Recordando Java…

```java
public class MyFirstJavaProgram {
  public static void main(String []args) {
        System.out.println("Hello World");
    }
}
```

Objetos y Tipos Primitivos

Soporte a Objetos y a Tipos Primitivos

```
int n = 5;
String str = “hola”;
```

Los tipos primitivos tienen su contrapartida en forma de objetos

```
int n = 5;
Integer nObj = 5;
```

Los objetos se inicializan con “new”

```
Perro p = new Perro();
```

¡Ojo! ¡Los tipos primitivos se pasan por valor, mientras que los objetos se pasan por referencia!

Tipos útiles

- Byte, Short, Integer, Long, Float, Double, Character y Boolean
- byte, short, int, long, float, double, char y boolean
- String
- Date, Calendar y SimpleDateFormat
- **Arrays:** Tipo[] array = new Tipo[n];
- **Colecciones (más usadas):** HashMap<TipoClave,TipoValor>, ArrayList<Tipo>, TreeSet<Tipo>
- **¡Ojo!** ¡ Podemos crear arrays de tipos primitivos u objetos! ¡Solo podemos crear colecciones de objetos, no de tipos primitivos!

Operadores

- +, -, *, /, %: Funcionan tanto en tipos primitivos como en sus contrapartidas como objetos. En el caso de los objetos, no mutan:

```
Integer n=5;
n++; // n = new Integer(n+1);
```

- + funciona en String para concatenar

- Operadores booleanos y binarios equivalentes a C++: &&, ||, !, ^, &, |, etc.

Estructuras de control

- Equivalente a C++: if, switch, while, for, do…while, while

- for (Tipo variable : Colección<Tipo>) { }

```
ArrayList<Perro> perros;
…
for (Perro p : perros) {
    System.out.println(p.name);
}
```

Excepciones

```
public class FileInputStream {
    public FileInputStream(String filename) throws
FileNotFoundException{/*…*/}
    public void read() throws IOException {
       /*…*/
       throw new IOException(/*…*/);
    }
}
//…
try {
    file = new FileInputStream(fileName);
    x = (byte) file.read();
}
catch(IOException i) {
    i.printStackTrace();
    return -1;
}
catch(FileNotFoundException f){
    f.printStackTrace();
    return -1;
}
```

Clases

```
public class Perro {
    int age;
    String name;

    public Perro(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public int getAge( ) { //Getter
        return age;
    }

    public int setAge(int newAge) { //Setter
        this.age = newAge;
    }
}
```

Atributos

Constructor/es

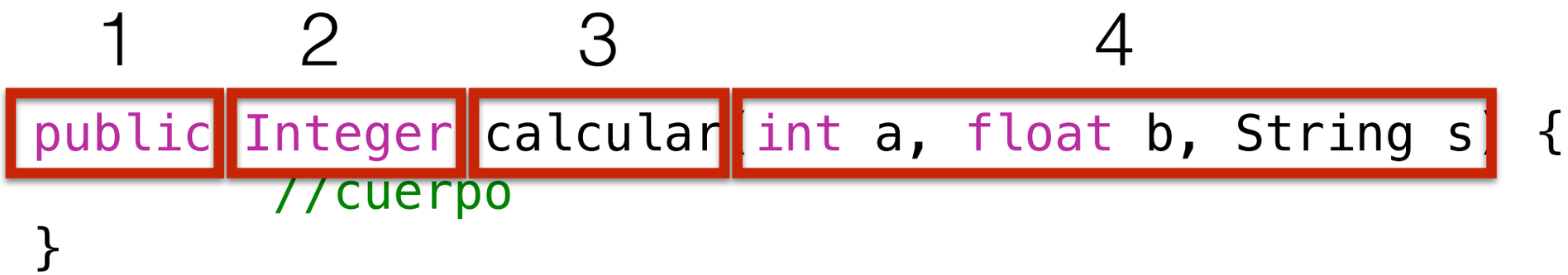
Métodos

Constructores

Deben tener como parámetros los necesarios para que el conjunto de atributos de la clase tengan valores válidos en mi sistema.

```
public class Perro {
    …
    public Perro(String name, int age) {
        this.name = name;
        this.age = age;
    }
    …
}
```

