Aplicaciones Java

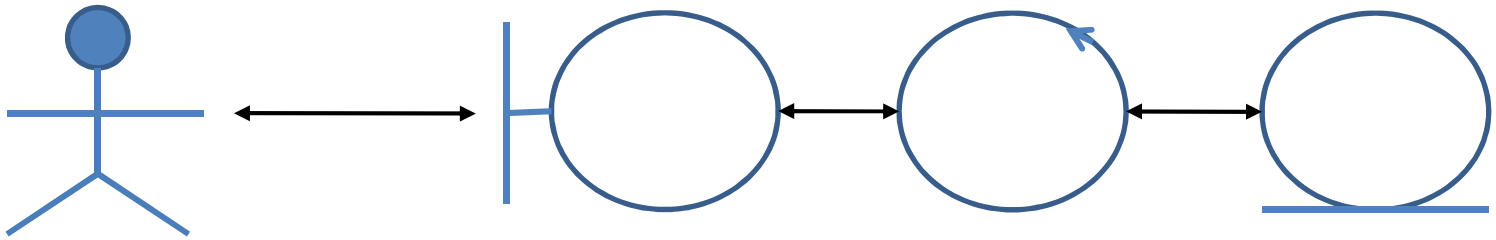
Juan Manuel Fernández Peña

Curso 2011, 2013

Aplicación

- Una aplicación es una colección de clases y recursos adicionales, con un punto de inicio.
- El punto de inicio está dado por el **método main** de alguna de las clases.
- Algunas clases pueden definirse en el último momento (ver ejemplo)

Organización



Usuarios
(humanos, máquinas, otros sistemas)

Clases fronterizas
(Interfaz de usuario)

Clases de control
Coordinan actividades de otras

Entidades
Clases que almacenan datos

Las Clases de Control muy simples se absorben en Clases Fronterizas o en Entidades

Clases fronterizas

- Orientadas a texto:
  - la interfaz lee datos y ordenes de consola o de archivo, las interpretan del texto a números u otros elementos.
  - Transmite las ordenes a clases de control o entidades
  - Transforma las respuestas a texto que aparece en consola o en archivos

Clases fronterizas

- Gráficas:
  - Utilizan ventanas con elementos gráficos de entrada (campos de texto, botones, menús, etc.)
  - Transmiten las ordenes a otras clases
  - Muestran los resultados en elementos gráficos (etiquetas, campos de texto, etc.)

Clases Entidad

- Su propósito principal es la representación de datos del dominio del problema, conservarlos e informarlos cuando se los piden.
- Pueden estar asociados a elementos persistentes (objetos serializados, tablas en una base de datos).
- En general son pasivos.

