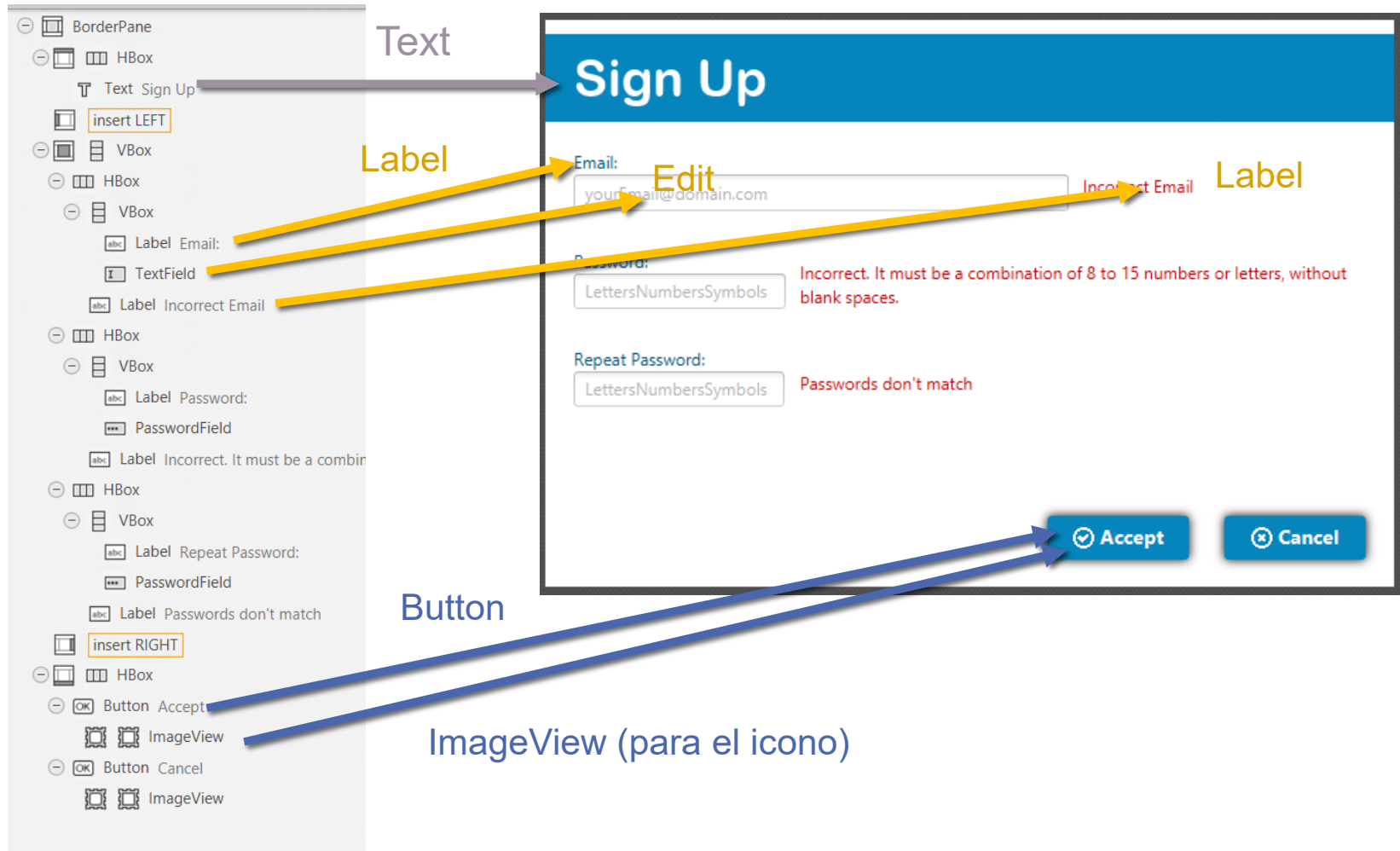


1. Diseñar la interfaz: Elegir contenedores

1. BorderPane



1. Diseñar la interfaz: Añadir contenedores y controles



1. Diseñar la interfaz: Dar formato a los controles

BorderPane

Propiedades

Color de fondo

The image shows the JavaFX IDE interface for designing a 'Sign Up' form. On the left, the 'BorderPane' container is selected in the 'Properties' window. The 'Style' property is set to '-fx-background-color: white'. The form itself has a blue header, white body, and blue buttons. Red error messages are visible next to the input fields.