Métodos



1. Modificadores: public, protected, private
2. Tipo devuelto (tipos primitivos u objetos)
3. Nombre del método
4. Parámetros (tipos primitivos u objetos)

Herencia

```
class Calculation {
    int z;

    public void addition(int x, int y) {
        z = x + y;
    }

    public void subtraction(int x, int y) {
        z = x - y;
    }
}

public class My_Calculation extends Calculation {
    public void multiplication(int x, int y) {
        z = x * y;
    }
}
```

Acceso a atributos/métodos “public” o “protected” de la superclase

Interfaces y clases abstractas

```
interface Animal {
    void comer();
    void mover();
}
```

```
public class Mamifero implements Animal {
  …
    public void comer() {
        System.out.println("Mamifero Come");
    }
    public void mover() {
        System.out.println("Mamifero Se Mueve");
    }
  …
}
```

```
public abstract class Mamifero implements Animal {
    public abstract int numeroPatas();
    …
}
public class Perro extends Mamifero {
    public int numeroPatas() {
        return 4;
    }
}
```

Polimorfismo

```
public void hacerCosas(Animal a){
    a.comer();
    a.mover();
    System.out.println(a.numeroPatas()); //ERROR: “a” no es Mamifero
}
public void mostrarPatas(Mamifero m) {
    System.out.println(m.numeroPatas());
}
...
Perro p = new Perro();
Animal a = new Perro(); //Perro es Animal
hacerCosas(p); //Perro es Animal
mostrarPatas(p); //Perro es Mamifero
```

¡Podemos especificar métodos sin saber qué clases las usarán!

Java SDK

- Java ofrece un conjunto de clases para poder realizar funcionalidades avanzadas sin instalar bibliotecas externas

- E/S archivos, comunicación por red, interfaces de usuario, programación concurrente, etc.

