Udemy | Programación Java SE desde 0

Fundamentos y estructuras básicas

Tipos de datos en Java

(Tipos Primitivos)

- **Enteros**
  - **Long**: 8 bytes - Sufijo L
  - **Int** : 4 bytes - Desde -2,147,483,648 hasta 2,147,483,647
  - **Short** : 2 bytes - Desde -32,768 hasta 32,767
  - **Byte**: 1 byte - Desde -128 hasta 127
- **Coma flotante** (decimales)
  - **Float**: 4 bytes - Aproximadamente 6 a 7 cifras decimales significativas. Sufijo F.
  - **Double**: 8 bytes - Aproxiimadamente 15 cifras decimales significativas.
- **Char** - Para representar caracteres (‘z’, ‘a’)
- **Boolean** - 2 únicos valores. **True | False**

Variables

Es un espacio en la memoria del ordenador donde se almacenará un valor que podrá cambiar durante la ejecución del programa
Se crean especificando el tipo de dato que almacenará más el nombre de la variable

```
Int salario;
```

Se inicia al darle un valor.

```
salario = 2000;
```

Estructura de carpeta programa java

- .**settings**
- **bin** | Aquí se guarda el archivo que lee la JVM. En formato .class
- **src |** El archivo con extensión .java se almacena en esta carpeta

Código bytecode. Archivo compilado que se guarda con la extensión .class. Está en un lenguaje a mitad de camino entre Java y el código máquina.

Constantes

Espacio en la memoria del ordenador donde se almacenará un valor que **no podrá cambiar** durante la ejecución del programa
Se crean agregando la palabra final y a continuación especificando el tipo de dato que almacenará en su interior y finalmente el nombre de la constante

```
Final double a_pulgadas = 2.54;
```

Es posible declalarla y luego iniciarla, pero normalmente se realiza todo de una vez.

Operadores en Java

- **Aritméticos**
  - **+** Suma
  - **-** Resta
  - ***** Multiplicación
  - **/** División
- **Lógicos, relacionales y booleanos**
  - **>** mayor que
  - **<** menor que
  - **<>** mayor o menor que
  - **!=** distinto que
  - **==** igual que
  - **&&** y lógico
  - **||** o lógico
- **Incremento** y decramento
  - **++** incremento
  - **--** decremento
  - **+=** nº incremento
  - **-=** nº decremento
- **Concatenación**
  - **+** une o concatena

Clases en Java

- Clases
    - **Propias**
    - **Predefinidas**
        - String
        - Math
        - Array
        - Thread
        - [...]
        -

Las clases predefinidas se encuentran en la **API de Java**.

- Paquetes
    - **Java**
        - Java.awt
        - Java.util
            - Java.util.regex
        - Java.io
        - [...]
    - **Javax**
        - Java.activity
        - Java.annotation
        - [...]
    - [...]

Primitivos vs Clases

- **Primitivos**
  - Byte
  - Short
  - Int
  - Double
  - Float
  - Long
  - Char
  - boolean
- **Clases** (Objetos)
  - Math
  - String
  - Image
  - File
  - JButon
  - JFrame
  - JTable
  - Rectangle
  - [...]

Clase String

```
String miNombre;
String mNombre = Cobo
```

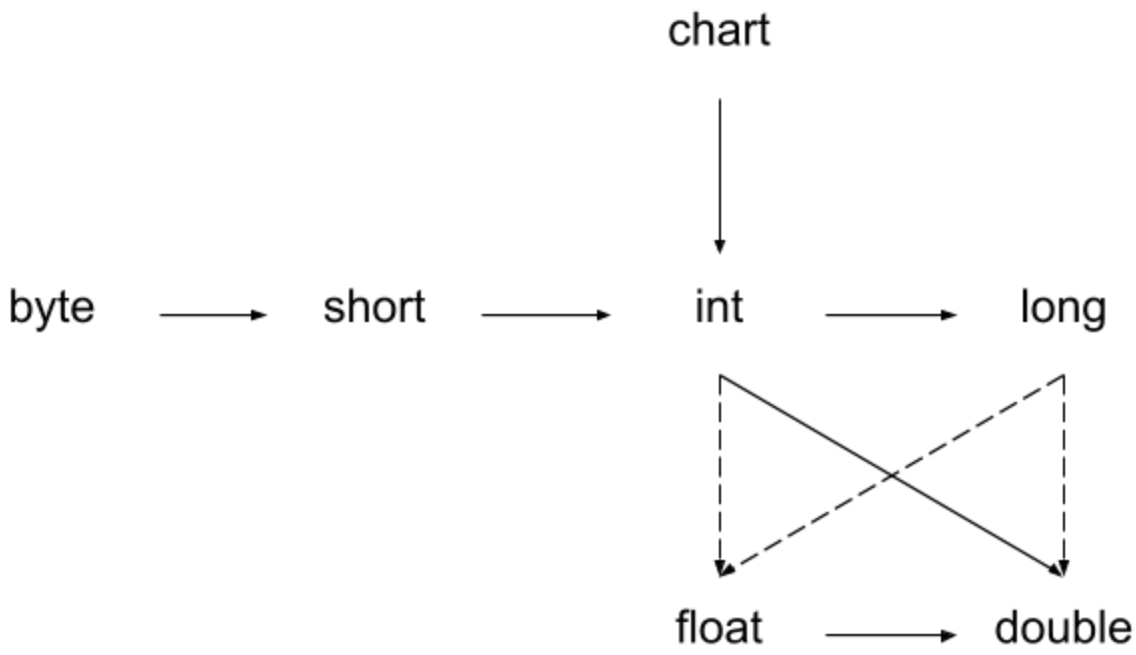
Métodos más usados

```
length(str);
substring(int index1, int index2);
charAt(int index);
equals(String);
equalsIgnoreCase(String);
```

Casting

```
int resultado = (int)Math.round(numero);
```

Hay que tener en cuenta que con tipos primitivos se puede perder precisión.



Paquetes

Se usan para organizar las clases, evitar conflictos de nombres y controlar la visibilidad de las clases.
Al crear paquetes propios:

- El nombre del paquete debe estar en minúsculas
- Por convención, se usa como nombre del paquete un nombre de dominio al revés.

`Com.cursojava.entradadatos`
`Com.cursojava.vistas`

Entrada / Salida

Formas de introducir datos en un programa Java

- A través de una G.U.I.
- Usando la clase Scanner.
  Scanner permite introducir información usando la consola .
  - Scanner
    - `nextLine()` - Texto
    - `nextInt()` - Números enteros
    - `nextDouble()` - Números decimales
- Usando la clase JOptionPane.
  Crea una ventana en la que se puede introducir información.
  - JOptionPane
    - `showInputDialog()` - Se trata de un método estático

Entrada / Salida con Scanner

1.- Se importa la clase Scanner

```
import java.util.Scanner
```

2.- Se crea un objeto de la clase Scanner

```
Scanner sc = new Scanner(System.in);
```

3.- Se crear una variable en la que guardar el dato introducido

```
Int age = sc.nextInt();
String name m= sc.nextLine();
[...]
sc.close();
```

JOptionPane

- Pertenece al paquete **javax.swing**
- El método más utilizado es ***showInputDialog()***
- Es un método **estático**
    - `JOptionPane.showInputDialog();`