The 'Sign Up' form layout includes:

- Header:** A blue bar with the text 'Sign Up'.
- Email:** A text field containing 'yourEmail@domain.com' with a red error message 'Incorrect Email'.
- Password:** A password field with a red error message 'Incorrect. It must be a combination of 8 to 15 numbers or letters, without blank spaces.'
- Repeat Password:** A password field with a red error message 'Passwords don't match'.
- Buttons:** 'Accept' and 'Cancel' buttons at the bottom.

The 'Properties' window on the right shows the 'Style' property set to '-fx-background-color: white'.

1. Diseñar la interfaz: Dar formato a los controles

HBox

Propiedades

Layout

Color de fondo

Coloca la cabecera separada del margen de la ventana

Coloca el título separado de los bordes del HBox

1. Diseñar la interfaz: Dar formato a los controles

Text

Texto y tipo de letra

Color del texto

Properties : Text

Text: Sign Up

Font: Arial Rounded MT Bold ...

Font Smoothing ...: GRAY

Text Alignment: [Left] [Center] [Right] [Justify]

Wrapping Width: 0

Strikethrough: []

Underline: []

Line Spacing: 0

Specific

Fill: WHITE

Smooth: [x]

1. Diseñar la interfaz: Dar formato a los controles

The image shows a Java Swing IDE interface with a 'Sign Up' form design. The form includes a title bar, an email input field with an 'Incorrect Email' error message, a password input field with an 'Incorrect. It must be a combination of 8 to 15 numbers or letters, without blank spaces.' error message, a repeat password input field with a 'Passwords don't match' error message, and 'Accept' and 'Cancel' buttons.

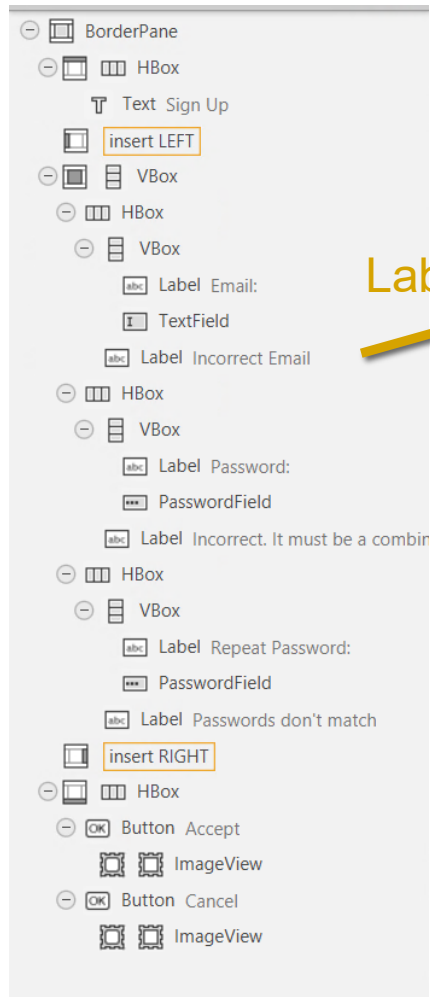
Annotations and property windows are used to configure the controls:

- Label:** A yellow arrow points to the 'Email:' label in the form.
- Properties : Label:** A yellow box highlights the 'Text' (Email:), 'Font' (System 12px), and 'Text Fill' (#025477) properties.
- Node:** A yellow box highlights the 'Alignment' (CENTER_LEFT) property.