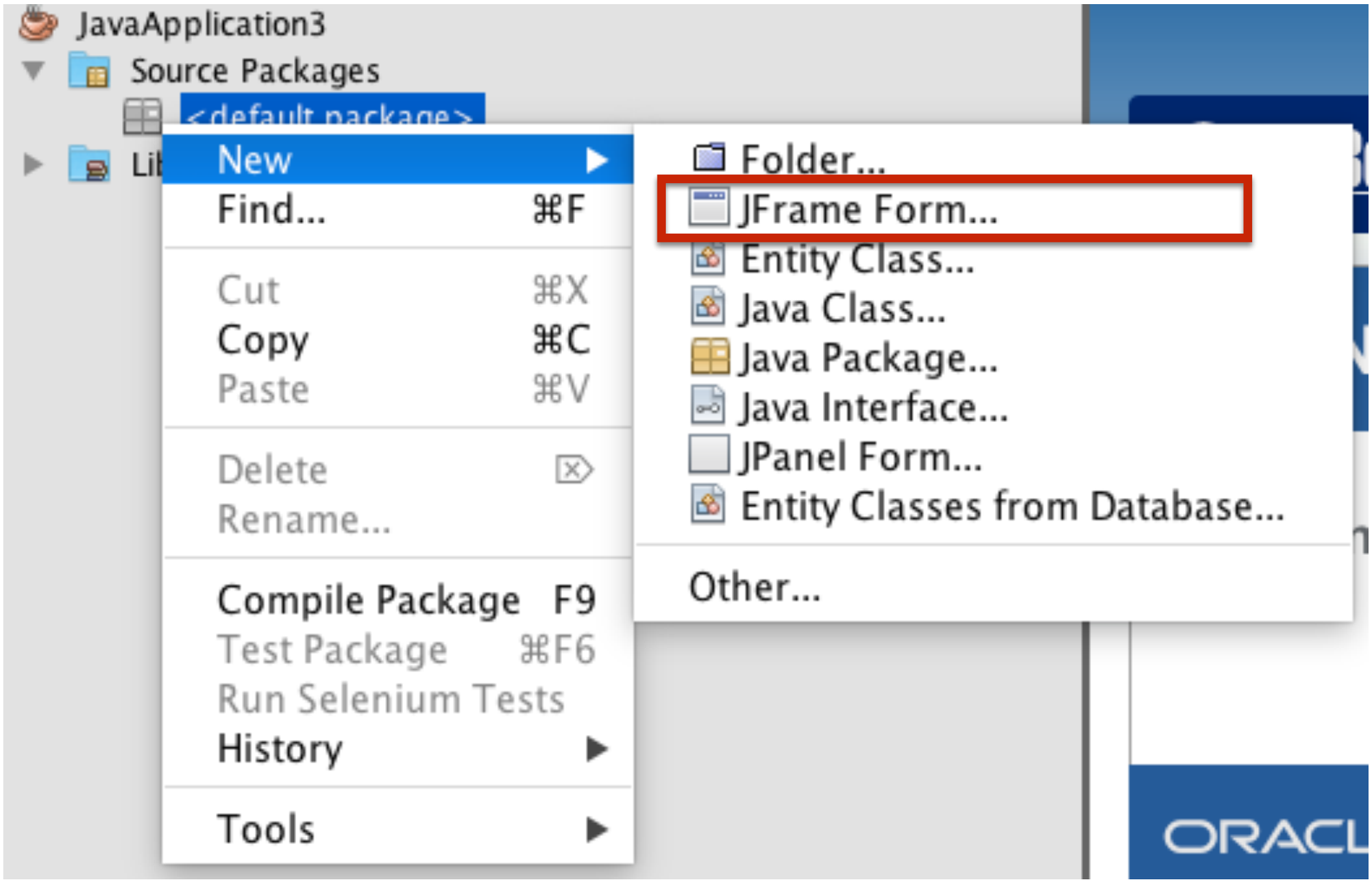
Interfaces de usuario

Netbeans

Comenzando…

- Abrir Netbeans
- Menú “File > New Project… > Java > Java Application”
- Seleccionar “Next” y poner un nombre al proyecto, además de una carpeta de destino
- Desactivar “Create Main Class”

Crear ventanas



Diseñar ventanas

Paleta de widgets

Start Page | VentanaPrincipal.java

Source | Design | History

Selector entre modo código y modo diseño

Área de diseño

Palette

- Swing Containers
  - Panel
  - Tabbed Pane
  - Split Pane
  - Scroll Pane
  - Tool Bar
  - Desktop Pane
  - Internal Frame
  - Layered Pane
- Swing Controls
  - Label
  - Button
  - Toggle Button
  - Check Box