Condicionales

IF

SWITCH

```
Switch (valor a evaluar){
      Case valor1:
            Código a ejecutar;
            break;
      Case valor1:
            Código a ejecutar;
            break;
      Case valor1:
            Código a ejecutar;
            break;
      Default:
}
```

- El valor a evaluar solo puede ser un **char, byte, short, String o un enum;**
- En los case no se permiten operadores relacionales. **Solo se puede evaluar igualdad**
- La opción break es opcional

Operador tenario

Bucles

- Indeterminados
    - While
    - Do-While
- Determinados
    - For
    - Foreach

While

Repite las líneas de código en el bucle mientras la condición sea verdadera. Si no se diera una condición falsa en ningún momento, se crearía un bucle infinito

```
while(Condición){
      Código a ejecutar
      [...]
      [...]
      [...]
}
```

Do-while

Similar al bucle while, salvo que **ejecuta el código del interior al menos uan vez**

```
do(Condición){
      Código a ejecutar
      [...]
      [...]
      [...]
}while
```

For

```
for(inicio bucle; fin bucle; contador bucle){
      Código a ejecutar
      [...]
      [...]
      [...]
}
```

Foreach

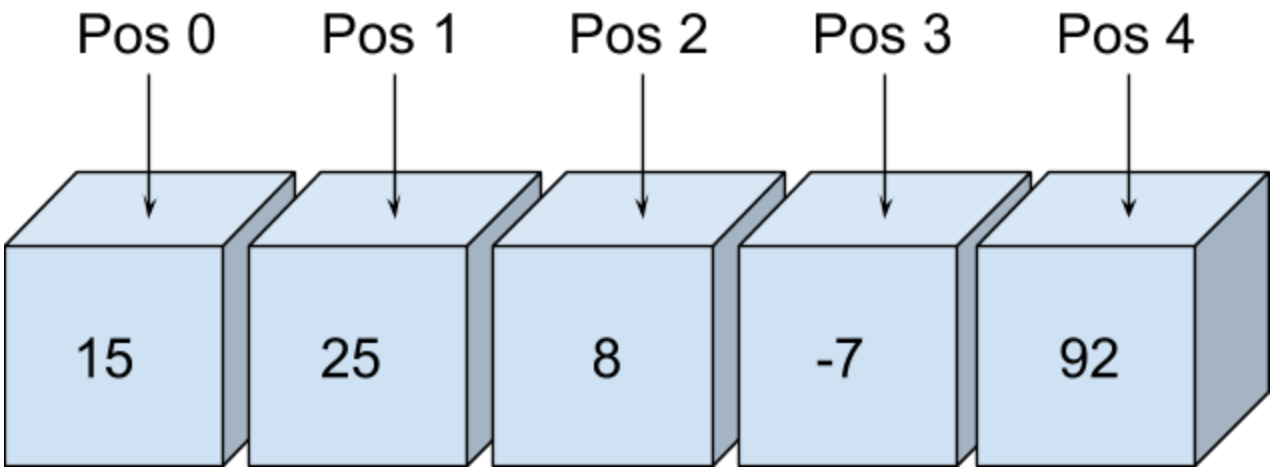
Arrays

Son estructuras de datos que contienen una colección de valores del mismo tipo. Siven para almacenar valores un objetos que normalmente tienen alguna relación entre sí, y que posiblemente interese manipular en grupo

```
int[] miMatriz = new int[10];
```

En Java, los arrays **solo pueden almacenar valores del mismo tipo.**

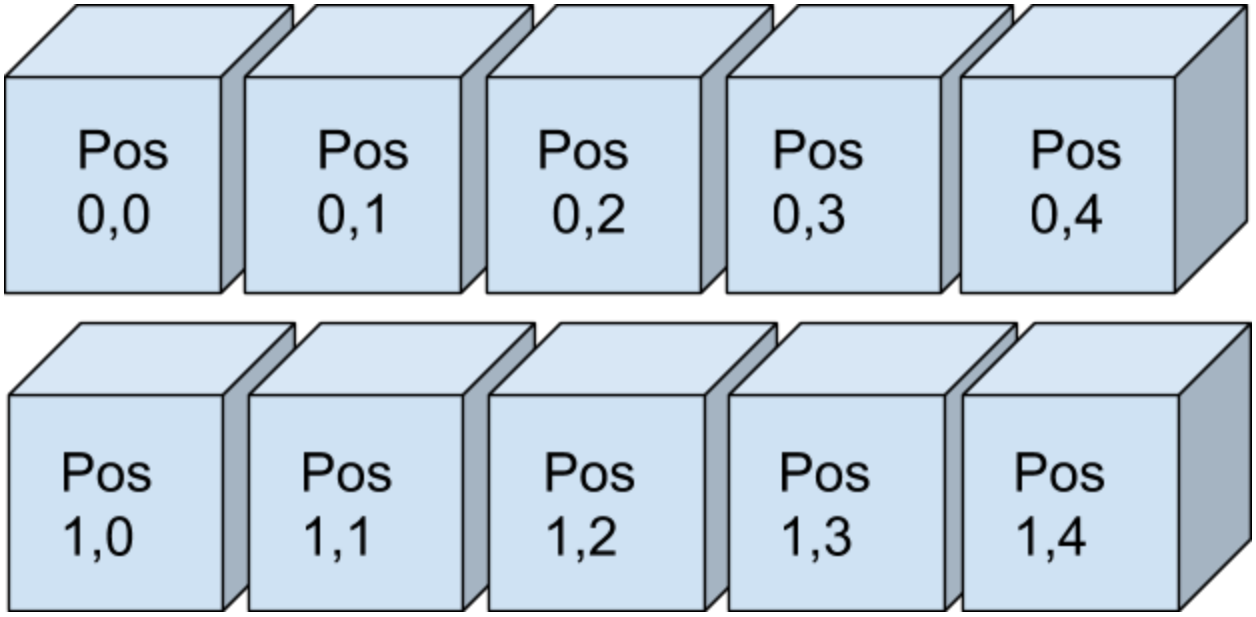
```
miMatriz[0] = 15;
miMatriz[1] = 25;
miMatriz[2] =  8;
miMatriz[3] = -7;
miMatriz[4] = 92;
```



Sabiendo qué datos va a icluir, se puede usar este método simplificado

```
int[] miMatriz = {15, 25, 8, -7, 92};
```

Array bidimensional




```
int[][] miMatriz = new int[5][3]
```

Programación Orientada a Objetos

- Paradigmas de programación
    - Orientada a procedimientos
        - Fortran, cobol, basic, etc
        - Mucho código en aplicaciones complejas
        - Difícil de descrifrar
        - Poco reutilizable
        - Un fallo en una línea posiblemente haga caer el programa
        - Código espagueti
        - Difícil de depurar
    - Orientada a objetos
        - C++, Java, Visual.NET, etc
        - Trasladar naturaleza de los objetos de la vida real al código de programación
        - Los objetos tienen un estado, comportamiento y propiedades
        - Ventajas:
            - Modularización
            - Reutilización. Herencia
            - Tratamiento de excepciones
            - Encapsulamiento

Terminología

Construcción de clases y objetos

Uso del this

Sobrecarga de métodos y constructores

Uso de “Final”

Si estamos seguros de que una propiedad **no debe cambiar jamás de valor** una vez que éste se le haya sido asignado debemos convertir esa propiedad en una **constante**, agregando la palabra ***final***

```
Private final String nombre
```