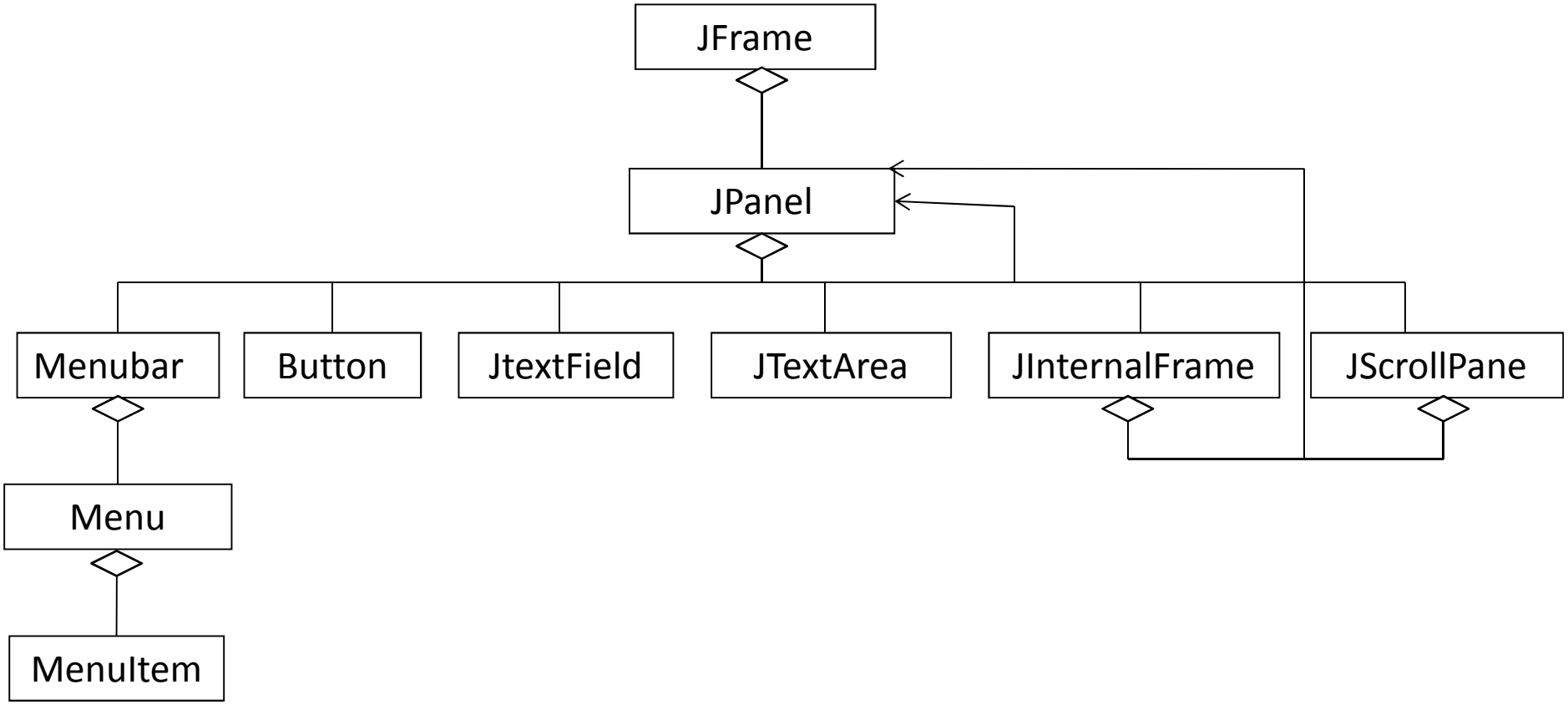
Clases de Control

- Su papel principal es ofrecer funcionalidad a través de algoritmos y conexión a entidades.
- Si las funciones son simples o conexión sencilla, se absorben en eventos de interfaces.

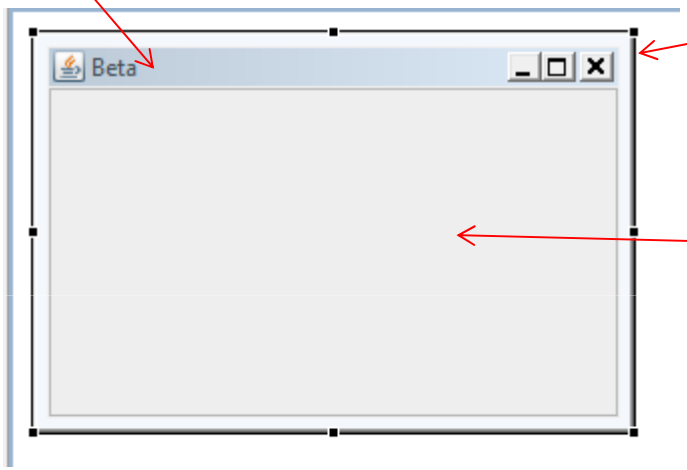
Aplicaciones gráficas en Java

- Se usa una clase que contiene una ventana principal (Application Window, JFrame, JApplet, etc.)
- La ventana es un marco vacío con mecanismos básicos.
- La ventana lleva un contenedor general sobre el que se agregan otros elementos, incluyendo otros contenedores

Aplicaciones gráficas en Java



Título de la ventana



Ventana con funciones de minimizar, maximizar, cerrar

Contenedor: JPanel

JFrame

```
import javax.swing.SwingUtilities;
import javax.swing.JPanel;
import javax.swing.JFrame;

public class Beta extends JFrame {
private JPanel jContentPane = null;
public static void main(String[] args) {
// TODO Auto-generated method stub
  SwingUtilities.invokeLater(new Runnable() {
  public void run() {
    Beta thisClass = new Beta();
    thisClass.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    thisClass.setVisible(true);
    }
   });
  }
public Beta() {
super();
initialize();
}
```

Contenedor

Clase anónima interna

Constructor (note initialize)

```
private void initialize() {
this.setSize(300, 200);
this.setContentPane(getJContentPane());
this.setTitle("Beta");
}

private JPanel getJContentPane() {
if (jContentPane == null) {
jContentPane = new JPanel();
jContentPane.setLayout(new BorderLayout()); //cambiar a null
}
return jContentPane;
}

}
```

Creación del contenedor

Application Window

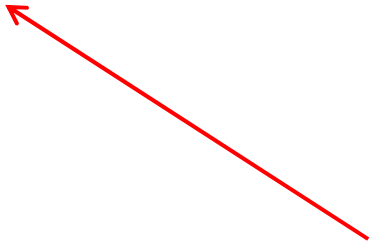
```
import java.awt.EventQueue;
import javax.swing.JFrame;
public class Aristarco {
private JFrame frame;

public static void main(String[] args) {
  EventQueue.invokeLater(new Runnable() {
  public void run() {
  try {
    Aristarco window = new Aristarco();
    window.frame.setVisible(true);
  } catch (Exception e) {
  e.printStackTrace();
  }
  }
  });
}
public Aristarco() {
  initialize();
}
```

La clase contiene el JFrame

Clase anónima interna

```
private void initialize() {
   frame = new JFrame();
   frame.setBounds(100, 100, 450, 300);
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   frame.getContentPane().setLayout(null);
}
}
```



