Universidad Nacional de Quilmes

CONSTRUCCIÓN DE INTERFACES DE USUARIO

2do Cuatrimestre de 2018

4.2.

NAVEGACIÓN

WINDOWS & DIALOGS

Ventana Principal

Hasta el momento venimos usando una sola ventana:

- Creamos una LoQueSea`Window` que extienda de `MainWindow<T>`
- Definimos el constructor pasándole el modelo
- Sobreescribimos el método `createContents` con el contenido
- Y levantamos la aplicación desde el `main`

Ventana Principal

```
class MainCompanyWindow extends MainWindow<Company> {
  new(Company model) { super(model) }
  static def void main(String[] arg) {
    new MainCompanyWindow(new Company).startApplication
  }
  override createContents(Panel mainPanel) {
    this.title = "Empresa"
    new Label(mainPanel).text = "Empleados: "
    new List<Employee>(mainPanel) => [
      value <=> "empl"
      (items <=> "employees")
        .adaptWith(Employee, "description")
   ]
  }
}
```

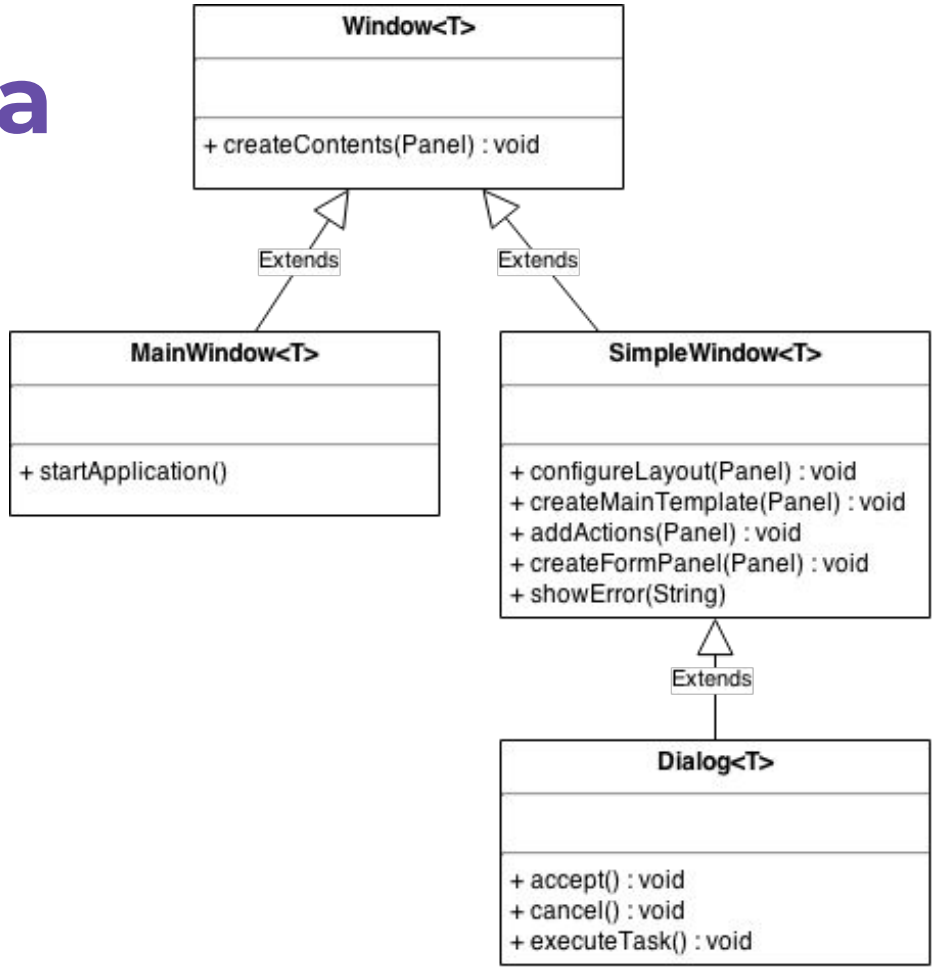
¿Y cómo levanto otra ventana?