La palabra final se puede asignar también a una clase o a un método

Uso de Static

Cuando una clase tiene una serie de variables, al crear objetos de esa clase, cada objeto tiene una copia independiente de esas variables. Sin embargo, si convertimos una variable en static

```
private static int id
```

La variable pasa a ser una **variable de clase**, los objetos dejan de tener una copia independiente de esta variable, sino que ésta pertenece a la clase. Para acceder a una variable static, hay que usar el nombre de clase

```
Empleado.id
```

Una constante también podría ser static

```
Public static final int NOMBRECONSTANTE
```

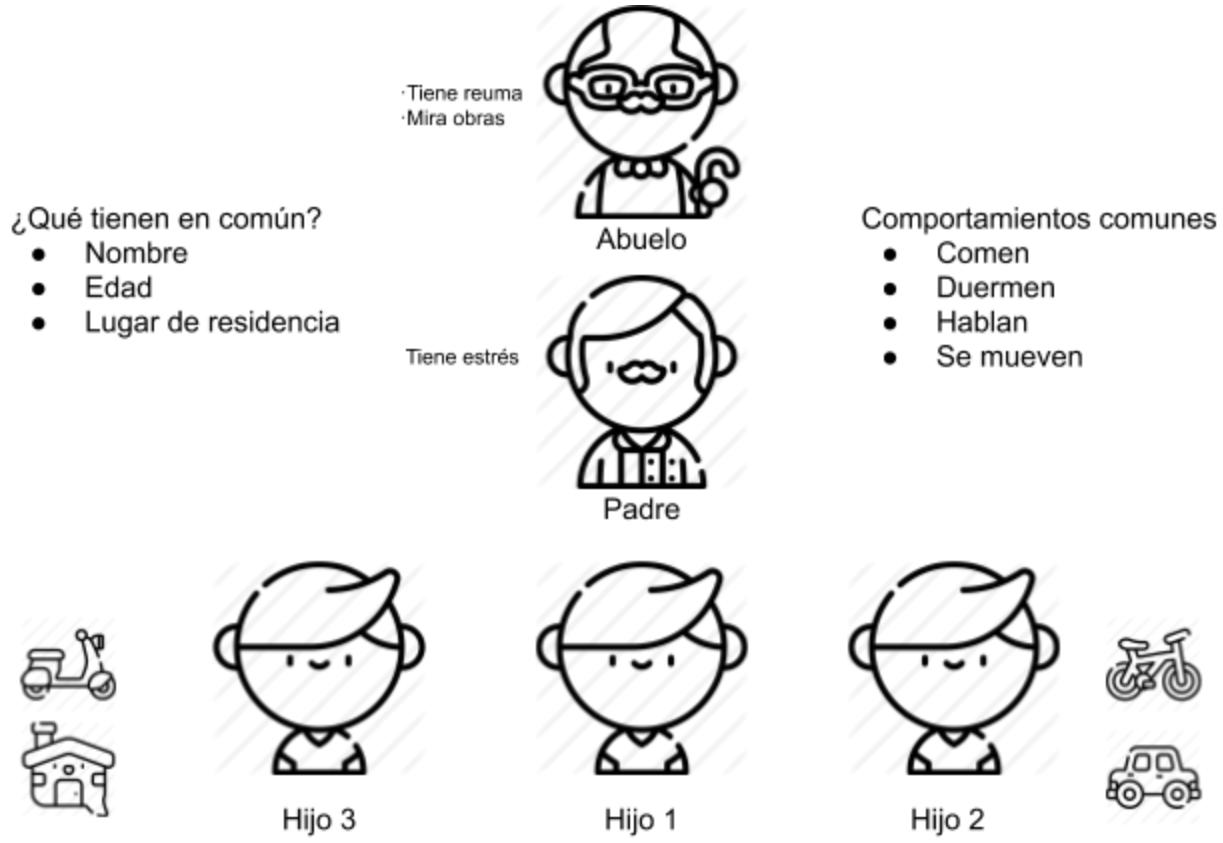
Métodos Static

- No actúan sobre objetos
- No acceden a propiedades, a menos que estas también sean static
- Para llamarlos se usa el nombre de la clase.

```
Math.sqrt();
Math.pow();
```

Herencia

Sirve para reutilizar código cuando creamos objetos que tienen características comunes.



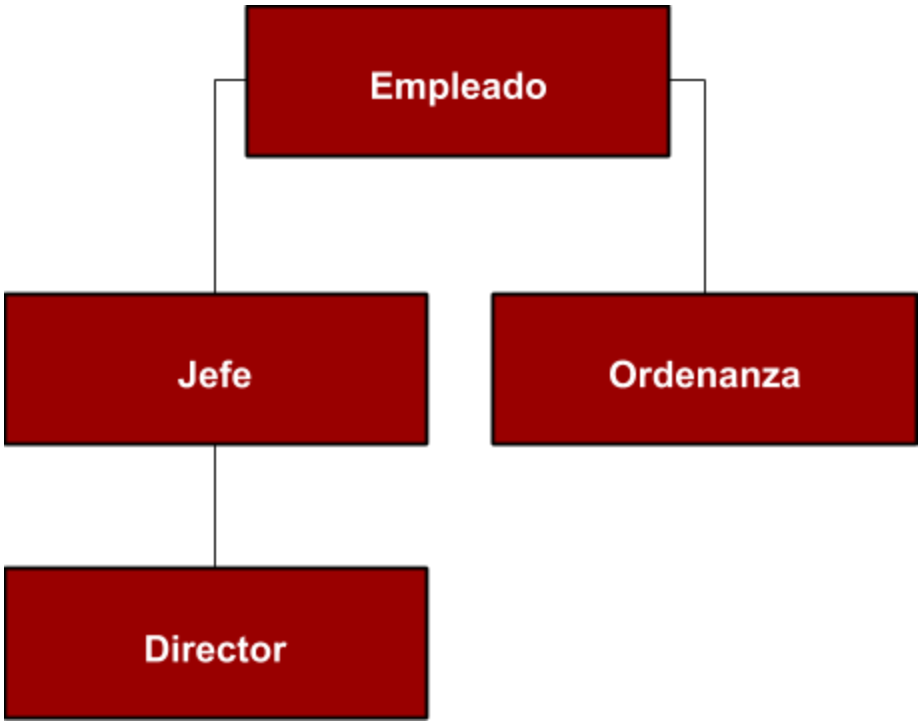
A la clase que se encuentra por arriba se le conoce como **Superclase** y a la que está por debajo **Subclase**

Una manera de identificar bien y decidir qué clase hereda de cuál se puede usar la regla de "***Es un***"?