Contenedor; aparece después

Para el uso de Window Builder

```
import java.awt.BorderLayout;
public class IUBanco extends JFrame {
private JPanel contentPane;
public static void main(String[] args) {
  EventQueue.invokeLater(new Runnable() {
      public void run() {
        try {
                IUBanco frame = new IUBanco();
               frame.setVisible(true);
            } catch (Exception e) {  e.printStackTrace();  }
       }       });
}
public IUBanco() {
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setBounds(100, 100, 450, 300);
  contentPane = new JPanel();
  contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
  contentPane.setLayout(new BorderLayout(0, 0)); //cambiar a null
  setContentPane(contentPane);
}  }
```

Clase anónima interna

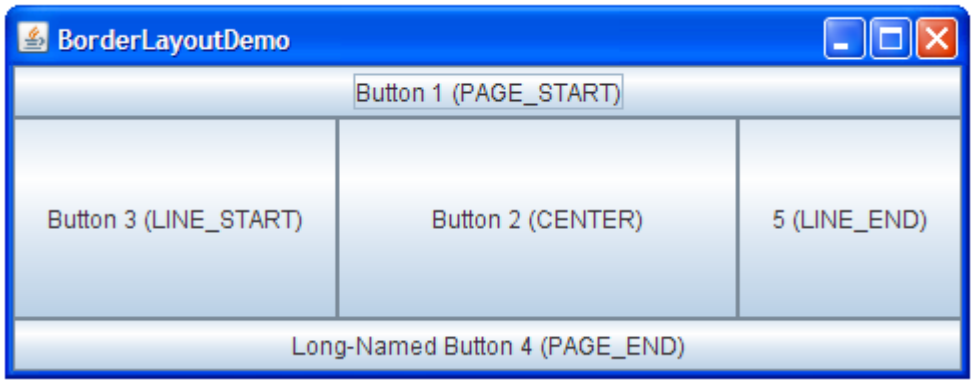
Note que la inicialización va dentro del constructor

Layout

http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

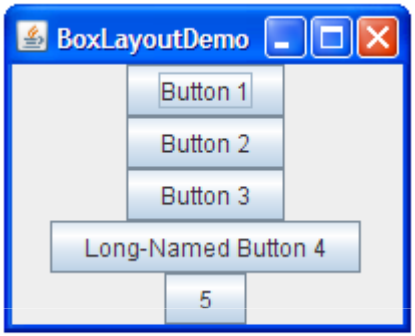
- BorderLayout
- BoxLayout
- CardLayout
- FlowLayout
- GridBagLayout
- GridLayout
- GroupLayout
- SpringLayout

Border Layout



Divide la ventana en cinco áreas y las llena secuencialmente cada una de ellas. Las áreas son Norte, Sur, Centro, Este y Oeste. Las piezas que se incluyen tienden a llenar el espacio, por lo cual aparecen deformadas (ver figura). Es el formato por omisión.

Box Layout

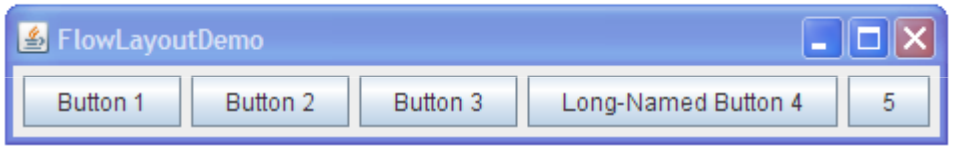


Coloca en una fila o columna

Card Layout

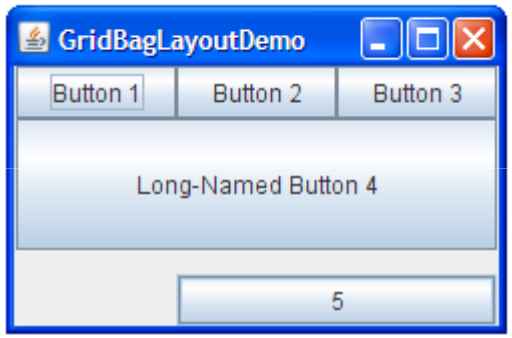
Reserva un área para componentes variables; en un Panel se guardan y cambiando de Panel se cambia de elementos

Flow Layout



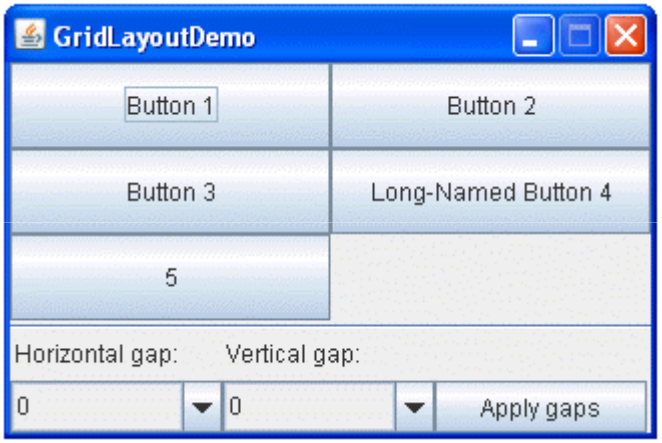
Una sola fila hasta llenar y luego sigue abajo