[JFrame] - Properties

Properties | Binding | Events | Code

Properties

defaultCloseOperatio	EXIT_ON_CLOSE
title	

Other Properties

alwaysOnTop	

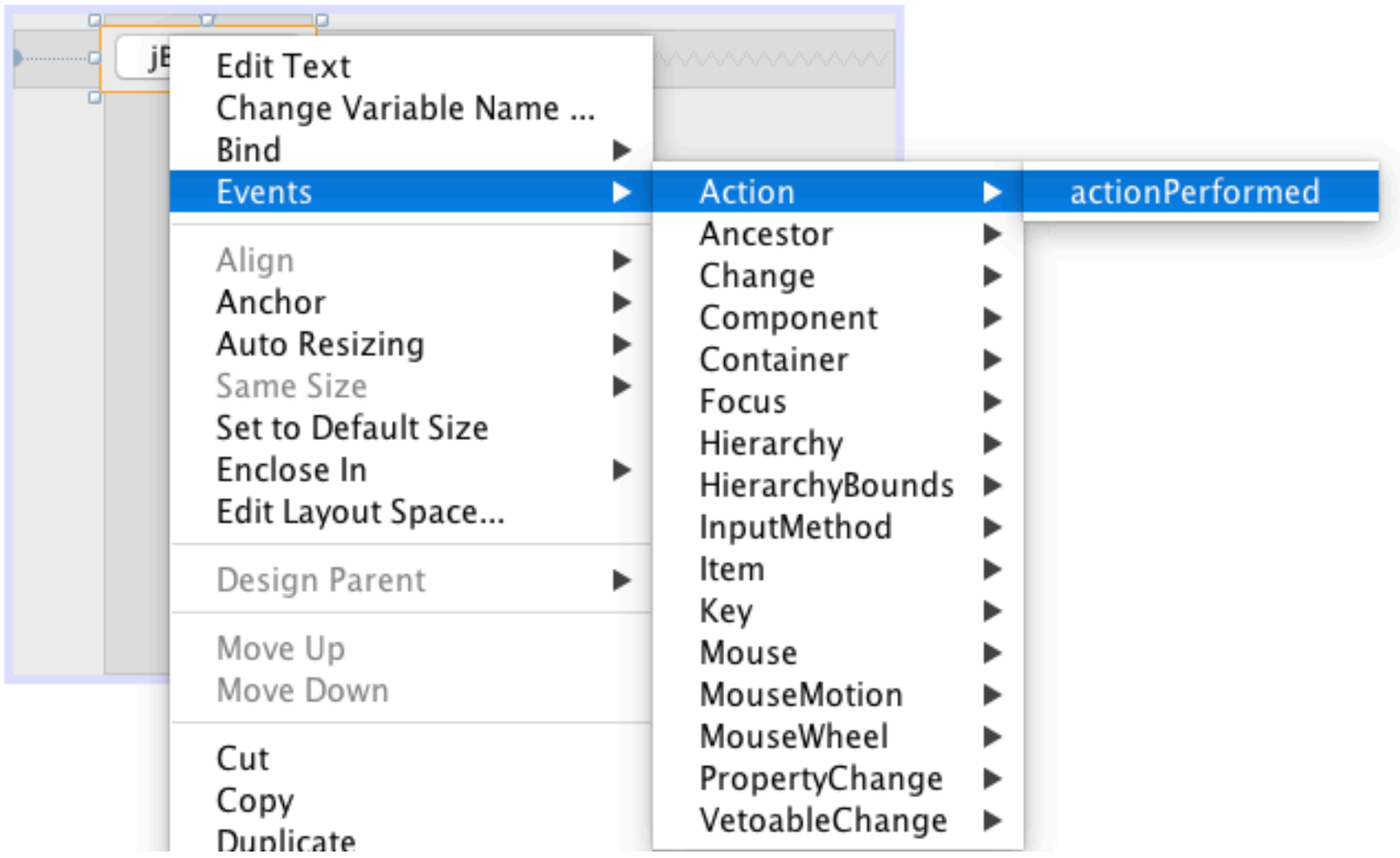