Ej. Clases Empleado - Jefe

¿***Es un*** jefe un empleado (siempre)? **Sí**
¿***Es un*** empleado un jefe (siempre)? **No**

Por lo tanto en este caso podemos decir la clase jefe heredará de la clase empleado. La clase director por su parte, heredará de Jefe



Sobreescritura de métodos

Clase Object

Modificadores de acceso

- Public: Accesible desde **cualquier lugar**
- Protected : Accesible desde su clase, su paquete y **cualquier subclase dentro o fuera de su paquete**
- Private: Accesible **sólo** desde la propia clase
- Default: Accesible desde **su clase y su paquete**

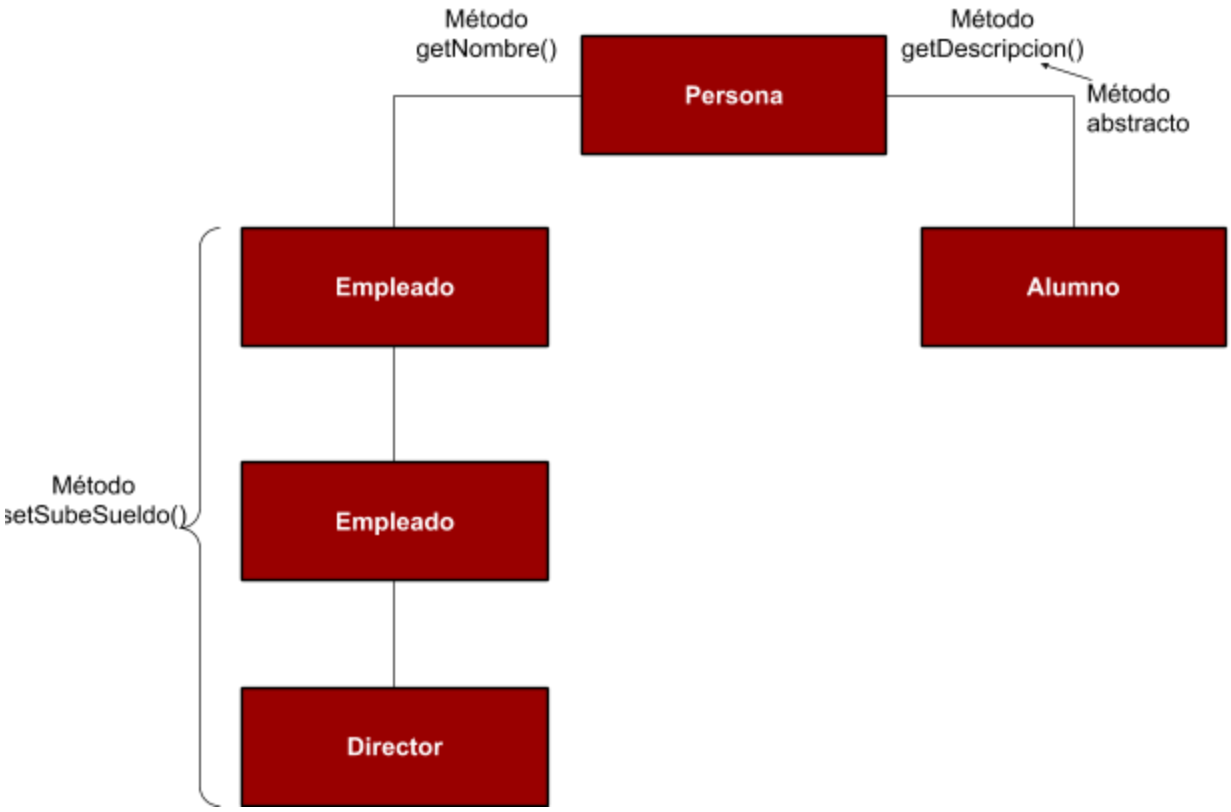
MODIFICADOR	CLASE	PACKAGE	SUBCLASE	TODOS
public	sí	sí	sí	sí
protected	sí	sí	si	no
private	sí	no	no	no
default	sí	sí	no	no

Polimorfismo y enlazado dinámico

Casting de objetos

Clases y métodos FINAL

Clases abstractas



Interfaces

Interfaces predefinidas

La interfaz ActionListener

Clases Internas

Utilidad

- Acceder a los campos privados de una clase desde otra
- Para ocultar una clase de otras pertenecientes al mismo paquete
- Para crear clases internas “anónimas”, **muy útiles para gestionar eventos y callbacks**
- Cuando solo una clase debe acceder a los campos de otra clase

Clases internas locales

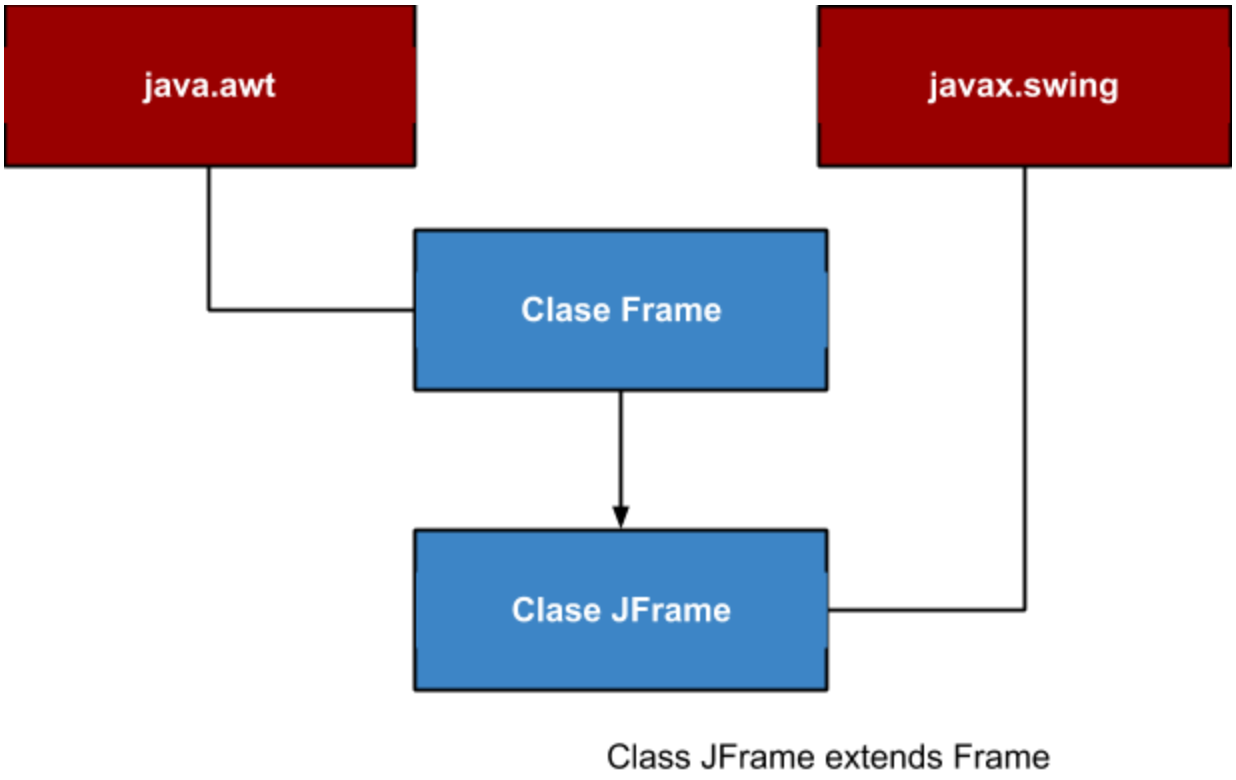
- Una **clase dentro de un método**
- Se usan cuando se **intancia una clase interna una única vez.** De este modo se simplifica el código
- Su ámbito queda restringido al método donde son declaradas
    - Están **muy encapsuladas.** Ni siquiera la clase a la que pertenece puede acceder a ella. **Tan solo puede acceder a ella el método donde están declaradas**
    - **El código resulta más simple**

Clases internas anónimas

- Se encuentran dentro de otra clase y no tienen nombre
- Simplifica el código creando una clase “inline” sobre la marcha. Evita tener que crear una clase adicional para después instanciarla
- Se usan sobre todo para gestionar eventos.