The IDE's component palette on the left shows the following structure:

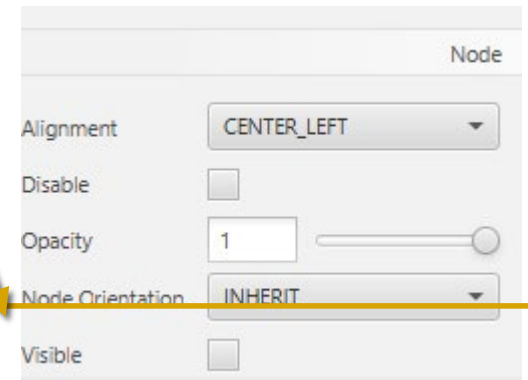
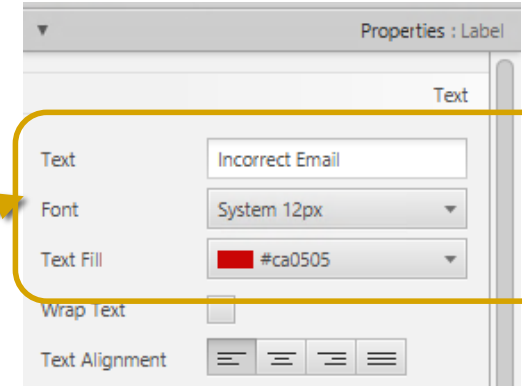
- BorderPane
 - HBox
 - Text Sign Up
 - insert LEFT
 - VBox
 - HBox
 - VBox
 - Label Email:
 - TextField
 - Label Incorrect Email
 - HBox
 - VBox
 - Label Password:
 - PasswordField
 - Label Incorrect. It must be a combin
 - HBox
 - VBox
 - Label Repeat Password:
 - PasswordField
 - Label Passwords don't match
 - insert RIGHT
 - HBox
 - Button Accept
 - ImageView
 - Button Cancel
 - ImageView

1. Diseñar la interfaz: Dar formato a los controles



Texto de la etiqueta, tipo de letra y color de la letra

Label



Ocultamos el mensaje de error

1. Diseñar la interfaz: Dar formato a los controles

The image shows a Java Swing IDE with a 'Sign Up' window design. The window has a blue header with the text 'Sign Up'. Below the header, there are three text input fields: 'Email:', 'Password:', and 'Repeat Password:'. The 'Email:' field contains 'yourEmail@domain.com' and has a red error message 'Incorrect Email' to its right. The 'Password:' field contains 'LettersNumbersSymbols' and has a red error message 'Incorrect. It must be a combination of 8 to 15 numbers or letters, without blank spaces.' to its right. The 'Repeat Password:' field contains 'LettersNumbersSymbols' and has a red error message 'Passwords don't match' to its right. At the bottom of the window, there are two buttons: 'Accept' and 'Cancel'.

Annotations and settings shown:

- TextField**: A yellow arrow points from the 'Email:' text field to the 'insert LEFT' button in the 'Properties : TextField' panel.
- Texto de ayuda al usuario y tamaño de la letra**: A yellow arrow points from the text 'Incorrect Email' to the 'Prompt Text' and 'Font' settings in the 'Properties : TextField' panel.
- Alignment**: A yellow arrow points from the 'Incorrect Email' text to the 'Alignment' dropdown menu in the 'Properties : TextField' panel, which is set to 'CENTER_LEFT'.
- Tamaño mínimo si se hace más pequeña la ventana**: A yellow arrow points from the 'Email:' text field to the 'Min Width' and 'Min Height' settings in the 'Layout : TextField' panel.
- Tamaño al abrir la ventana**: A yellow arrow points from the 'Email:' text field to the 'Pref Width' and 'Pref Height' settings in the 'Layout : TextField' panel.