Grid Bag Layout



Sigue una tabla, pero un elemento puede usar varias celdas. Los tamaños pueden variar

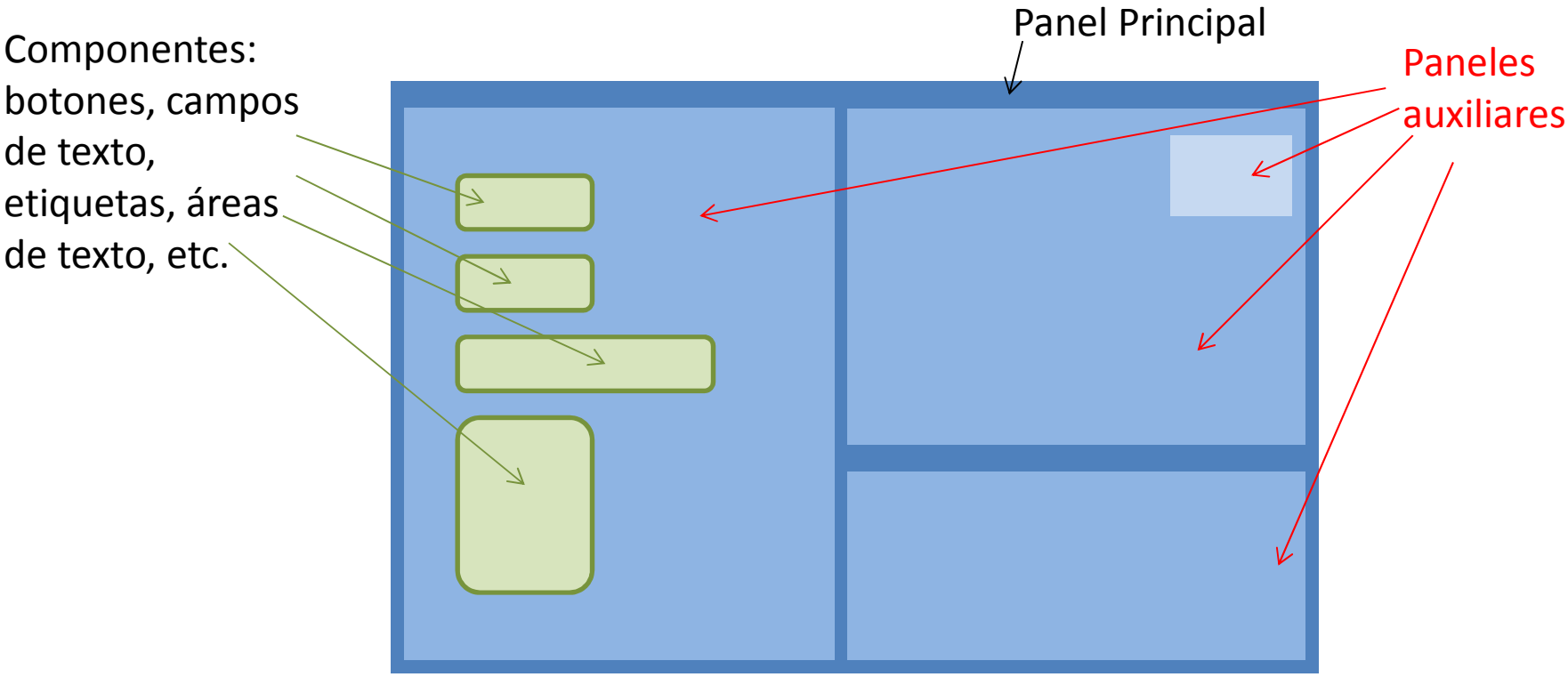
Grid Layout



Los acomoda en una tabla con mismo tamaño

JPanel

- Esta clase es como un contenedor sin bordes. Además del contenedor principal de una ventana, se pueden agregar paneles para hacer subdivisiones o agrupar componentes



JTextArea

- Se utiliza cuando hay más de una línea de texto. Se puede definir número de líneas y columnas al crearla.
- Usando setText trabaja como JTextField
- Usando append agrega al final del texto
- Usando “\n” se puede saltar de línea