Interfaces gráficas.

Swing vs JavaFx0



Class JFrame extends Frame

```
package com.cursojava.GUIs;

import java.awt.Color;
import java.awt.Dimension;

import javax.swing.*;

public class PrimerJFrame {

	public static void main(String[] args) {
		JFrame miVentana = new JFrame();
		miVentana.setSize(600, 350);
		miVentana.setLocation(600,300);
		miVentana.getContentPane().setBackground(new
Color(51,51,51));



		miVentana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		miVentana.setVisible(true);
	}

}
```

Métodos de JFrame

```
setLocationRelativeTo(null);
setBounds(600, 350, 450, 450);
setResizable(false);
setExtendedState(JFrame.MAXIMIZED_BOTH);
setTitle("Ventana de Prueba");
```

Cambiar icono del JFrame

Se puede cambiar el icono del frame

```
Toolkit tk = Toolkit.getDefaultToolkit();
Image iconImg = tk.getImage(String url);
```

Escribir en el JFrame

Es necesario primero crear un JPane y añadirlo al JFramel. Para escribir en éste, debemos usar el método ***paintComponent(Graphics g)*** que JPanel hereda.
Debemos llamar al método de la clase parent antes de continuar.
Para escribir una String debemos usar el método drawString del objeto Graphics

```
g.drawString(String str, int x, int y); //método sobrecargado
```

PaintComponent se invoca automáticamente al crearse la ventana y al realizarse cualquier operación con el JPanel, como redimensioanrla

Colores y tipografías

Nota: Una vez usado el método paintComponent, la única manera de cambiar el color de background de la ventana es haciendo una llamada a:

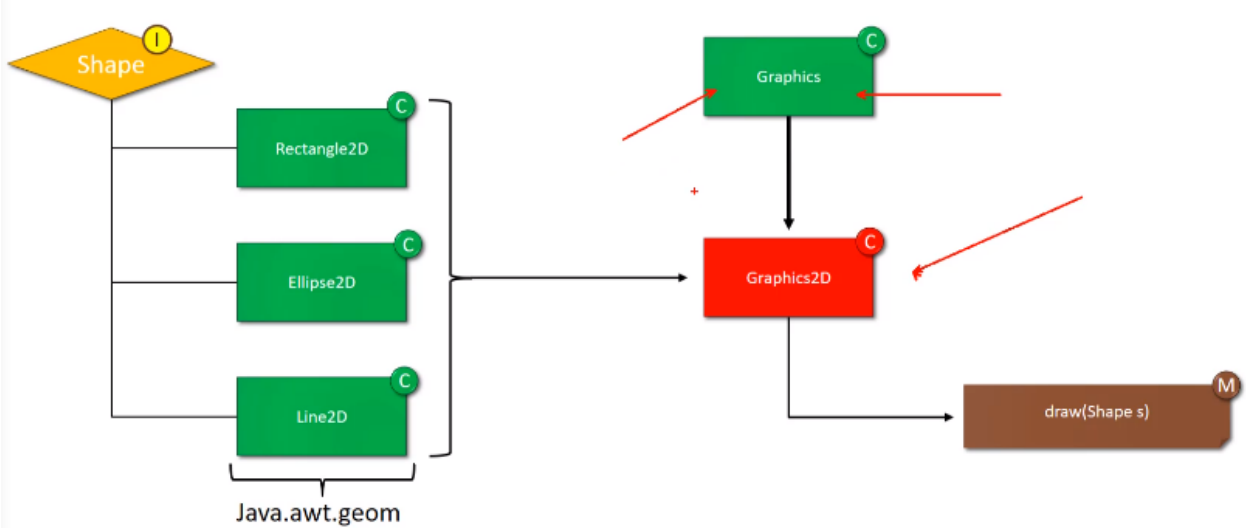
```
super.paintComponent(g);
```

Dibujar en el JPanel

Esto se puede hacer con métodos del objeto Graphics de paintComponent

```
g.drawRect(20, 20, 200, 200) // x, y, width, height
g.drawLine(20, 20, 200, 200) // x, y, x2, y2
```

Java 2D



Shape es una interface implementada en diferentes clases de las que hereda Graphics2D. El (posiblemente) método más importante de Graphics2D es **draw(Shape s)** con el que podremos **crear un objeto de las clases 2D, como Rectangle2D, Ellipse2D o Line2D y pasarlo por parámetro a draw(Shape s) para que lo dibjje**
Para usar esto, haríamos un casting al Objeto Graphics de paintComponent.

```
Graphics2D 2g = (Graphics2D)g;
```

A partir de ahí:

```
Rectangle2D rectangle =
      new Rectangle2D.double(100,100, 200, 150);// x, y, width, height
2g.draw(rectangle);
```

Graphics2D permite cambiar el trazo (esto debe hacerse **antes de dibujar el elemento**):

```
BasicStroke stroke =
      new BasicStroke(4); // pixels - Método sobrecargado

g2.setStroke(stroke);
```

Colores

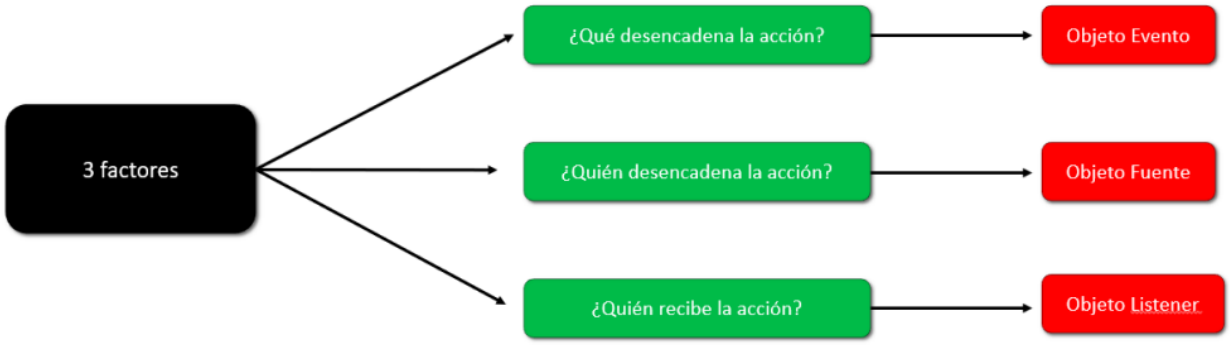
Fuentes

```
String [] misFuentes = GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamilyNames();
```

