Universidad Nacional de Quilmes

CONSTRUCCIÓN DE INTERFACES DE USUARIO

VENTANAS

MANEJO DE ERRORES

Hasta acá vimos

- Interfaces (UI vs UX)
- Aplicaciones Desktop
- MVC vs MVVM
- Binding
- Arena Framework
  - Labels
  - Buttons
  - Inputs
  - Selectors
  - Panels
  - Tables

Qué vamos a ver hoy

- Distintos tipos de Ventanas en Arena
- Manejo de Errores
  - Validaciones
  - Excepciones
- Con esto prácticamente ya pueden crear una App Desktop completa

Start Application

- Hasta ahora para levantar una aplicación escribimos:

```
class MyWindow(model:*) : MainWindow<*>(model)

fun main () {
  val setUp = listOf("cosa 1", "cosa 2")
  MyWindow(MyModel(setUp)).startApplication()
}
```

- Podemos “hacer cosas” antes que se abra la ventana
- Pero se “sienten” externas a nuestra aplicación

Application

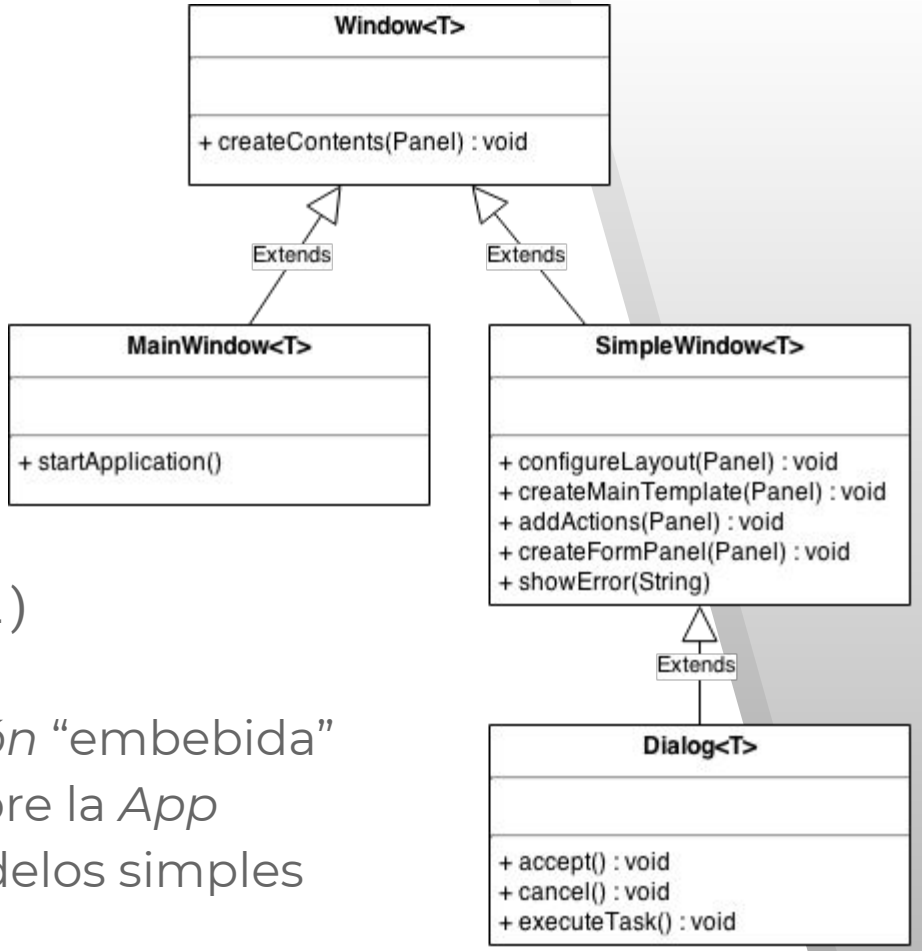
```
fun main() = MyApp().start()
```

```
class MyApp() : Application() {
  override fun createMainWindow(): Window<*> {
    val setUp = listOf("cosa 1", "cosa 2")
    val model = MyModel(setUp)
    return MyWindow(this, model)
  }
}
```

```
class MyWindow : Window<MyModel> {
  constructor(parent: WindowOwner, model: MyModel) : super(parent, model)
  override fun createContents(mainPanel: Panel?) { /* cosas */ }
}

class MyModel(val setUp: Any)
```

