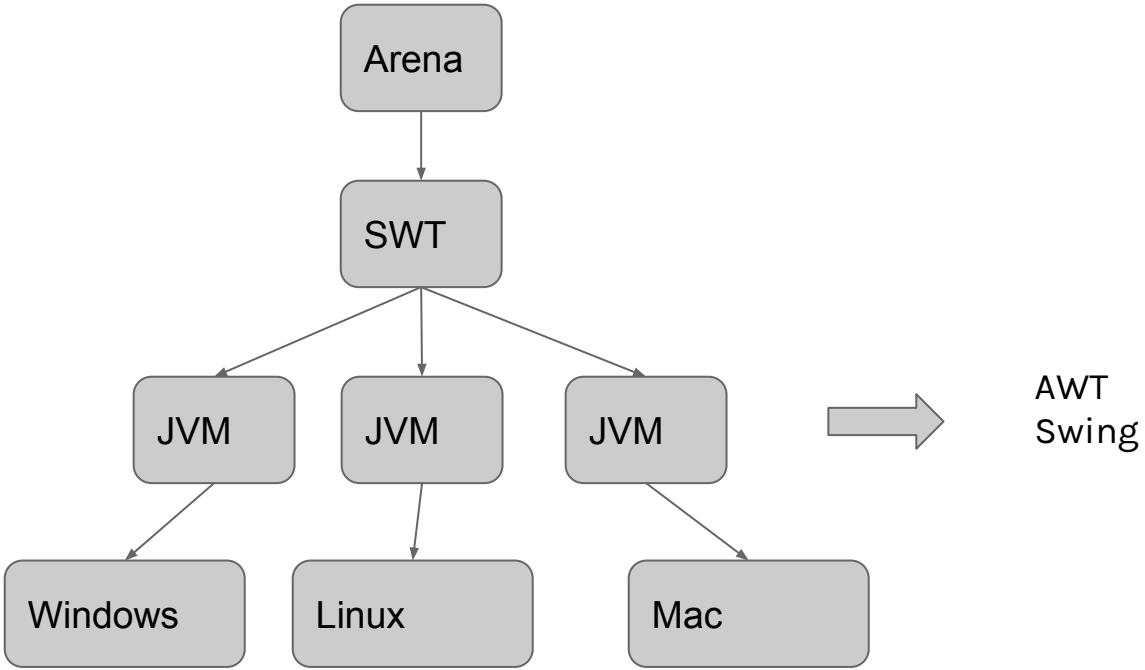
Universidad Nacional de Quilmes

CONSTRUCCIÓN DE INTERFACES DE USUARIO

1er Cuatrimestre de 2019

Interfaces Desktop Multiplataforma



Interfaces Desktop Arena

- Framework para facilitar la construcción de interfaces Desktop desarrollado por Uqbar con fines educativos.
- Basado en SWT (Standard Widget Toolkit): conjunto de componentes para construir interfaces gráficas en Java, (widgets) desarrollados por el proyecto Eclipse.
- Link a Arena: http://arena.uqbar-project.org/index.html

Arena - Maven

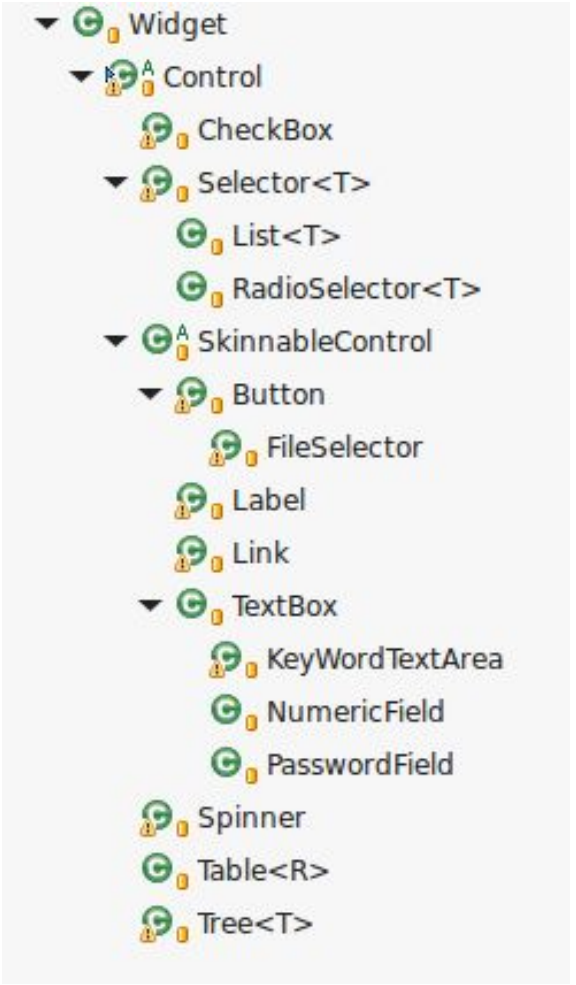
```
<repositories>
  <repository>
    <id>uqbar</id>
    <name>Uqbar</name>
    <url>http://maven.uqbar.org/releases/</url>
  </repository>
</repositories>


<parent>
  <groupId>org.uqbar-project</groupId>
  <artifactId>arena-xtend-parent</artifactId>
  <version>3.6.3</version>
</parent>
```