Imágenes

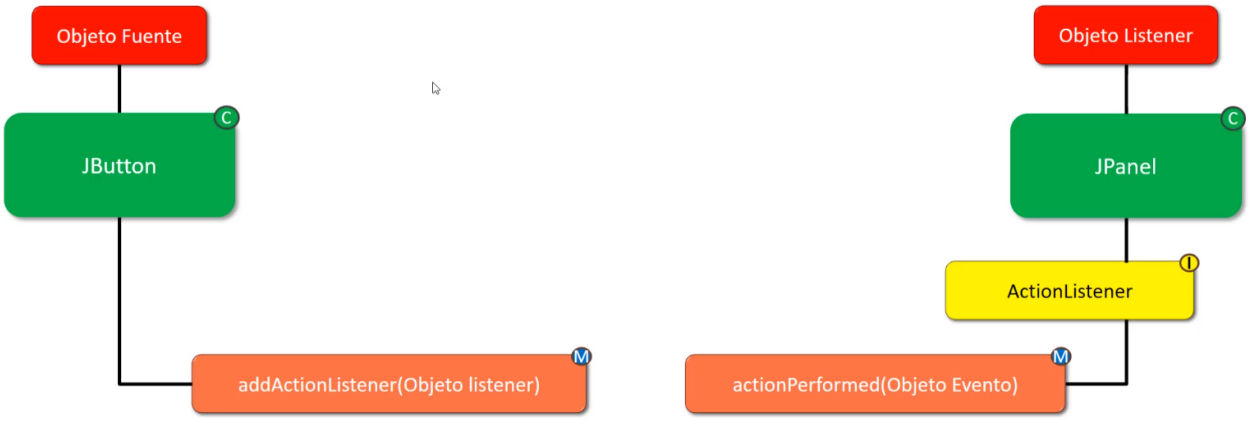
```
File miImagen = new File("src/com/cursojava/GUIs/imagenes/bee.png");
           try {
                Image img = (Image)ImageIO.read(miImagen);
                g.drawImage(img, 0, 0, null);
           } catch (IOException e) {
                e.printStackTrace();
                System.out.println("No se encontró la archivo");
           }
```

Interfaces gráficas & Eventos



- Event Object
  - Eventos de **ratón** usará ***ActionListener***
  - Eventos de **ventana** usará ***WindowListener***

Ambas heredan de ***EventObject*.**



De crear un botón se usará un objeto JButton. Para que este botón desarrolle una acción se le debe poner a la escucha con ***addActionListener***

Hacer click en ese botón hará que otro objeto reciba la acción, este será el objeto listener. Y este componente que va a recibir un evento tiene que implementar la interfaz ***ActionListener***

```
class Panel extends JPanel implements ActionListener{
[...]
}
```

Esto obligará a implementar el método ***actionPerformed*** correspondiente a esta interfaz (en este caso un cambio de color en el JPanel)

```
@Override
```

```
public void actionPerformed(ActionEvent e) {
this.setBackground(new Color(65, 65, 150));
}
```

Eventos en botones

Para añadir un evento a un botón se le añade el método addActionListener().

```
JButton toBlueButton = new JButton("Button");
toBlueButton.addActionListener(this);
```

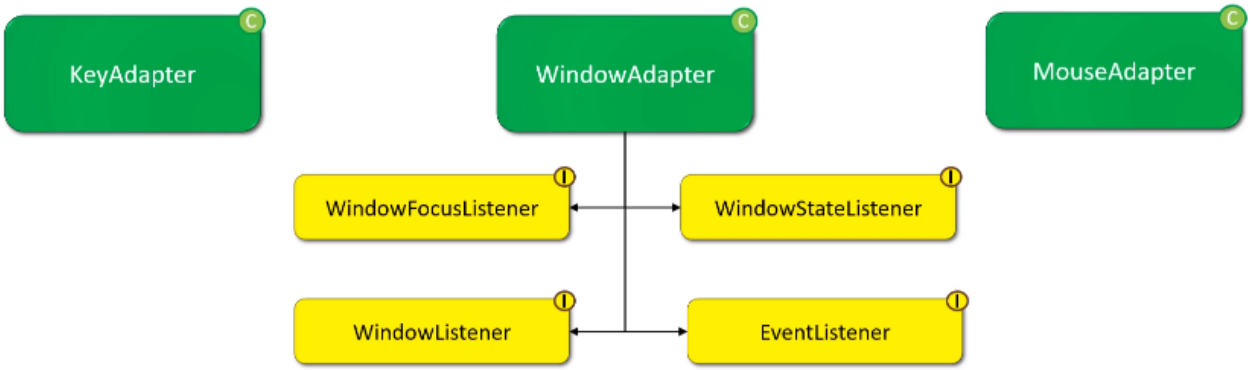
El método ***getSource()*** del objeto **e**, permite identificar la fuente del evento en caso de, por ejemplo, tener varios botones

Obj o = e.getSource();

```
if(o.equals(btn1)){
      [...]
}else if(o.equals(btn2)){
      [...]
}else{
      [...]
}
```

Eventos de ventana

Las ventanas hacen uso de ***WindowListener***, interfaz que obliga a escribir siete métodos
Esto se puede evitar usando una **clase adaptadora**, ayudando a que el código sea más sencillo.



WindowAdapter, incluye todos los métodos incluídos en las interfaces que implementa (***WindowFocusListener, WindowsStateListener, WindowListener, EventListener)***. Al hacer nuestra clase heredar de ***WindowAdapter***, **se podrá reescribir, unicamente los eventos que se necesiten**
Lo mismo ocurrirá con las otras clases adapter

Cambios de estado.

Para cambios de estado de ventana se usa ***WindowStateListener***. Esta interfaz solamente incluye un método. ***windowStateChanged(WindowEvent e)***
El objeto ***WindowEvent*** incluye métodos para identificar el estado anterior y el posterior al cambio. ***getNewState() - getOldState()***.

Eventos de teclado

Se desencadenan al pulsar teclas en el teclado. Se usa la interfaz ***KeyListener***, que obliga a implementar cuatro métodos. Existe una clase adapter ***KeyAdapter*** para facilitar no tener que reescribir todos los métodos

- `keyPressed(KeyEvent e);`
- `keyReleased(KeyEvent e);`
- `keyTyped(KeyEvent e);`

Eventos de ratón

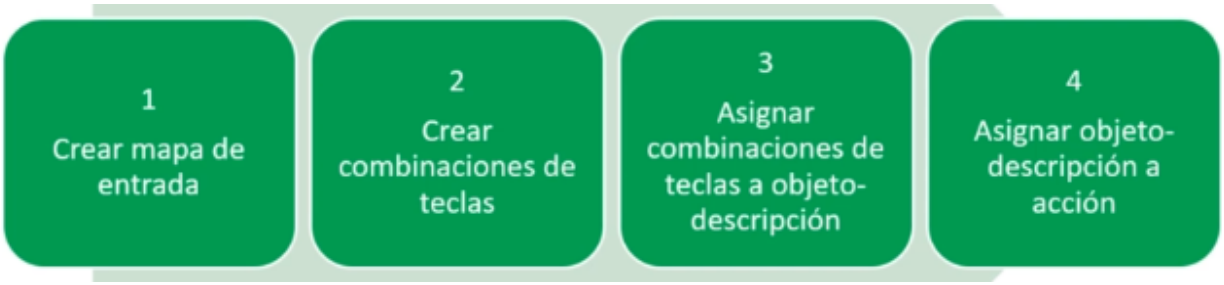
Se desencadenan con acciones del ratón. Usa la interfaz ***MouseListener***, que contiene cinco métodos. También dispone de una clase ***MouseAdapter***

Eventos de foco

Pueden haber efectos de foco en ventanas y componentes

Múltiples fuentes de eventos.