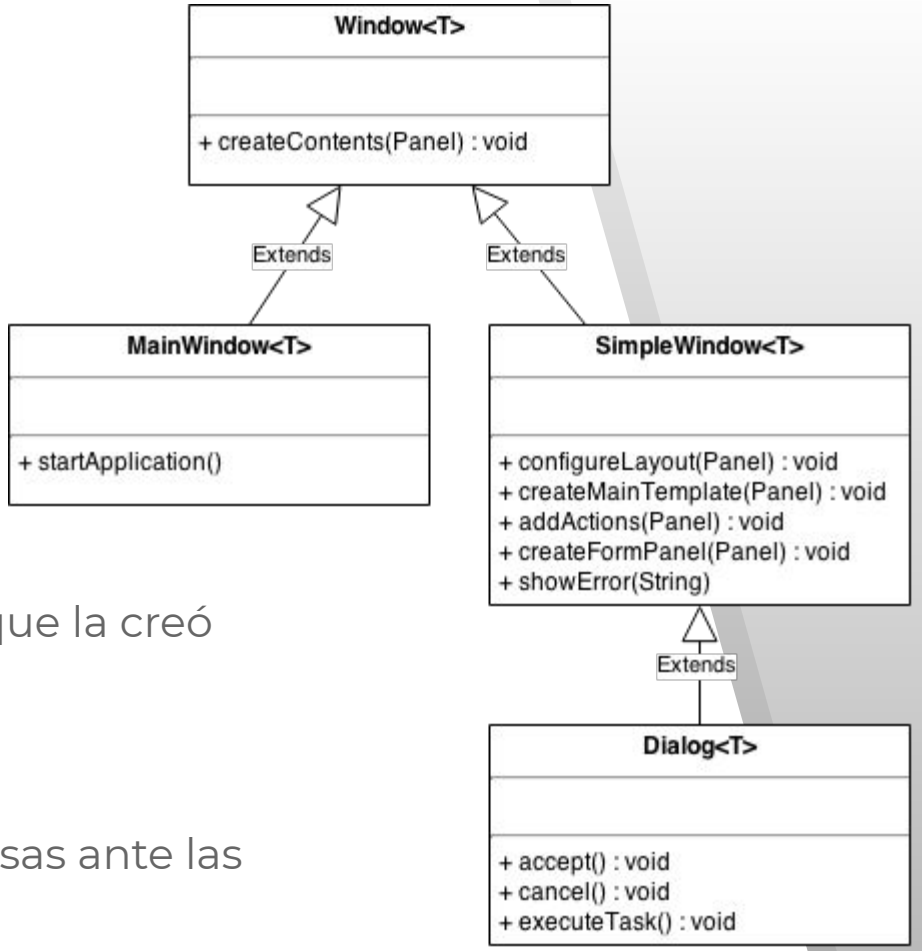
Ventanas

- Arena permite extender de 4 tipos de Ventanas
  - `Window`
    - ➤ Ventana base
    - ➤ Hay que sobreescribir `createContents(Panel)`
  - `MainWindow`
    - ➤ Viene con una *Aplicación* “embebida”
    - ➤ No tenemos control sobre la *App*
    - ➤ Útil para pruebas o modelos simples



Ventanas

- `SimpleWindow`
  - Provee un Template base
    - `ErrorsPanel`
    - `FormPanel`
    - `ActionsPanel`
- `Dialog`
  - Extiende `SimpleWindow`
  - Hay una relación con la ventana que la creó
  - Control sobre las acciones
    - `Accept`
    - `Cancel`
  - La ventana padre puede hacer cosas ante las acciones disparadas

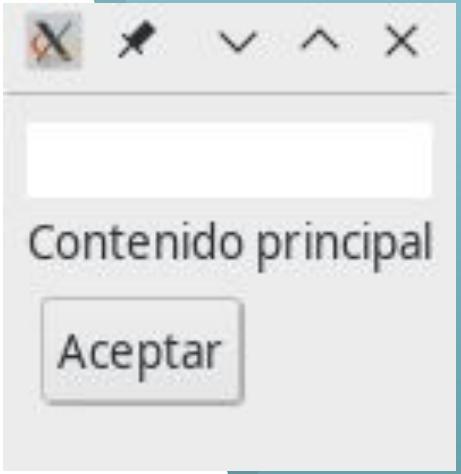


MainWindow

```
fun main() {
  val setUp = listOf("cosa 1", "cosa 2")
  val model = MyModel(setUp)
  MyWindow(model).start()
}

class MyWindow : MainWindow<MyModel> {
  constructor(model: MyModel) : super(model)
  override fun createContents(mainPanel: Panel?) { /* cosas */ }
}

class MyModel(val setUp: Any)
```