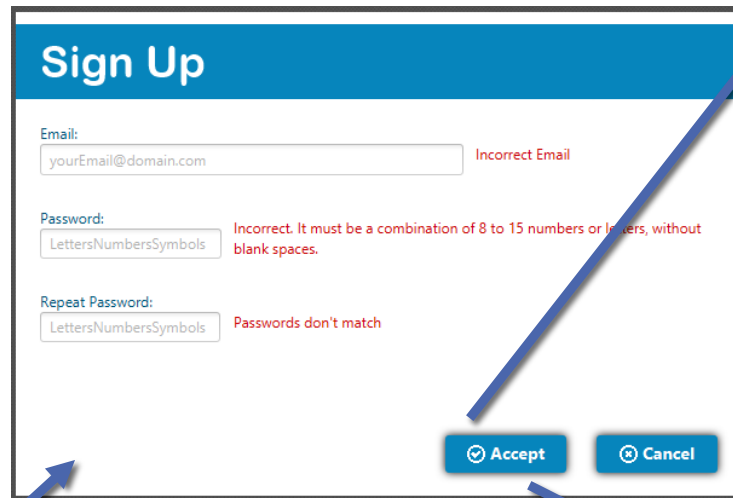
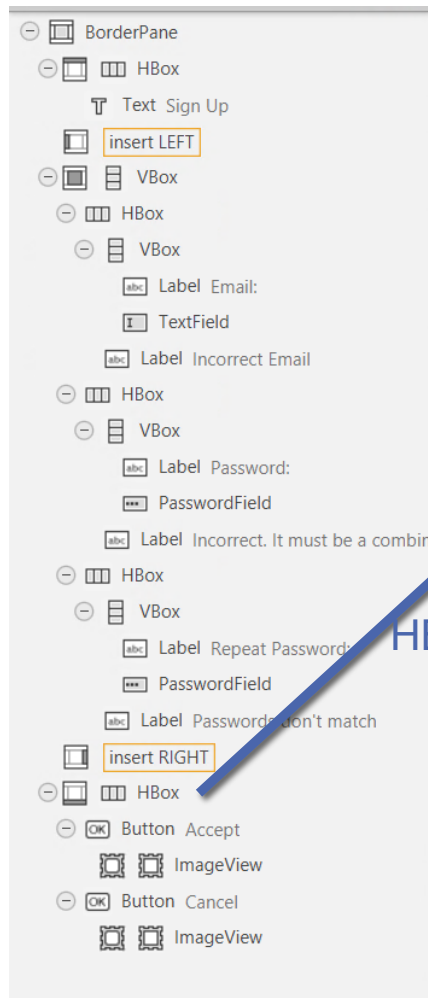
The 'Properties : TextField' panel shows the following settings:

- Prompt Text: yourEmail@domain.com
- Text: (empty)
- Font: System 12px
- Alignment: CENTER_LEFT

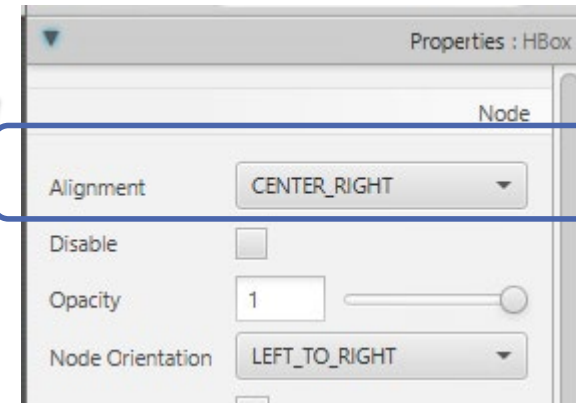
The 'Layout : TextField' panel shows the following settings:

- Min Width: 150
- Min Height: 26
- Pref Width: 350
- Pref Height: 26
- Max Width: USE_COMPUTED_SIZE
- Max Height: USE_COMPUTED_SIZE
- Width: 350
- Height: 26