Interfaces Desktop Componentes

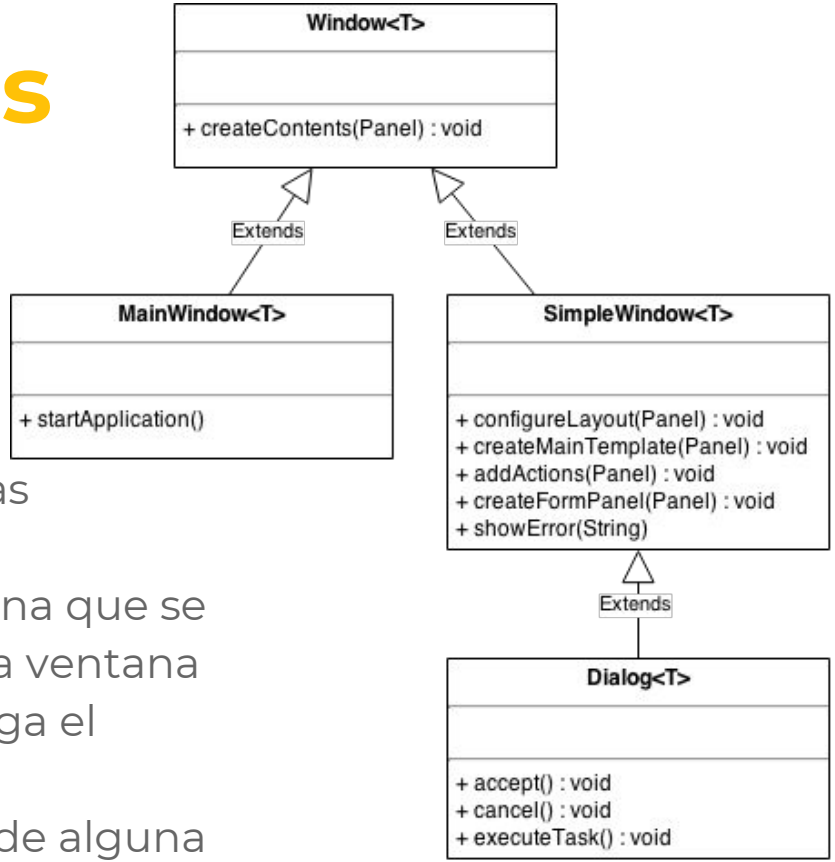
- TextBox - Input de texto
- Label - Etiqueta de texto
- Button
- Panels - Contenedor de componentes
- Windows y Dialogs
- Error Panel

Ver documentación Arena:

http://arena.uqbar-project.org/documentation/components.html



Ventanas y Diálogos



- **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

Ejemplo - MainWindow

```
class LoginWindow : MainWindow<Login>(Login()) {
    override fun createContents(panel: Panel) {
        this.title = "Login del sistema"

        Label(panel).setText("Ingrese el usuario")
        TextBox(panel)

        Label(panel).setText("Ingrese el password")
        PasswordField(panel)

        Button(panel).setCaption("Autenticar")
    }
}
fun main() {
    LoginWindow().startApplication()
}
```