SimpleWindow

```
fun main() = MyApp(listOf("cosa 1", "cosa 2")).start()
class MyApp(private val setUp: Any?) : Application() {
  override fun createMainWindow(): Window<*> {
    return MyWindow(this, MyModel(setUp))
  }
}

class MyWindow : SimpleWindow<MyModel> {
  constructor(parent: WindowOwner, model: MyModel) : super(parent, model)
  override fun createFormPanel(mainPanel: Panel?) {
    Label(mainPanel) withText "Contenido principal"
  }
  override fun addActions(actionsPanel: Panel?) {
    Button(actionsPanel) with { caption = "Aceptar" }
  }
}

class MyModel(val setUp: Any)
```



Cómo levanto otra ventana

- Haciendo

  `OtraWindow(owner, viewModel).open()`
- En general a partir de una acción
- Debe conocer la ventana que la invocó

  (por convención *owner* o *parent*)
- También necesita un *modelObject*
- Se provee al instanciarla

Window ⇒ Dialog

```
class MyWindow(model: MyModel) : MainWindow<MyModel>(model) {
  override fun createContents(mainPanel: Panel) {
    Button(mainPanel) with {
      caption = "Abrir Diálogo"
      onClick { MyDialog(thisWindow, modelObject) with {
        onAccept { /* Cosas que quiero hacer si se cliquea en Aceptar  */ }
        onCancel { /* Cosas que quiero hacer si se cliquea en Cancelar  */ }
        open()
      }
    }
}}
class MyDialog(owner:WindowOwner, model:MyModel) : Dialog<MyModel>(owner, model) {
  override fun createFormPanel(mainPanel: Panel) {
    Label(mainPanel) withText "Dialog"
  }
  override fun addActions(actionsPanel: Panel) {
    Button(actionsPanel) with { caption = "Aceptar";  onClick { accept() } }
    Button(actionsPanel) with { caption = "Cancelar"; onClick { cancel() } }
}}
```

Window ⇒ Window

- Los `Dialog` funcionan como “modales”
- Se usan para algo específico y se cierran
- Es posible abrir `Windows` desde `Windows`
- Pero no tienen el control de Acciones de los `Dialogs`
- Es posible seguir abriendo `Ventanas` desde `Ventanas` ya abiertas
- No es recomendable salvo que se cierre la que invocó
  - Porque se van “*stackeando*” en pantalla
  - Y “*bloquea*” a la ventana que la invocó (hasta que se cierre)

Window ⇔ Window

```
class MyWindow1 : Window<MyModel> {
  constructor(owner: WindowOwner, model: MyModel) : super(owner, model)
  override fun createContents(mainPanel: Panel) {
    Label(mainPanel) withText "Soy Window 1"
    Button(mainPanel) with {
      caption = "Abrir Window 2"
      onClick { thisWindow.close(); MyWindow2(thisWindow, modelObject).open() }
  }}
}
class MyWindow2 : Window<MyModel> {
  constructor(owner: WindowOwner, model: MyModel) : super(owner, model)
  override fun createContents(mainPanel: Panel?) {
    Label(mainPanel) withText "Soy Window 2"
    Button(mainPanel) with {
      caption = "Abriw Window 1"
      onClick { thisWindow.close(); MyWindow1(thisWindow, modelObject).open() }
  }}
}
```

Validaciones

- En un mundo ideal el usuario completa los formularios con la información correcta. En el mundo real... bueno, ya saben...
- Es importante comprender
  - El contexto en que el usuario va a interactuar con nuestra app
  - Y lo que pretende conseguir al completar la información