Para trabajar con múltiples fuentes de eventos tenemos la interfaz ***Action***. Ésta implementa seis métodos, además de ***actionPerformed(ActionEvent e)***. ***AbstracAction*** tiene una clase que tiene ya definidos estos métodos, menos ***actionPerformed***, de modo que de usarla, hay que programar este último


1
Crear mapa de entrada
2
Crear combinaciones de teclas
3
Asignar combinaciones de teclas a objeto-descripción
4
Asignar objeto-descripción a acción

Ejemplo en una clase con un botón que cambia el background a azul

```
        public Panel() {

/**ButtoEvents es una clase creada por mí que implementa la interfaz
AbstractActionJ
* Al crear este botón se usa el constructor que acepta como parámetros
el texto del botón y un objeto de la clase Action (en este caso cAzul)
**/
ButtonEvents actionAzul= new ButtonEvents("Azul", new Color(40, 40,
100));
JButton btnAzul = new JButton(actionAzul);
this.add(btnAzul);
/**
*Se asigna la combinación “ctrl + a” al objeto KeyStroke “teclaAzul”
**/
KeyStroke teclaAzul = KeyStroke.getKeyStroke("ctrl A");
/**
* InputMap crea un map de teclas que se usa cuando el componente está
* on focus
**/
InputMap inputMap = getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW);
/**
* Añade el bindeo al mapa. En este caso “ctrl A”
**/
inputMap.put(teclaAzul, "Fondo Azul");
/**
* Se incluye en la variable actionmap el actionmap usado para
determinar qué acción disparar para determinada tecla o combinación de
teclas
**/
ActionMap actionMap= getActionMap();
/**Añade la acción al bindeo **/
actionMap.put("Fondo Azul", actionAzul);
}
```

En resumen:

1. Se crea **shortcut**, es decir atajo de tecaldo
2. Se crea el **inputmap** y se introduce el **shortcut**
3. Se crea el **actionmap** y se le añade el **inputmap**

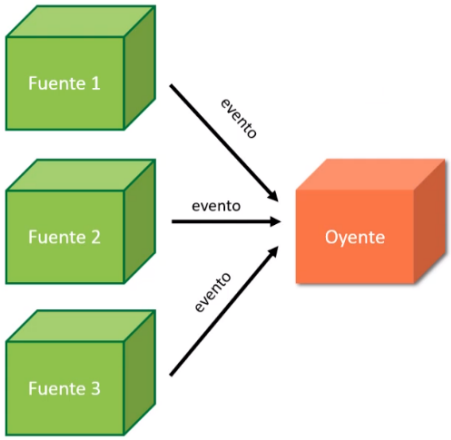
Una forma cómoda podría ser crear los objetos InputMap y ActionMap y entonces crear los shortcuts y guardarlos
Un emplo para tres botones:

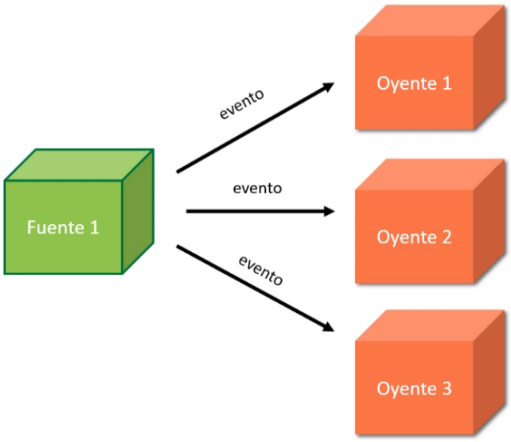
```
ButtonEvents actionRojo = new ButtonEvents("Rojo", new Color(100, 40, 40));
ButtonEvents actionVerde= new ButtonEvents("Verde", new Color(40, 100, 40));
ButtonEvents actionAzul = new ButtonEvents("Azul", new Color(40, 40, 100));

InputMap inputMap = getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW);
ActionMap actionMap = getActionMap();

/** ROJO **/
KeyStroke redShortCut = KeyStroke.getKeyStroke("ctr R");
inputMap.put(redShortCut, "Fondo Rojo");
actionMap.put("Fondo Rojo", actionRojo);

/** AZUL **/
KeyStroke blueShortCut = KeyStroke.getKeyStroke("ctrl A");
inputMap.put(blueShortCut, "Fondo Azul");
actionMap.put("Fondo Azul", actionAzul);


/** VERDE **/
KeyStroke shortCutVerde = KeyStroke.getKeyStroke("ctr V");
inputMap.put(shortCutVerde, "Fondo Verde");
actionMap.put("Fondo Verde", actionVerde);
```

Se puede simplificar de un poco creando directamente el objeto key Stroke

```
InputMap inputMap = getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW);
ActionMap actionMap = getActionMap();

/** ROJO **/
inputMap.put(KeyStroke.getKeyStroke("ctrl R"), "Fondo Rojo");
actionMap.put("Fondo Rojo", actionRojo);

/** AZUL **/
inputMap.put(KeyStroke.getKeyStroke("ctrl A"), "Fondo Azul");
inputMap.put(KeyStroke.getKeyStroke("ctrl D"), "Fondo Azul");
actionMap.put("Fondo Azul", actionAzul);

/** VERDE **/
inputMap.put(KeyStroke.getKeyStroke("ctrl V"), "Fondo Verde");
actionMap.put("Fondo Verde", actionVerde);
```

Múltiples oyentes de eventos

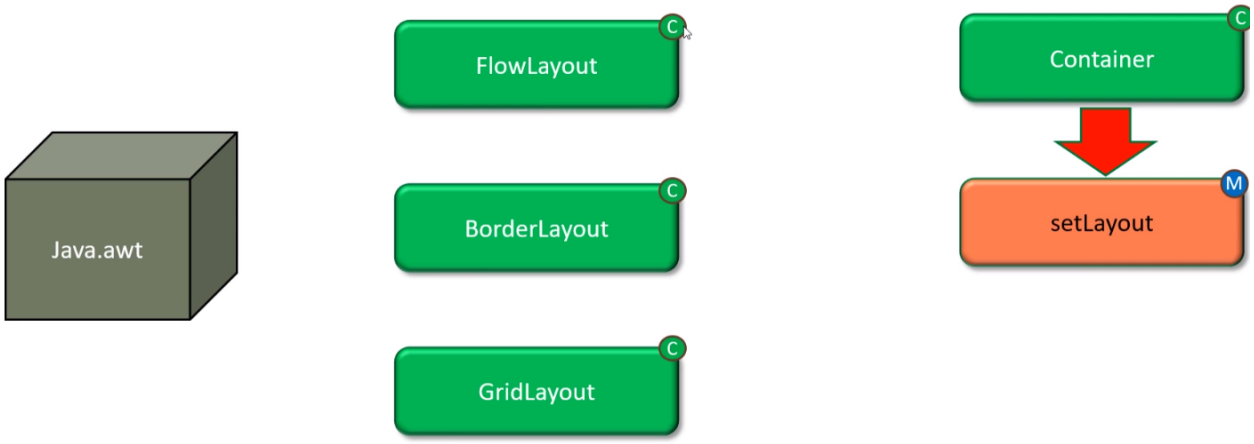




Layouts

Existen tres tipos de layouts para java.