Recomendaciones

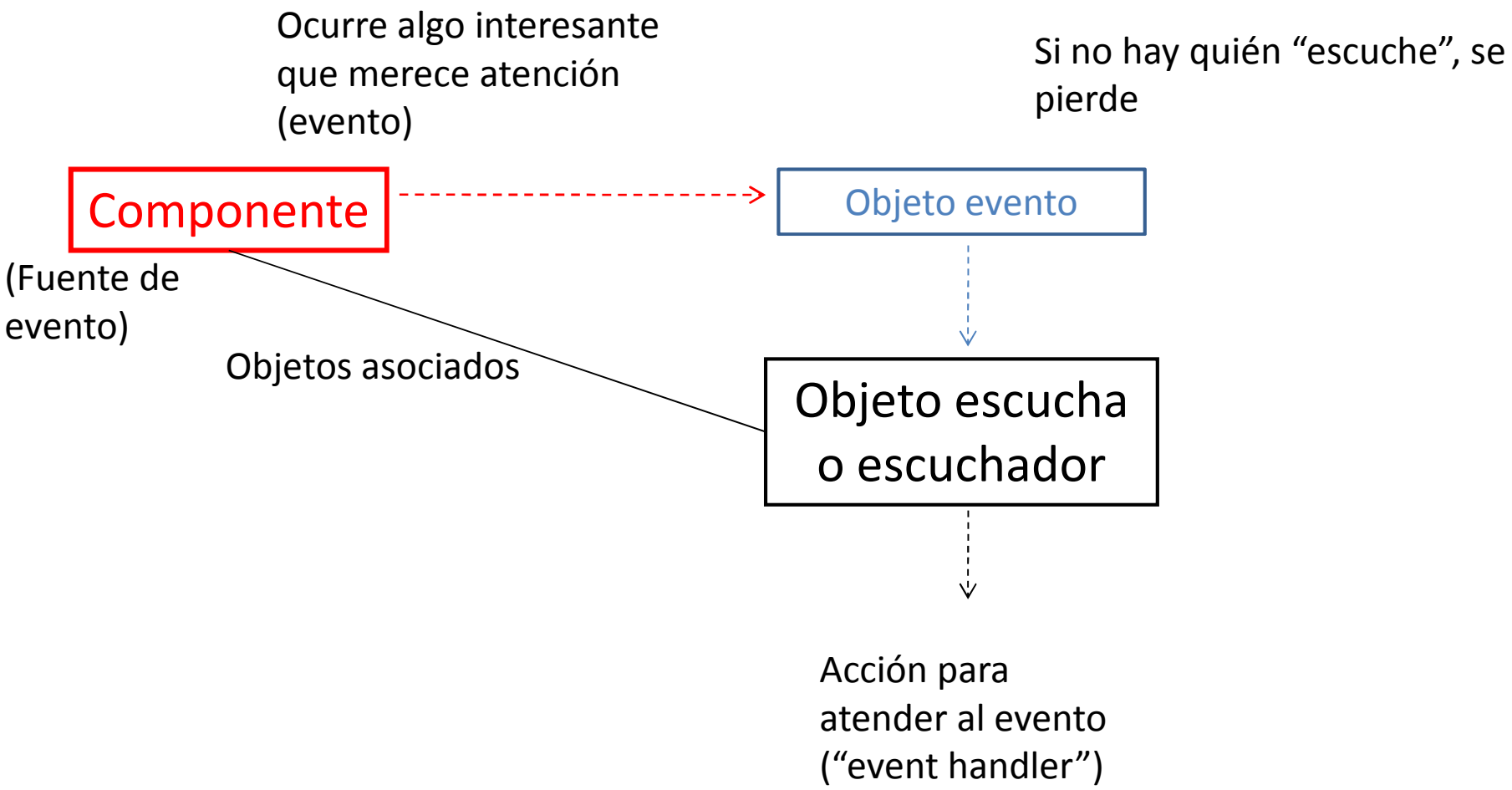
- Construya las interfaces de manera iterativa, agregando elementos poco a poco y probándolos, sobre todo en caso de eventos.
- El editor WindowBuilder oculta los elementos gráficos cuando uno de ellos tiene un error de sintaxis. Para poder verlos, marque las líneas con error como comentarios y corrija una a una.
- Una clase interfaz necesita tener agregada una o más clases entidad o de control, para poder trabajar, de manera similar a los casos de prueba o la interfaz de texto.
- Un error común es no declarar ningún objeto o no crearlo (new), lo que se traduce en un NullPointerException.