Validaciones

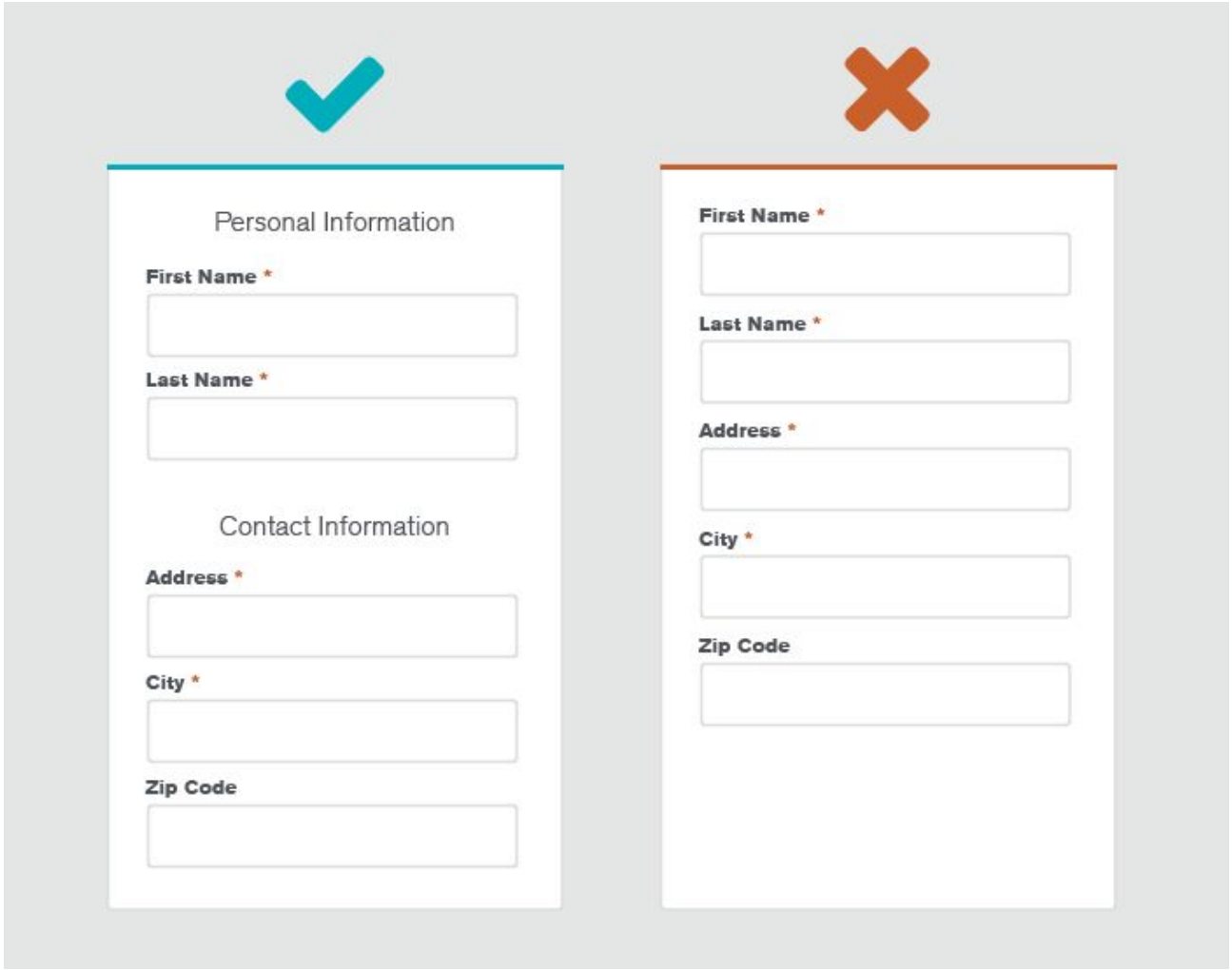
- Deben estar diseñadas para comunicarse con los usuarios y guiarlos en áreas de ambigüedad
- Cuando un usuario encuentra dificultades (por ejemplo al completar un formulario) el problema es más emocional que técnico
- Es importante convertir una interacción ambigua en una clara y ayudar al usuario a comprender en qué se equivocó y cómo debe corregirlo