Windows - Dialogs

- Pero las ventanas `Window` no pueden ser inicializadas desde un `main()` como sí sucede 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 + SimpleWindows

```
class Main : Application() {
   override fun createMainWindow(): MainWindow {
      return MainWindow(this, User())
   }
}

fun main() { Main().start() }

class MainWindow : SimpleWindow<User>{
   constructor (owner: WindowOwner, model: User ) : super(owner, model)
   override fun addActions(actionsPanel : Panel) { /* Actions Panel */  }
   override fun createFormPanel(mainPanel : Panel) {
       Label(mainPanel).setText("Prueba SimpleWindow")
   }
}
```

Application + Dialog

```
class FormWindow : SimpleWindow<User> {
  constructor (owner: WindowOwner, model: User) : super(owner, model)

  override fun createFormPanel(mainPanel: Panel) {
    Button(mainPanel).onClick(Action {
      AddNameDialog(this, this.modelObject).open()
    })
  }
}
class AddNameDialog : Dialog<User> {
   constructor(owner: WindowOwner, model: User ) : super(owner, model)
   override fun createFormPanel(panel : Panel ) {
      Label(panel).setText("Dialog Example: ")
      Button(panel).setCaption("Aceptar").onClick(Action { this.close() })
   }
}
```

Conceptos necesarios
Eventos

- Cualquier suceso en el sistema
- Comunicación entre dos componentes
- Inversión de la relación de conocimiento
- Tanto la vista como el modelo, pueden producir eventos

Conceptos necesarios Binding

- Vincular o enlazar propiedades del modelo con la UI.
- Relaciona dos propiedades manteniendolas sincronizadas
- El binding puede ser direccional o bidireccional
- Muchos framework proveen mecanismos de binding.

Binding - Arena

En la UI podemos “bindear” una propiedad con el modelo de la siguiente forma:

```
Label(panel).bindValueToPropertie<String, ControlBuilder>("user")
```

En el modelo con el annotation

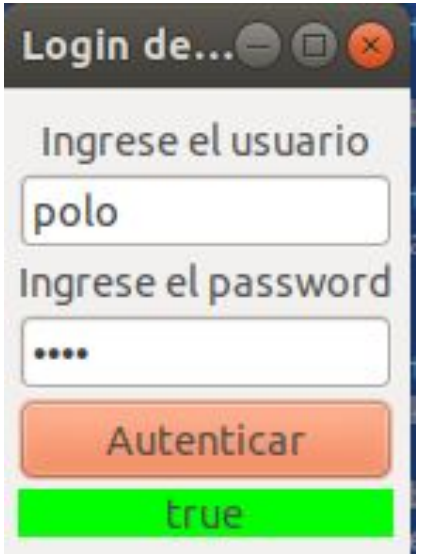
```
@Observable
class Login {
   var user: String =  ""
}
```

Modelo + Binding

```
@Observable
class Login {
    var user: String =  ""
        set(value) { resetAuth(); field = value }
    var password: String = ""
        set(value) { resetAuth(); field = value }
    var authenticated: Boolean = false
    var authenticatedColor: Color = Color.ORANGE

   fun authenticate() {
        authenticated = (user == "polo") && (password == "polo")
        authenticatedColor =
                if (authenticated) Color.GREEN else Color.ORANGE
    }
    private fun resetAuth() {
        authenticated = false
        authenticatedColor = Color.ORANGE
    }
```

Windows + Binding

```
class LoginWindow : MainWindow<Login>(Login()) {
   override fun createContents(panel: Panel) {

       this.title = "Login del sistema"
       Label(panel).setText("Ingrese el usuario")
       TextBox(panel).bindValueToProperty<Int, ControlBuilder>("user")

       Label(panel).setText("Ingrese el password")
       PasswordField(panel).bindValueToProperty<Int, ControlBuilder>("password")

       Button(panel)
               .setCaption("Autenticar")
               .onClick { modelObject.authenticate() }

       val status = Label(panel)
       status.bindValueToProperty<Double, ControlBuilder>("authenticated")
       status.bindBackgroundToProperty<ControlBuilder, Any, Any>("authenticatedColor")
    }
}
fun main() {
   LoginWindow().startApplication()
}
```



¿Preguntas?

¿Hacemos una Demo?