Eventos y su manejo

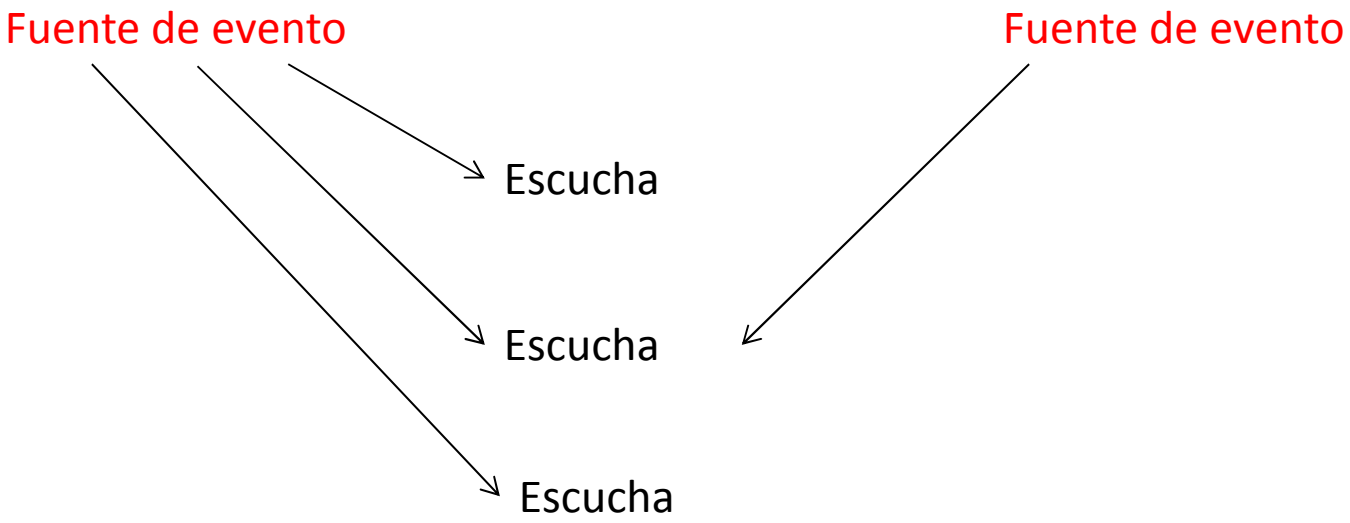
Eventos

- Muchas aplicaciones gráficas operan como sigue:
  - Se inicia la aplicación, sucediendo varias cosas transitorias (imágenes, avisos, llamados a esperar.
  - Aparece una ventana (principal o de identificación), que permanece inactiva …
  - hasta que hacemos algo que la saca de ese estado: oprimir una tecla, seleccionar un menú, dar clic a un botón, …