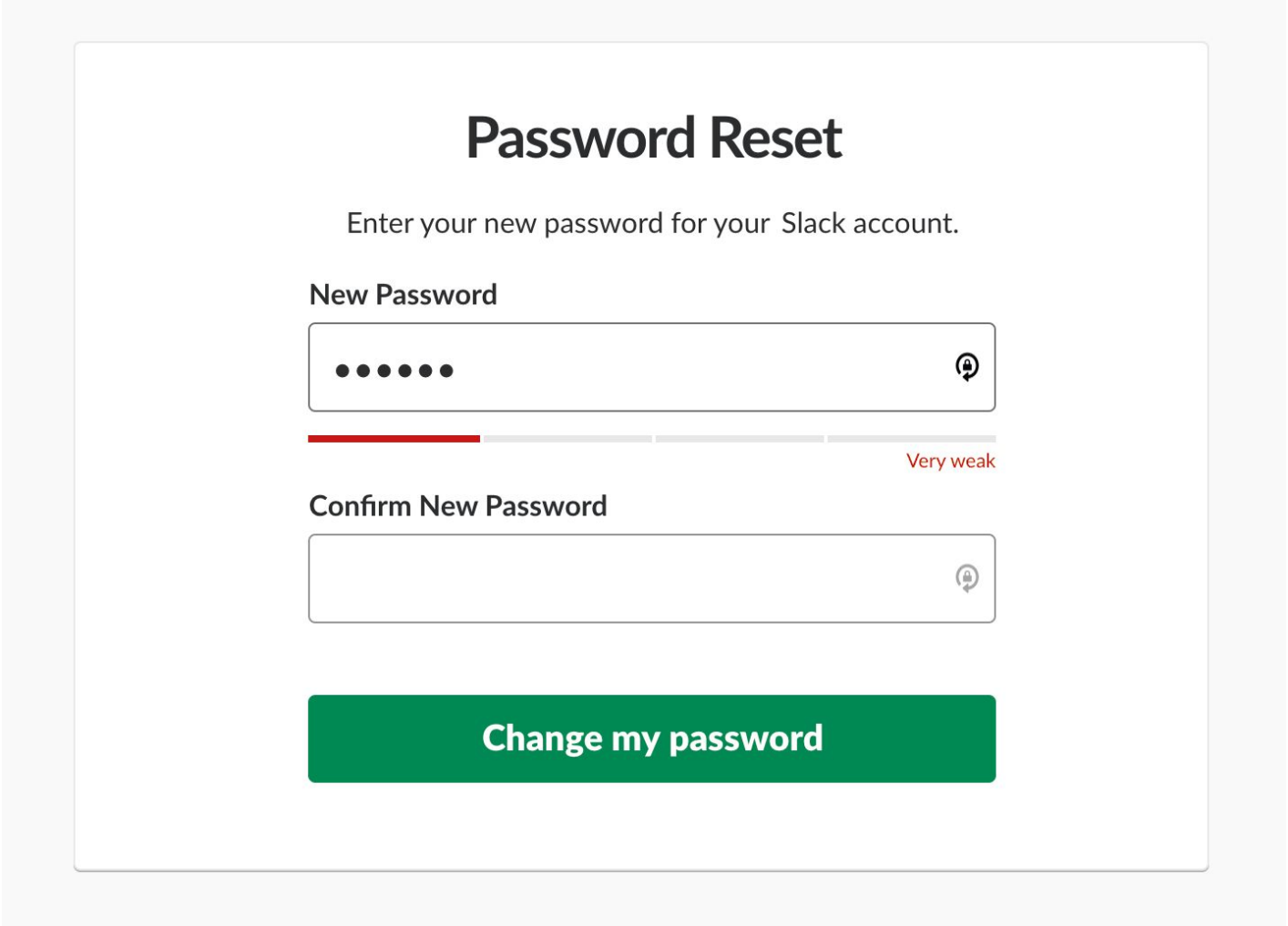
Validaciones ⇒ Errores comunes

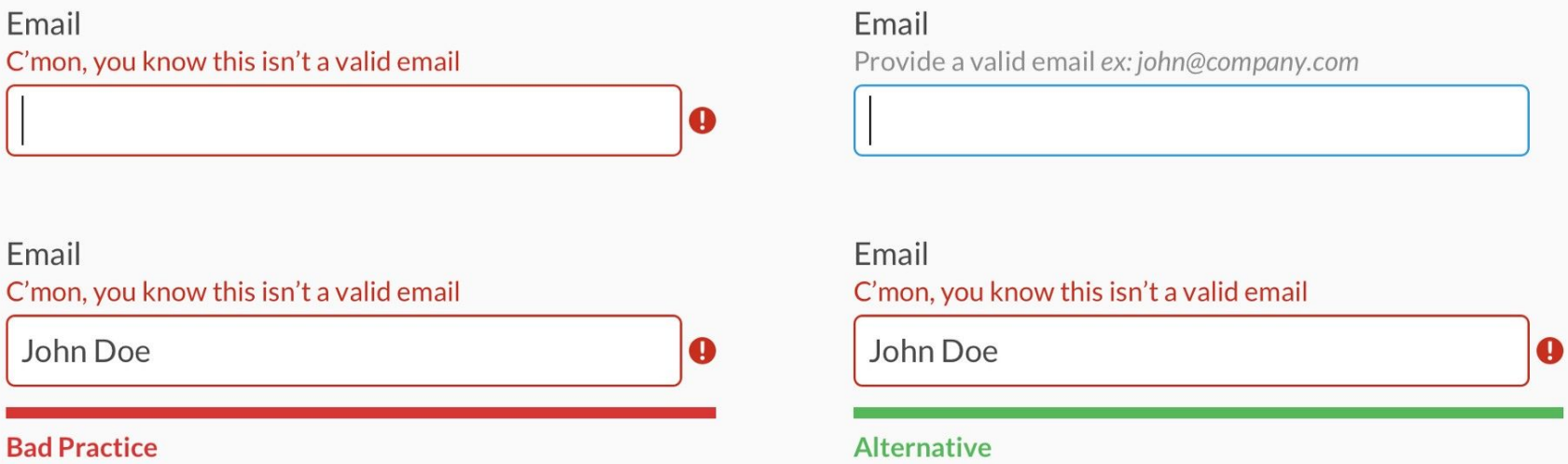
- Esperar a que el usuario envíe el formulario para mostrarle los errores
- Mostrar los errores en un modal que se cierra al querer ir a corregir los errores
- Utilizar los mensajes de error provenientes del core tecnológico

Validaciones ⇒ Mejores estrategias

- Proporcionar información en el momento (si es útil)
- Mostrar los mensajes de error en el lugar más apropiado para el que el usuario puede hacer la corrección
- Informar en un lenguaje claro y sencillo cuál es el problema, sin utilizar lenguaje técnico


Personal Information
First Name *
Last Name *
Contact Information
Address *
City *
Zip Code
First Name *
Last Name *
Address *
City *
Zip Code


Password Reset
Enter your new password for your Slack account.
New Password
Very weak
Confirm New Password
Change my password


Email
C'mon, you know this isn't a valid email
Email
Provide a valid email ex: john@company.com
Email
C'mon, you know this isn't a valid email
John Doe
Email
C'mon, you know this isn't a valid email
John Doe
Bad Practice
Alternative

Join Twitter today.

Full name

Marcin Treder

✓ Name looks great.

Email address

marcin@uxpin.com

✗ This email is already registered. Want to login or recover your password?

Create a password

6 characters or more! Be tricky.

Choose your username

Don't worry, you can change it later.

Suggestions: MTreder · TrederMarcin · TrederMarcin ·


Sorry, 8 errors prevented this registration from being submitted. Please correct any errors inc
again.
The passwords you typed do not match. Please type them again and ensure that they match.
Please enter a valid email address.
You must fill in your date of birth.
Date of birth day selection has an invalid value.
Date of birth month selection has an invalid value.
Date of birth year selection has an invalid value.
You may not join deviantART without agreeing to the terms of service.
You may not join deviantART without agreeing to abide by its etiquette policy.
Please, do not forget to retype your password before submitting.
Deviant name
fiewonfoiewn
Only letters, numbers, and hyphens can be used
Password
Retype password
Real Name
Bob
Will be displayed on your profile
Email Address
bob.com
Retype Email Address
bob.com
A confirmation email will be sent to your email address

Whoops! There were some problems with your input.

- The first name must be at least 5 characters.
- The last name field is required.
- The email must be a valid email address.
- The mobileno must be a number.
- The password field is required.
- The confirm password field is required.
- The details field is required.

First Name:

Har

The first name must be at least 5 characters.

Last Name:

Enter Last Name

The last name field is required.

Email:

itsolutionstuff@

The email must be a valid email address.