[JFrame]

Configuración del widget seleccionado

Posicionar widgets

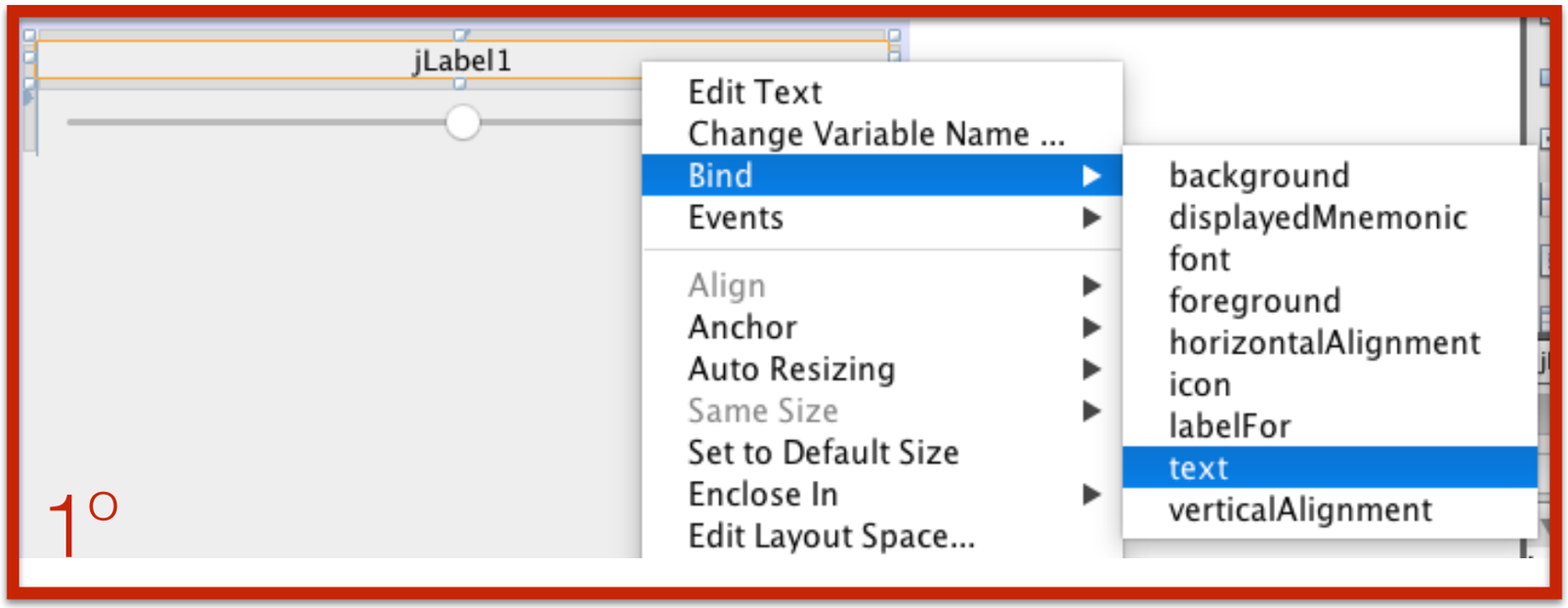
¡Probar como afecta el redimensionamiento de las ventanas a la posición de los widgets!