Evento y su contexto



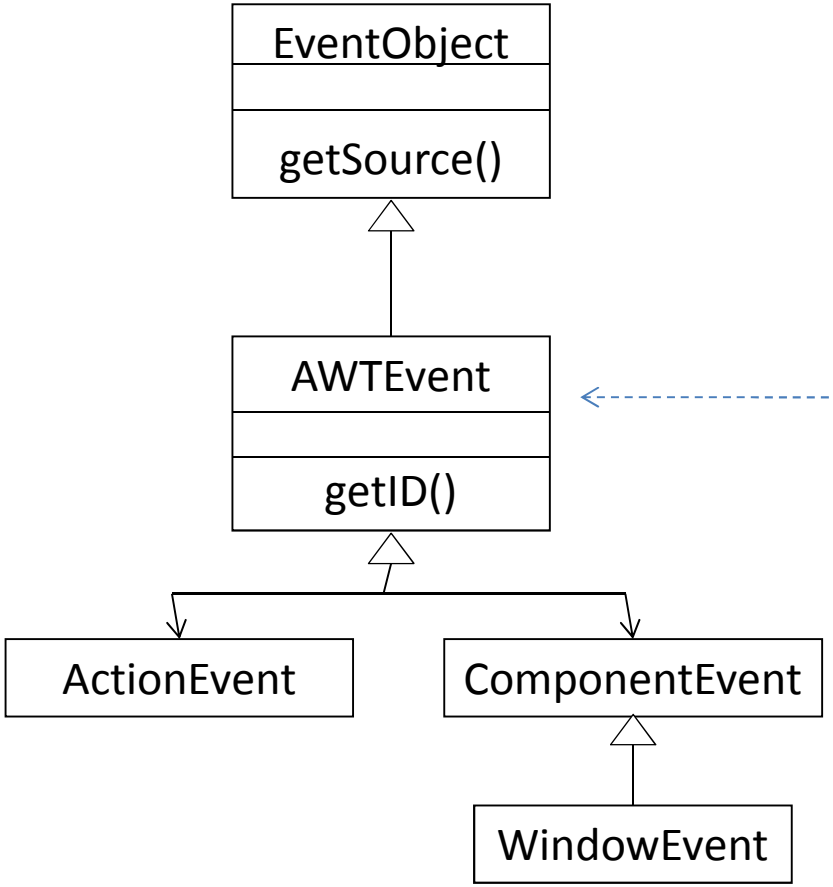
Posibilidades de eventos



Un evento a muchos oyentes,
Un oyente a varios eventos

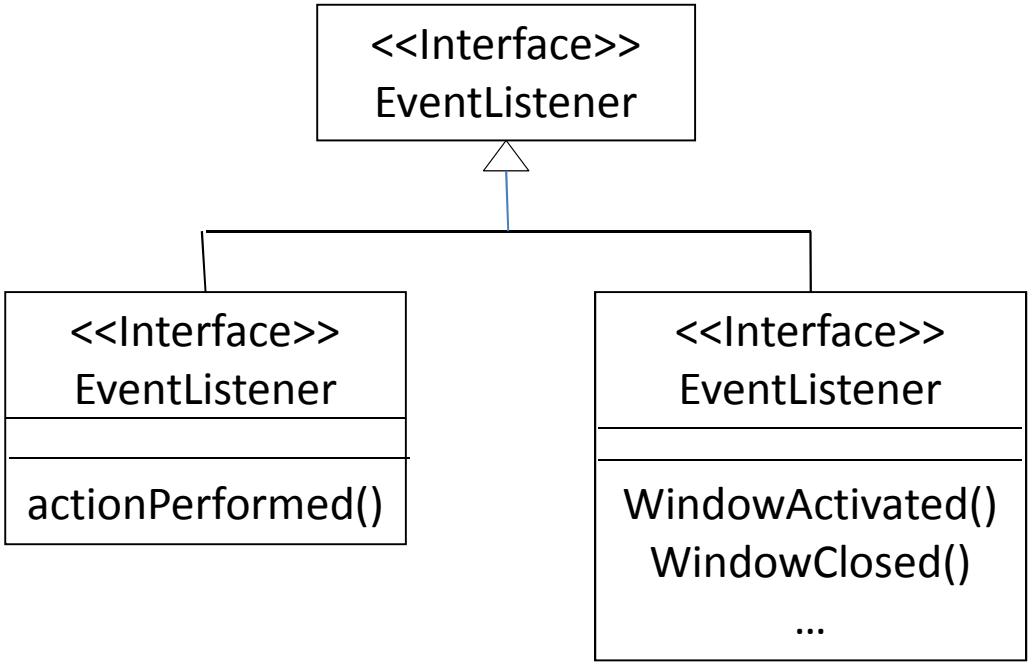
La conexión es una forma de "*callback*" donde se registran para que les avisen cuando algo pase.

Objeto evento



ID es el tipo de evento. Es un número. Si se inventan eventos adicionales, debe cuidarse de asignar números mayores a los que están reservados.

Objeto Escucha



Escucha como clase propia

- **import java.awt.event.ActionEvent;**
- **import java.awt.event.ActionListener;**
- **public class Escucha implements ActionListener {**
- @Override
- **public void actionPerformed(ActionEvent arg0) {**
- // **TODO Auto-generated method stub**
- System.*out.println("El escucha recibió un evento del componente "+arg0.getSource());*
- }
- /* Note que esta forma le da autonomía, pues es un objeto de una clase
- * bien definida; eso es bueno para tareas complejas
- * permite manejar eventos de varios componentes en un solo lugar
- * Pero ... no tiene acceso a los atributos y elementos gráficos de ningún componente
- */
- }

Componente.addActionListener(new Escucha());

Escucha como clase interna

- La clase se declara dentro del archivo de la principal; de este modo tiene acceso a sus atributos, pero pierde visibilidad

Escucha como clase anónima

- Como lo único necesario es el método, cuando es una acción más o menos sencilla se crea dentro de la misma llamada:

- jButton7.addActionListener(**new java.awt.event.ActionListener() {**
- **public void actionPerformed(java.awt.event.ActionEvent e) {**
- System.*out.println("actionPerformed()"); // **TODO Auto-generated***
- **if (cual) jButton7.setText(*equis);***
- **else jButton7.setText(*ooo);***
- cual = !cual;
- }
- });

Creación de eventos en botón

- Con botón derecho sobre botón
- Elegir “Add event handler”
- Elegir “Action” y luego “action performed”
- Crea clase anónima interna; debe llenarse la acción