Mobile No:

mobile

The mobileno must be a number.

Password:

Enter Password

The password field is required.

Confirm Password:

Enter Confirm Passowrd

The confirm password field is required.

Details:

Enter Details

The details field is required.

Submit

Validaciones cheat sheet

Hay que intentar limitar la frustración que el usuario **siente** al “equivocarse”. Si se frusta, cierra y se va.

- No culpar al usuario
- Escribir como humano y para humanos
- Asegurarse que los mensajes sean claros y que la forma de corregir los errores sea intuitiva
- Decidir si conviene enumerar todos los errores en la parte superior de la pantalla o usar validación en línea

Validaciones en la Vista

- Pueden ser realizadas sin la intervención del modelo de negocio
- Se realizan en la misma capa
- Son menos costosas porque
  - No es necesario consultar al modelo
  - Generalmente alcanza con un análisis sintáctico
- Ejemplos
  - El campo “usuario” no puede estar vacío
  - El formato del “mail” es incorrecto
  - El “password” debe contener al menos 8 caracteres, alguna mayúscula y algún número

Validaciones en el Modelo

- Implican lógica de negocio
- La vista debe “re-decodificar” el error para presentarlo correctamente
- Son más costosas porque requieren comunicación entre capas
- Ejemplos
  - El “nombre de usuario” ya está ocupado
  - El número de tarjeta no es válido
  - Tenés que tener dinero en la cuenta para poder operar

Excepciones

- *Exception* es la abreviatura de "Exceptional Event"
- Se puede definir como:
  - Un evento que ocurre durante la ejecución de un programa que **interrumpe** el *flujo normal* de instrucciones
- Significa que algo salió mal y no podemos continuar
- Hay mucha discusión acerca del uso de excepciones
  - Si lanzarlas o retornar valores de error
  - Cuándo y cómo usarlas
  - Handlearlas o dejarlas subir
  - Etc ...

Excepciones » Cuando Sí

TIP: *Lanzar una excepción cuando no se cumple la precondición del método.*

Ejemplo

- La función `isInfluencer(user)` retorna `true` si el `user` tiene más de 50k de seguidores, `false` sino
  - ¿Qué pasa si el usuario no existe?
  - Retornar `false` en ese caso, ¿qué significa?
- El método `transferMoneyTo(amount, account)` transfiere una cantidad de plata de mi cuenta a otra
  - ¿Si `account` existe pero está bloqueada?
  - ¿Tengo la suficiente cantidad de plata?

Excepciones » Cuando Sí

- Muchas precondiciones se pueden validar dentro de la función o antes de invocarlas.
- Lanzar o no excepciones puede depender también del alcance y el control que tengamos sobre la interfaz de funciones que proveemos
- Si no sabemos quién y cómo va a terminar usando nuestras clases, es preferible tirar excepciones

Excepciones » Cuando No

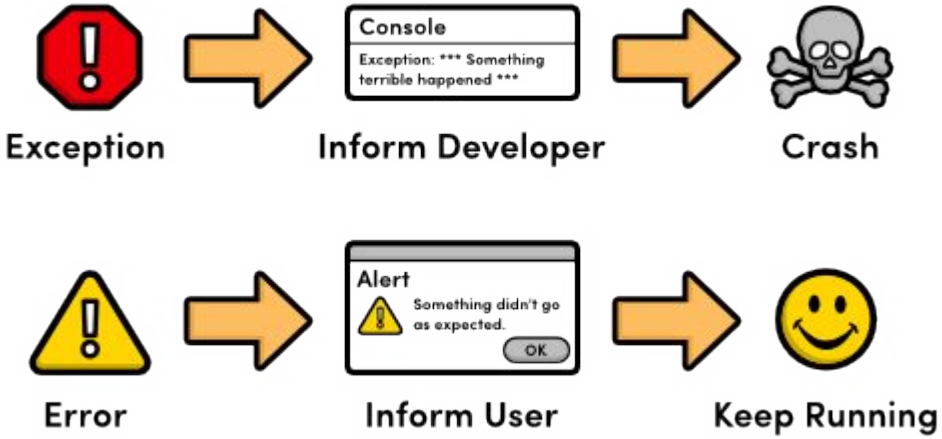
TIP: *Cuando el return es lo suficientemente declarativo*