1. Diseñar la interfaz: Dar formato a los controles

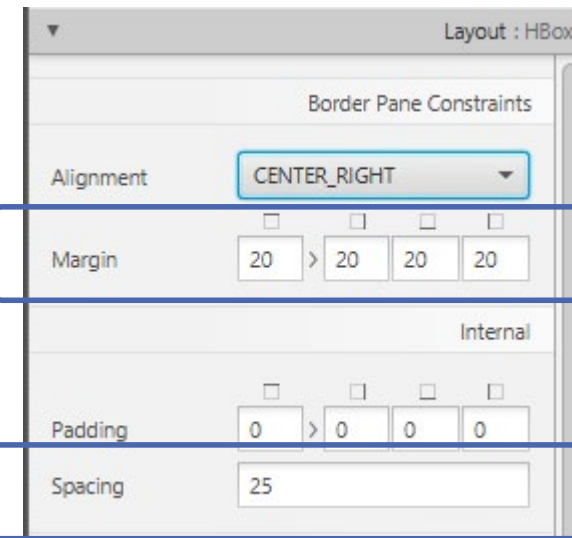


Botones a la derecha



HBox

Margen alrededor de los botones



Separación entre botones

1. Diseñar la interfaz: Dar formato a los controles

Texto, tipo y color de letra

Button

ImageView

No podrá ser más pequeño de 80
Tamaño de 100 al abrir la ventana

Selecciona el icono de la carpeta icons
Tendrás que ajustar la imagen al tamaño del botón

Properties : Button

Text

Accept

Font

System 14px (Bold)

Text Fill

WHITE

Layout : Button

Size

Min Width

80

Min Height

USE_COMPUTED_SIZE

Pref Width

100

Pref Height

USE_COMPUTED_SIZE

Max Width

USE_COMPUTED_SIZE

Max Height

USE_COMPUTED_SIZE

Width

100

Specific

Image

@ ..\icons\accept_white

Preserve Ratio

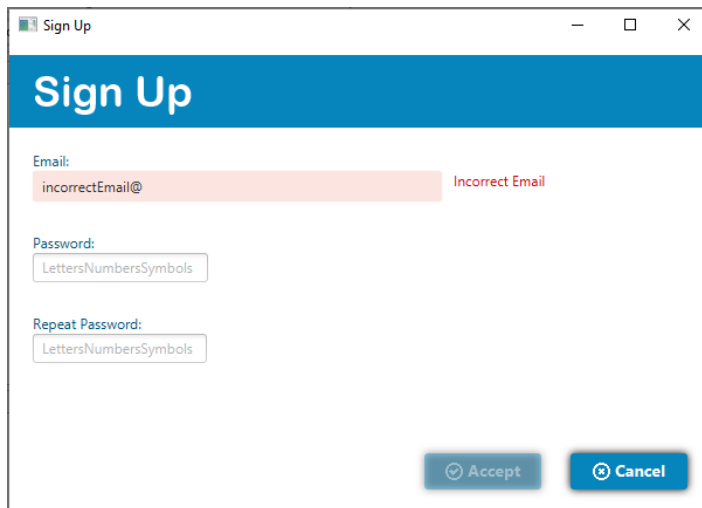
Smooth

2. Implementar la validación

2. Implementar la validación de los datos cuando el usuario abandona el campo:
 - Email: comprueba que el dato corresponde con un email válido. En caso de error, informa al usuario y lo posiciona en el campo para que lo corrija
 - Contraseña: comprueba que es un email válido. En caso de error, informa al usuario y lo posiciona en el campo para que lo corrija
 - Contraseña2: comprueba que es el mismo valor que el introducido previamente. En caso de error, borra el contenido de ambos campos de contraseña y posiciona al usuario el primer campo de contraseña.

2. Implementar la validación: email

- En caso de error, informa al usuario y lo posiciona en el campo para que lo corrija.



The screenshot shows a 'Sign Up' window with a blue header. Below the header, there are three input fields: 'Email:', 'Password:', and 'Repeat Password:'. The 'Email:' field contains the text 'incorrectEmail@' and has a red border with the text 'Incorrect Email' to its right. The 'Password:' and 'Repeat Password:' fields have a light blue border and contain the placeholder text 'LettersNumbersSymbols'. At the bottom right of the form, there are two buttons: 'Accept' and 'Cancel'.