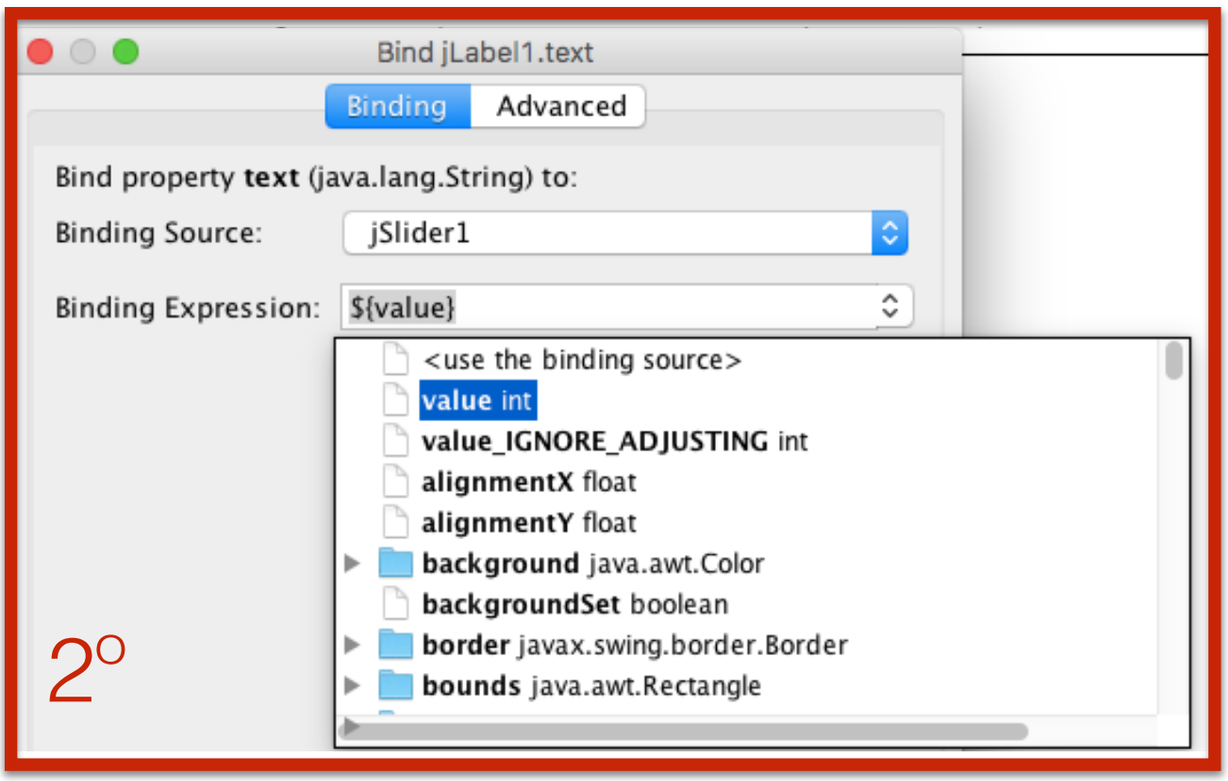
Asociar eventos

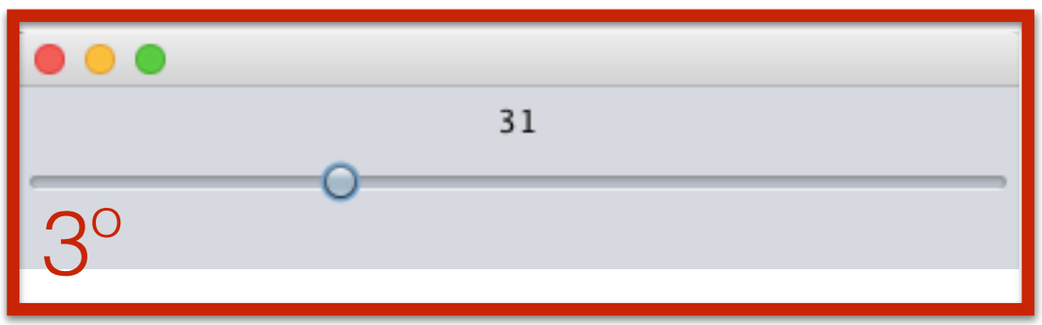



```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
```