Ejemplo

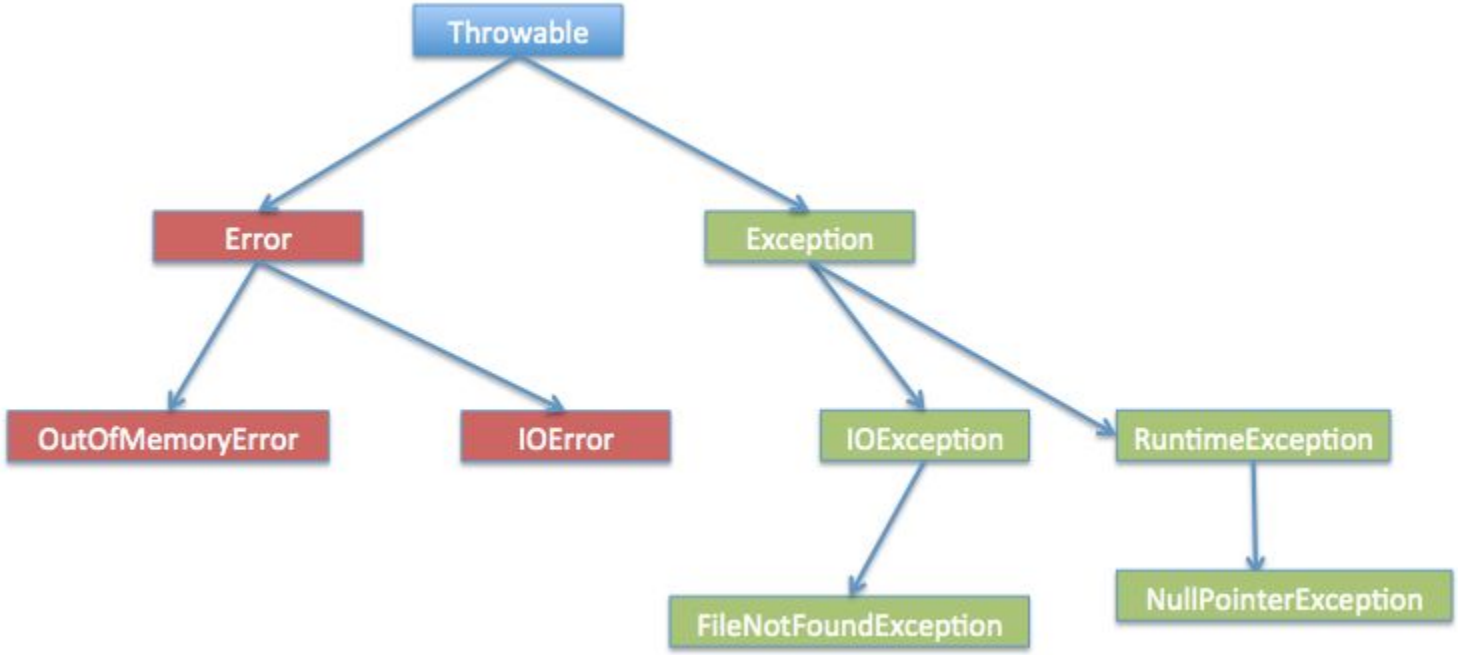
- `existsUser(user)` retorna `true` o `false` dependiendo si el usuario existe en la BD
  - No hay terceras posibilidades
- `favMovies(user)` retorna una lista de películas favoritas y sabemos que el usuario es válido (login)
  - Si no tiene películas favoritas alcanza con retornar una lista vacía

Excepciones » Decisiones

- Si el usuario tiene que conocer del error » Mostrarlo
- Si es algo que no tenía que pasar » Que rompa
- Aunque se puede capturar “al final” y mostrar “Ups, pasaron cosas” pero informar del problema internamente.



Jerarquía de Excepciones (en Java)



Excepciones » Arena

- Arena cuenta con la excepción `UserException`
- Es una excepción “especial” porque es *handleada* por el propio framework
- Se utiliza para facilitar la exposición de mensajes de error hacia el usuario
- El mensaje de error puede mostrarse en una ventana (tipo diálogo) o utilizar un panel especial llamado `ErrorPanel`
- Cada vez que queramos informar un error al usuario debemos lanzar una `UserException`

Ya pueden seguir codeando