- Añadiremos como atributo una propiedad boolean para controlar si el valor es correcto o no
- En el método Initialize:
 - Inicializaremos a false la propiedad boolean.
 - Añadiremos un listener a la propiedad `focusedProperty()` del campo de edición:
 - Si el usuario está abandonando el campo, el nuevo valor de la propiedad es false.
 - Si el usuario entra en el campo, la propiedad es true

2. Implementar la validación: email

```
public class FXMLSignUpController implements Initializable {
    @FXML
    private TextField eemail;
    ....
    //properties to control valid fields values.
    private BooleanProperty validEmail;
    ....
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        validEmail = new SimpleBooleanProperty();
        validEmail.setValue(Boolean.FALSE);

        //Check values when user leaves edits
        eemail.focusedProperty().addListener((observable, oldValue, newValue)->{
            if(!newValue){ //focus lost.
                checkEditEmail();
            }
        });
    }
    ...
}
```

1. **Añade** la propiedad a la clase controller. Se usará para controlar si valor introducido en el campo email es correcto o no

2. **Crea e Inicializa** la propiedad añadida a FALSE.

3. **Añade** un listener a la propiedad **focusedProperty()** del TextEdit del email

El nuevo valor de la propiedad es falso cuando el usuario abandona el TextEdit.
Añade la llamada a un método que también vas a implementar.

2. Implementar la validación: email

4. Añade este método

```
private void checkEditEmail(){  
    if(!Utils.checkEmail(eemail.textProperty().getValueSafe()))  
        //Incorrect email  
        manageError(lIncorrectEmail, eemail,validEmail );  
    else  
        manageCorrect(lIncorrectEmail, eemail,validEmail );  
}
```

`Utils.checkEmail`: método de la clase `Utils` dada en el proyecto base:

- Devuelve cierto si el valor corresponde con un email.
- Falso en caso contrario

Los métodos `manageError` y `manageCorrect` se han proporcionado en el proyecto base. Reciben como parámetro:

- `Label` añadida para informar al usuario del error
- `TextEdit` añadido para introducir el valor
- `Propiedad` añadida para controlar que el valor introducido es correcto o no

2. Implementar la validación: contraseña

- En caso de error, informa al usuario y lo posiciona en el campo para que lo corrija.
 - Repite el proceso seguido para validar el email:
 1. Define una propiedad boolean `validPassword`
 2. Crea e inicializa la propiedad en el método `Initialize`
 3. Añade un listener a `focusedProperty()` del campo de edición de la contraseña:
 - Si el nuevo valor es falso, llama a un método que compruebe el valor de la propiedad
 4. Añade el método que comprueba si el valor introducido es una contraseña válida:
 - Usa el método `Utils.checkPassword` proporcionado

2. Implementar la validación: contraseña2

- Comprueba que es el mismo valor que el introducido previamente.
- En caso de error, borra el contenido de ambos campos de contraseña y posiciona al usuario el primer campo de contraseña.
- Repite el proceso seguido para validar el email:
 1. Define una propiedad booleana `equalPasswords`
 2. Crea e inicializa la propiedad en el método `Initialize`
 3. Añade un listener a `focusedProperty()` del campo de edición de la contraseña2:
 - Si el nuevo valor es falso, llama a un método que compruebe si el valor es el mismo
 4. Añade el método que comprueba es el mismo valor que el introducido previamente. Si el valor no es el mismo:
 - Informará al usuario
 - borrará el valor de los campos de edición que contienen la contraseña
 - Mandará al usuario al primer campo de la contraseña

2. Implementar la validación: contraseña2

- Comprueba que es el mismo valor que el introducido previamente.
- En caso de error, borra el contenido de ambos campos de contraseña y posiciona al usuario el primer campo de contraseña.