Vincular propiedades

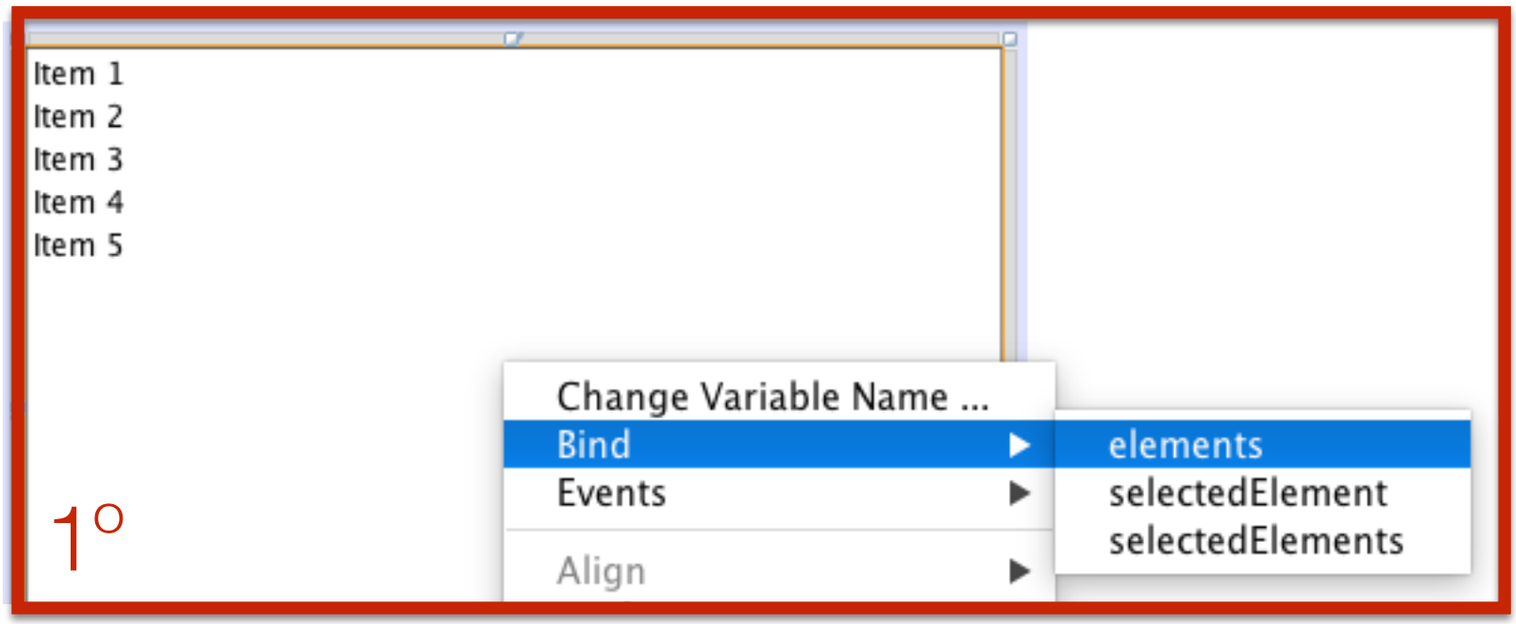


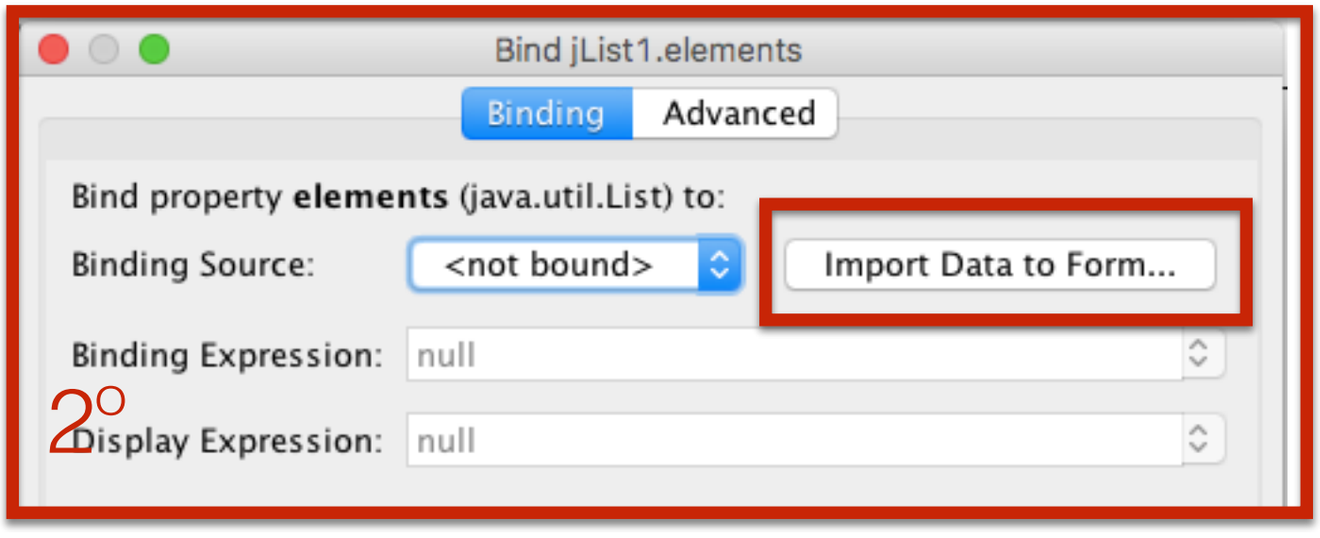




¡Sin código fuente!

Vincular DB (JList)





¡Sin código fuente!

Listas y Tablas (JList y JTable)

Los datos que contienen las tablas los provee un **Model** (modelo de datos)

```
public class CustomListModel extends AbstractListModel{

    private ArrayList<Persona> lista = new ArrayList<>();
    //obligatorios
    @Override
    public int getSize() {
        return lista.size();
    }
    @Override
    public Object getElementAt(int index) {
        Persona p = lista.get(index);
        return p.getNombre();
    }
    //auxiliares
    public void addPersona(Persona p){
        lista.add(p);
        this.fireIntervalAdded(this, getSize(), getSize()+1);
    }
    public void eliminarPersona(int index0){
        lista.remove(index0);
        this.fireIntervalRemoved(index0, getSize(), getSize()+1);
    }
    public Persona getPersona(int index){
        return lista.get(index);
    }
}
```

```
CustomListModel listModel = new CustomListModel();
lista.setModel(listModel);
```