- A partir de algún evento podemos instanciar una nueva ventana
- Esta ventana debe conocer a la ventana que la invocó (solemos llamarla owner or parent)
- Y también debe tener un model object, el cual debe ser “pasado” desde el parent

Tipos de Ventanas

- **`Window`**: es la clase abstracta para todas las ventanas
- **`MainWindow`**: es un tipo especial de ventana que se usa para aplicaciones simples o de una sola ventana
- **`SimpleWindow`**: ventana común que agrega el panel de errores
- **`Dialog`**: es una ventana final que depende de alguna de las anteriores y que debe generar una acción y cerrarse

Jerarquía

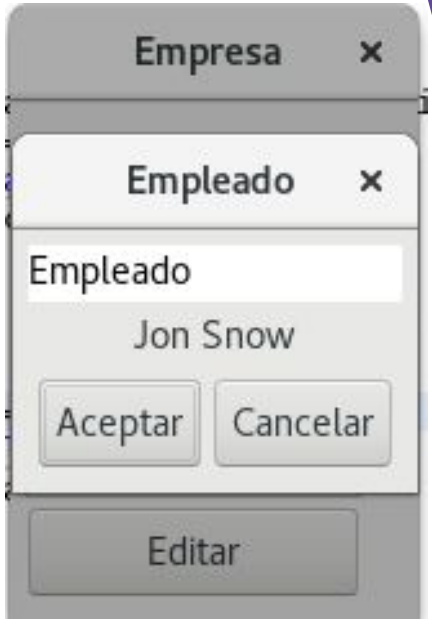

Window<T>
+ createContents(Panel) : void
Extends
Extends
MainWindow<T>
+ startApplication()
SimpleWindow<T>
+ configureLayout(Panel) : void
+ createMainTemplate(Panel) : void
+ addActions(Panel) : void
+ createFormPanel(Panel) : void
+ showError(String)
Extends
Dialog<T>
+ accept() : void
+ cancel() : void
+ executeTask() : void

Window ⇒ Dialog

```
class MainCompanyWindow extends MainWindow<Company> {
  override createContents(Panel mainPanel) {
    ...
    new Button(mainPanel) => [
      caption = "Editar"
      onClick([|this.edit])
    ]
  }
  def edit() {
    new EmployeeDialog(this, modelObject.employees.get(0)) => [
      onAccept[this.modelObject.employees.add(
                  new Employee("White Walker", 0))
      ]
      open
    ]
  }
}
```

Dialog

```
class EmployeeDialog extends Dialog<Employee> {
  new(WindowOwner owner, Employee model) {
    super(owner, model)
  }
  override protected void addActions(Panel actions) {
    new Button(actions) => [
      caption = "Aceptar"
      onClick [|this.accept]
    ]
    new Button(actions) => [
      caption = "Cancelar"
      onClick [|this.cancel]
    ]
  }
  override protected createFormPanel(Panel mainPanel) {
    new Label(mainPanel).text = this.modelObject.name
  }
}
```



Window ⇒ Window

- Las ventanas `Dialog` funcionan como “modales”, o sea que se usan para una función específica y se cierran.
- Se pueden seguir abriendo ventanas o dialogs desde un `Dialog` pero no es recomendable porque se van “stackeando”
- Para poder trabajar con ventanas independientes deben ser `Window` o `SimpleWindow`.
- Una `MainWindow` puede abrir una `Window` pero luego nunca más se puede volver a la `MainWindow`.

Window ⇒ Window

- Pero las ventanas `Window` no pueden ser inicializadas desde un `main()` como sí sucedía con `MainWindow`
- Es necesario otra estrategia de inicialización
  - Hay que usar la clase `Application`
  - Que se encarga de inicializar la aplicación y llamar a la `Window` que indiquemos como "inicial"
  - Luego vamos a poder interactuar entre `Windows` y `Dialogs` libremente

Application ⇒ Window

```
class MyApplication extends Application {
  override protected Window<?> createMainWindow() {
    new CompanyWindow(this)
  }
  def static main(String[] args) {
    new MyApplication().start()
  }
}
```

Window ⇔ Window

```
class CompanyWindow
extends SimpleWindow<Company> {
  new(WindowOwner parent) {
    super(parent, new Company)
  }
  override createContents(Panel mainPanel) {
    this.title = "Empresa"
    new Button(mainPanel) => [
      caption = "Abrir Empleado"
      onClick[
        this.close
        new EmployeeWindow(this,
          this.modelObject.employees.get(0)
        ).open
      ]
    ]
  }
}
```

```
class EmployeeWindow
extends SimpleWindow<Employee> {
  new(WindowOwner owner, Employee model) {
    super(owner, model)
  }
  override createContents(Panel mainPanel) {
    this.title = "Empleado"
    new Button(mainPanel) => [
      caption = "Volver a Empresa"
      onClick[
        this.close
        new CompanyWindow(this).open
      ]
    ]
  }
}
```

Demo