4. Añade este método

```
private void checkEquals(){  
    if(epassword.textProperty().getValueSafe().compareTo(  
        epassword2.textProperty().getValueSafe()) != EQUALS){  
  
        showErrorMessgae(1PassDifferent,epassword2);  
        equalPasswords.setValue(Boolean.FALSE);  
        epassword.textProperty().setValue("");  
        epassword2.textProperty().setValue("");  
        epassword.requestFocus();  
  
    }else  
        manageCorrect(1PassDifferent,epassword2,equalPasswords);  
}
```

- Comprueba que los dos edits tienen el mismo valor introducido
- Se muestra el error
- Pone la propiedad a FALSE
- Limpia los campos de edición
- Envía el foco al primer campo contraseña

5. Añade a la clase la constante EQUALS

```
public class FXMLSignUpController implements Initializable {  
    ...  
    //When to strings are equal, compareTo returns zero  
    private final int EQUALS = 0;  
    ...  
}
```

2. Implementar la validación: Métodos proporcionados

En la clase Controller:

```
private void manageError(Label errorLabel, TextField textField, BooleanProperty boolProp ){
    boolProp.setValue(Boolean.FALSE);
    showErrorMessage(errorLabel, textField);
    textField.requestFocus();
}
```

- Pone a FALSO la propiedad que guarda si el valor es válido
- Muestra el mensaje de error y cambia el color de fondo del edit
- Manda el foco al edit

```
private void manageCorrect(Label errorLabel, TextField textField, BooleanProperty boolProp ){
    boolProp.setValue(Boolean.TRUE);
    hideErrorMessage(errorLabel, textField);
}
```

- Pone a CIERTO la propiedad que guarda si el valor es válido
- Esconde el mensaje de error y cambia el color de fondo del edit

```
private void showErrorMessage(Label errorLabel, TextField textField)
{
    errorLabel.visibleProperty().set(true);
    textField.styleProperty().setValue("-fx-background-color: #FCE5E0");
}
```

- Muestra el mensaje de error
- Cambia el color de fondo del edit a rojo

```
private void hideErrorMessage(Label errorLabel, TextField textField)
{
    errorLabel.visibleProperty().set(false);
    textField.styleProperty().setValue("");
}
```

- Esconde el mensaje de error
- Cambia el color de fondo del edit a su valor por defecto

2. Implementar la validación: Métodos proporcionados

En la clase Utils:

```
public static Boolean checkEmail (String email)
```

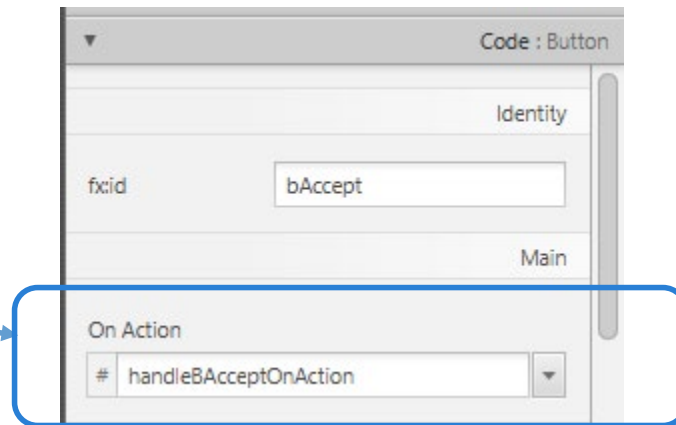
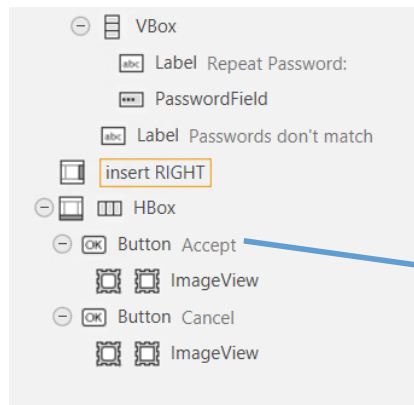
- Comprueba si el string email contiene una dirección de correo electrónico válido.

```
public static Boolean checkPassword (String password)
```