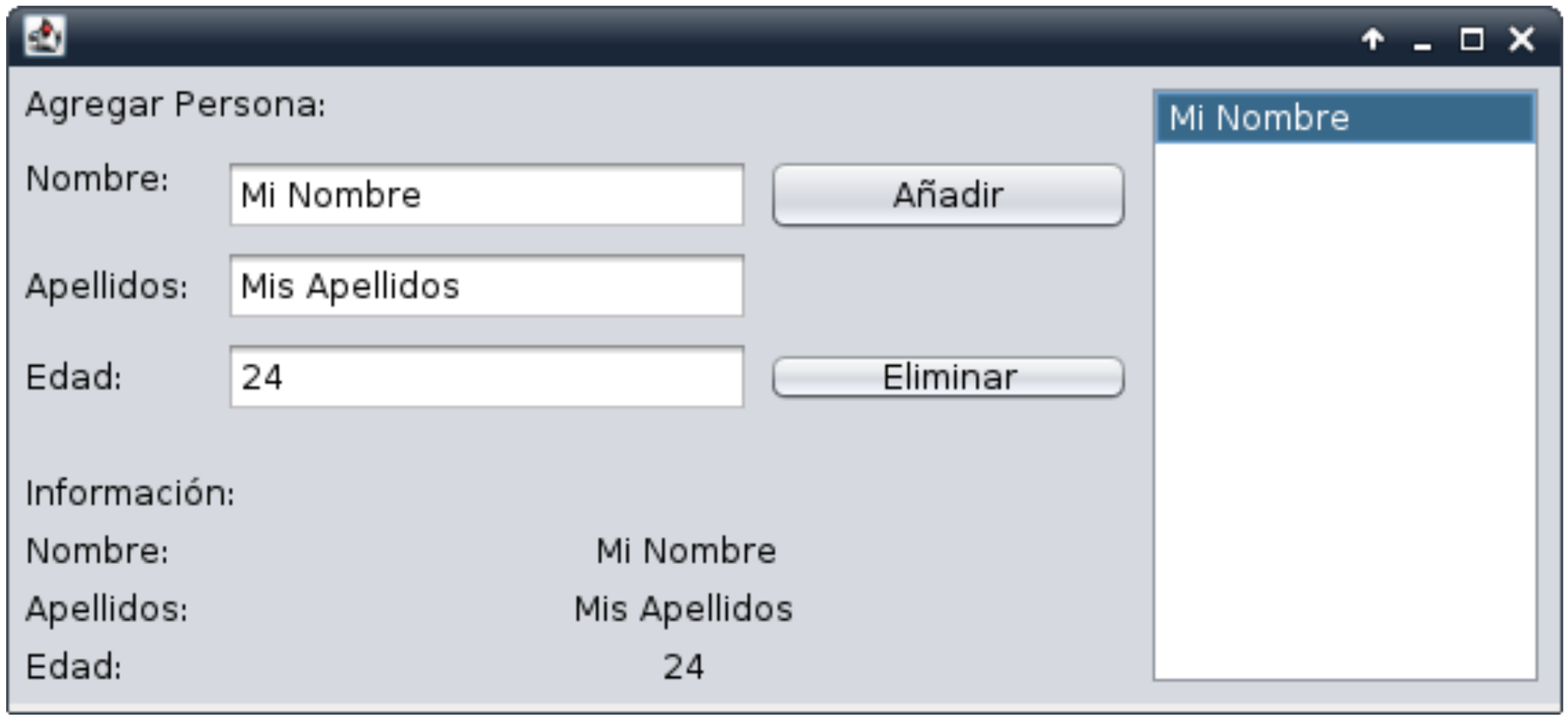
Listas y Tablas (JList y JTable)

- Para modelos de datos muy simples: Clase por defecto **DefaultListModel<Tipo>** (“Tipo” normalmente es String)

- Para JTable, todas las clases son equivalentes a las de JList, pero con la palabra Table en vez de List (ej: DefaultTableModel<Tipo>). También hay métodos que hacen referencia a las columnas.

```
DefaultListModel<String> model = new DefaultListModel<>();
model.addElement("Alumno 1");
model.addElement("Alumno 2");
model.addElement("Alumno 3”);
lista.setModel(model);
```

Ejercicio:


Agregar Persona:
Nombre:
Mi Nombre
Añadir
Apellidos:
Mis Apellidos
Edad:
24
Eliminar
Información:
Nombre:
Mi Nombre
Apellidos:
Mis Apellidos
Edad:
24
Mi Nombre

Notas finales

- Mostrar alertas, entradas de texto, confirmaciones, etc.: Clase **JOptionPane** (métodos estáticos)

- Mostrar ventana: **(new Ventana()).setVisible(true);**

- El método **setDefaultCloseOperation** de la clase **JFrame** permite especificar qué hacer cuando pulsemos el botón cerrar de la propia ventana

- ¡Tratar de utilizar patrón **Modelo-Vista-Controlador**!

- ¡**Nombrar los atributos** que referencian a widgets adecuadamente!