- **FlowLayout -** (por defecto)
- **BorderLayout**. **-** Divide el panel en 5 secciones (norte, sur, este, oeste y centro). Lo que se ubique en estas zonas, ocupará todo el espacio existente en esa zona
- **GridLayout**. **-** Crea un gid con las columnas y filas que indiquemos



Componentes Swing

JRadioButton

Ejemplo con 3 botones en un grupo.

```
JRadioButton rButton1 = new JRadioButton();
JRadioButton rButton2 = new JRadioButton();
JRadioButton rButton3 = new JRadioButton();

ButtonGroup buttonGroup = new ButtonGroup();
buttonGroup.add(rButton1);
buttonGroup.add(rButton2);
buttonGroup.add(rButton3);

add(rButton1);
add(rButton2);
add(rButton3);
```

JTextField

```
JTextField txt1 = new JTextField(10);
```

Eventos en JTextField

JText fiel hereda de JTextComponent, esta clase tiene un método getDocument, por lo que JTextField lo tiene también por herencia. Se considera que dentro de cada textfield hay un documento. Para esto, java tienen la interfaz Document, que tiene un método ***addDocumentListener()*** que pone en escucha el documento, que con cualquier cambio disparará un ***documentEvent()***. Esto se captura con una clase que implemente la interfaz ***DocumentListener***.

JTextArea

Métodos más usados:

- `getText();`
- `setLineWrap(boolean)`
- `getLineWrap()`

Incluir barras de scroll en el JTextArea

Para esto, el JTextArea tiene que ser agregado en un **JScrollPane**

JTextPane

JTextPane parece ofrecer la misma función de un JTextArea y además opciones para editar parte del texto del área como si de un editor de texto se tratase

JCheckBox

Métodos más usados:

- `isSelected()`
- `setSelected()`

JSlider

Métodos más usados:

- `setPaintTicks(boolean);`
- `setMayorTickSpacing(int);`
- `setMinorTickSpacing(int)`
- `setPaintLabels(boolean)`

Los eventos de este componente se capturan con ***ChangeListener***

JSpinner

Los eventos de este componente se capturan con ***ChangeListener***

JSpinnerModel

JMenuBar - JMenu - JMenuItem

JToolBar

Método más usado
`add(Action acción);`

JPopupMenu

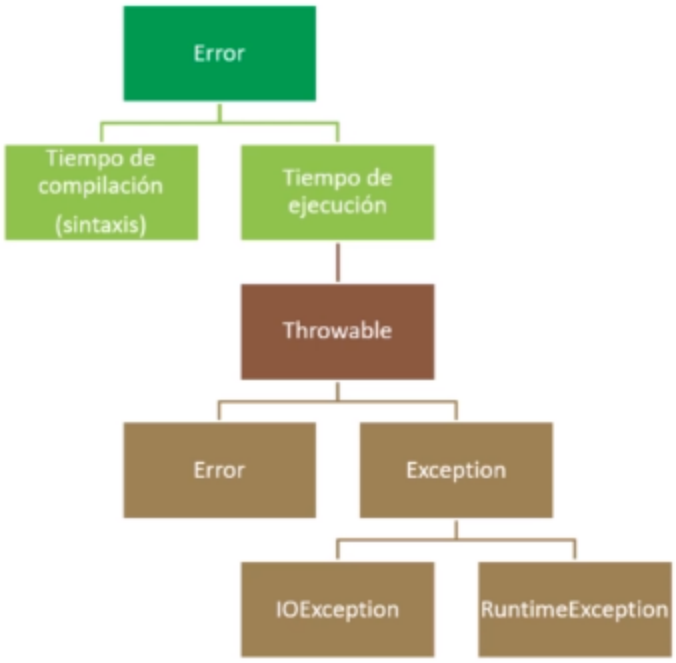
Disposiciones avanzadas

BoxLayout

```
Box userBox = Box.createHorizontalBox();
userBox.add(lblUsuario);
userBox.add(Box.createHorizontalStrut(10));
userBox.add(txtUsuario);

[...]

Box buttonBox = Box.createHorizontalBox();
JButton btn1 = new JButton("Ok");
JButton btn2 = new JButton("Cancelar");

buttonBox.add(btn1);
buttonBox.add(Box.createGlue());
buttonBox.add(btn2);


Box verticalBox = Box.createVerticalBox();

verticalBox.add(userBox);
verticalBox.add(passBox);
verticalBox.add(buttonBox);


this.add(verticalBox);
```

Spring

Excepciones



- **Error**:
  Es un error de hardware. Pasa si el ordenador se queda sin espacio en el disco duro o la memoria está llena o corrupta.
- **Exception**
  Estos errores si son controlables.
    - **IOException**
        - (Excepciones comprobadas)
        - No son culpa del programador (no siempre)
        - Try / catch
    - **RuntimeException**
        - (Excepciones no comprobadas)
        - Es responsabilidad del programador
        - Throws

Cuando se use un método que puede lanzar excepción, como las de **IOException (comprobadas)**, esto debe ser controlado con un bloque **try / catch.**
Si la excepción pertenece a las del grupo **RuntimeException (con comprobadas)** no obliga a usar un bloque try/catch. En este caso debemos usar la claúsula **throws**

Excepciones propias

Deben tener 2 constructores. Uno vacío, y uno con un atributo mensaje. Si capturamos esta excepción, podremos acceder a este con el método ***e.getMessage();***

Ejemplo:

```
class MailTooShortException extends RuntimeException{

      public MailTooShortException() { };

      public MailTooShortException(String message) {
            super(message);
      };

}
```

Método que lanza esta excepción

```
     private static void examinaMail(String email) throws
MailTooShortException  {

          int arroba = 0;
          boolean punto = false;

          if(email.length() <= 3) {

               MailTooShortException exception = new
MailTooShortException(
                         "\nEmail has 3 or less characters");
               throw exception;
          }

          for(int i = 0; i < email.length(); i ++) {
               if(email.charAt(i) == '@') {
                    arroba ++;
               }

               if(email.charAt(i) == '.') {
                    punto = true;
               }
          }

          if(arroba == 1 && punto == true) {
               System.out.println("Email válido");
          }else {
               System.out.println("Email no válido");
          }
     }
```

Capturar varias excepciones

Esto se puede realizar facilmente rodeando el método que produce la excepción por un bloque try / catch, con tantos catches como excepciones pueda lanzar. Se puede capturar en cualquier caso de una forma más general, por ejemplo simplemente capturando una ***Exception***
Es importante el orden, es decir, de poner en el primer catch una Exception, el programa no continuaría buscando, así que habría que ir de más a menos específico

```
public class VariasExcepciones {

	public static void main(String[] args) {

		try {
			division();

		} catch (ArithmeticException e) {
			System.out.println("No se puede realizar división por
0");
			JOptionPane.showMessageDialog(null, "No se puede
realizar división por 0");
		} catch (NumberFormatException e) {
			System.out.println("Debes introducir un número");
			JOptionPane.showMessageDialog(null, "Debes introducir
un número");
		}
	}

	static void division() throws ArithmeticException,
NumberFormatException {
		int num1 =
Integer.parseInt(JOptionPane.showInputDialog("Introduce el
dividento"));
		int num2 =
Integer.parseInt(JOptionPane.showInputDialog("Introduce el divisor"));
		System.out.println("El resultado es " + num1 / num2);
	}

}
```