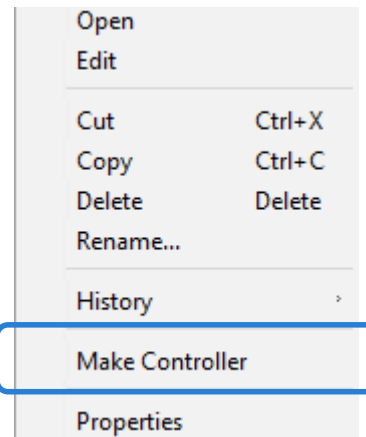
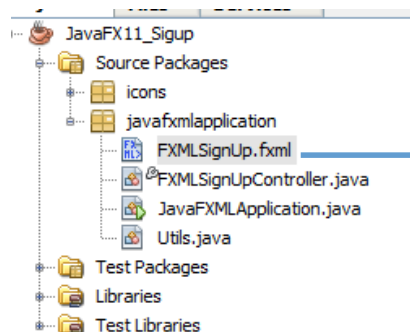
- Comprueba si el string password tiene una longitud entre 8 y 15 caracteres y es una combinación de números y letras.

3. Implementar el botón *Aceptar*

1. Añade el manejador para el evento `OnAction` del botón en scene builder



2. Llama a Make Controller



3. Implementar el botón **Aceptar**

3. Proporciona el código del manejador:

@FXML

```
private void handleBAcceptOnAction(ActionEvent event) {
```

```
    eemail.textProperty().setValue("");  
    epassword.textProperty().setValue("");  
    epassword2.textProperty().setValue("");
```

```
    validEmail.setValue(Boolean.FALSE);  
    validPassword.setValue(Boolean.FALSE);  
    equalPasswords.setValue(Boolean.FALSE);
```

```
}
```

- Limpia el contenido de los text edits del email, contraseña y repetición de contraseña

- Pone a FALSE las propiedades creadas para guardar si el valor introducido es válido

3. Implementar el botón **Aceptar**

4. Añade el siguiente código al método initialize, para que el botón solo se habilite si el usuario introduce todos los valores y son válidos.

```
@Override  
public void initialize(URL url, ResourceBundle rb) {  
    ...
```

```
    BooleanBinding validFields = Bindings.and(validEmail, validPassword)  
        .and(equalPasswords);
```

Usamos la clase `Bindings` para hacer un `and` lógico sobre las tres propiedades que guardan si el valor introducido en los campos es válido y guardamos el resultado en una variable de tipo `BooleanBinding`

```
    bAccept.disableProperty().bind(  
        Bindings.not(validFields)  
    );  
    ...  
}
```

Hacemos que la `disableProperty()` del botón esté conectada al valor de la `BooleanBinding` anterior. Solo se habilitará cuando todas las propiedades de control estén a cierto

3. Implementar el botón Cancelar

- Cierra la ventana. Vamos a implementarlo usando un método de conveniencia y una expresión lambda.
- 1. Proporciona el fx:id bcancel al botón cancelar
- 2. Llama a Make Controller
- 3. Añade el siguiente código al método initialize:

```
@Override
public void initialize(URL url, ResourceBundle rb) {
    ...
    bCancel.setOnAction( (event)->{
        bCancel.getScene().getWindow().hide();
    });

    ...
}
```

- Accede a la escena en la que está el botón
- Desde la escena, accede al stage o ventana.
- Llama al método hide() que la cierra.

Siempre se puede acceder a la escena y ventana en la que está un nodo

Referencias

- Tutorial Oracle: Handling JavaFX Events
<http://docs.oracle.com/javafx/2/events/jfxpub-events.htm>
- API JavaFX 11: <https://openjfx.io/javadoc/17/>
- JavaFX 8 Event Handling Examples:
<http://code.makery.ch/blog/javafx-8-event-handling-examples/>
- Cálculo Lambda en Java: Raoul-Gabriel Urma, Mario Fusco, and Alan Mycroft, *Java 8 in Action*
Lambdas, streams, and functional-style programming