Ejemplo: interfaz para Figuras

```
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
public class Euclides {
private JFrame frmEuclidesJugandoA;
private JTextField textField;
private JTextField textField_1;
private JTextField textField_2;
private JTextField textField_3;
private LasFiguras misFigs;
```

```
public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
public void run() {
try {
Euclides window = new Euclides();
window.frmEuclidesJugandoA.setVisible(true);
} catch (Exception e) {
e.printStackTrace();
}
}
});
}

public Euclides() {
misFigs = new LasFiguras();
initialize();

}
```

```
private void initialize() {
frmEuclidesJugandoA = new JFrame();
frmEuclidesJugandoA.setTitle("Euclides jugando a la geometr\u00EDa");
frmEuclidesJugandoA.setBounds(100, 100, 450, 300);
frmEuclidesJugandoA.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frmEuclidesJugandoA.getContentPane().setLayout(null);
JLabel lblNewLabel = new JLabel("Figura");
lblNewLabel.setBounds(47, 21, 38, 14);
frmEuclidesJugandoA.getContentPane().add(lblNewLabel);
JLabel lblNewLabel_1 = new JLabel("Par\u00E1metro");
lblNewLabel_1.setBounds(27, 59, 58, 14);
frmEuclidesJugandoA.getContentPane().add(lblNewLabel_1);
JLabel lblNewLabel_2 = new JLabel("\u00C1REA");
lblNewLabel_2.setBounds(26, 97, 46, 14);
frmEuclidesJugandoA.getContentPane().add(lblNewLabel_2);
JLabel lblNewLabel_3 = new JLabel("PER\u00CDMETRO");
lblNewLabel_3.setBounds(27, 128, 58, 14);
frmEuclidesJugandoA.getContentPane().add(lblNewLabel_3);
```

```
textField = new JTextField();
textField.setBounds(117, 18, 86, 20);
frmEuclidesJugandoA.getContentPane().add(textField);
 textField.setColumns(10);
textField_1 = new JTextField();
textField_1.setBounds(117, 56, 86, 20);
frmEuclidesJugandoA.getContentPane().add(textField_1);
textField_1.setColumns(10);

textField_2 = new JTextField();
textField_2.setBounds(117, 94, 86, 20);
frmEuclidesJugandoA.getContentPane().add(textField_2);
textField_2.setColumns(10);

textField_3 = new JTextField();
textField_3.setBounds(117, 122, 86, 20);
frmEuclidesJugandoA.getContentPane().add(textField_3);
textField_3.setColumns(10);
);
```

```
JButton btnNewButton = new JButton("AGREGAR");
btnNewButton.addActionListener(new ActionListener() {  //Manejo de evento
public void actionPerformed(ActionEvent arg0) {
if (textField.getText().equals("Círculo")){
misFigs.agregaFigura(new Círculo(Double.parseDouble(textField_1.getText())));
textField.setText("YA");
}
else
if (textField.getText().equals("Cuadrado")){
misFigs.agregaFigura(new Cuadrado(Double.parseDouble(textField_1.getText())));
textField.setText("YA");
}
else
textField.setText("Figura desconocida");
}
});
btnNewButton.setBounds(10, 171, 91, 23);
frmEuclidesJugandoA.getContentPane().add(btnNewButton);  //botón sobre el fondo
```

```
JButton btnNewButton_1 = new JButton("Calcula \u00C1rea");
btnNewButton_1.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent arg0) {
textField_2.setText(""+misFigs.dameArea(Integer.parseInt(textField_1.getText())));
}
});
btnNewButton_1.setBounds(156, 171, 123, 23);
frmEuclidesJugandoA.getContentPane().add(btnNewButton_1);

JButton btnNewButton_2 = new JButton("Calcula Per\u00EDmetro");
btnNewButton_2.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent arg0) {
textField_3.setText(""+misFigs.damePerímetro(Integer.parseInt(textField_1.getText())))
    ;
}
});
btnNewButton_2.setBounds(295, 171, 123, 23);
frmEuclidesJugandoA.getContentPane().add(btnNewButton_2);
}
}
```

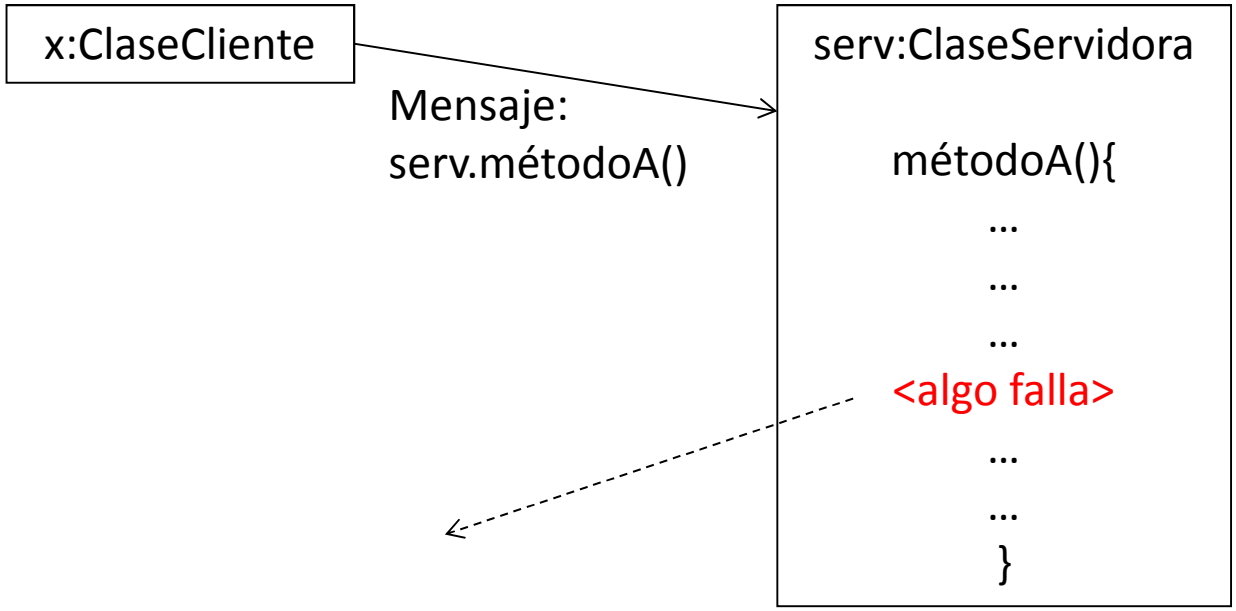
Excepciones

Excepción

- Anomalía que se produce al momento de ejecución
- Condición no usual en un programa

- Se manifiesta cuando el programa interrumpe su funcionamiento, al no saber “qué hacer”.
- En Java muestra un trazado de ejecución.

Ocurrencia de excepción



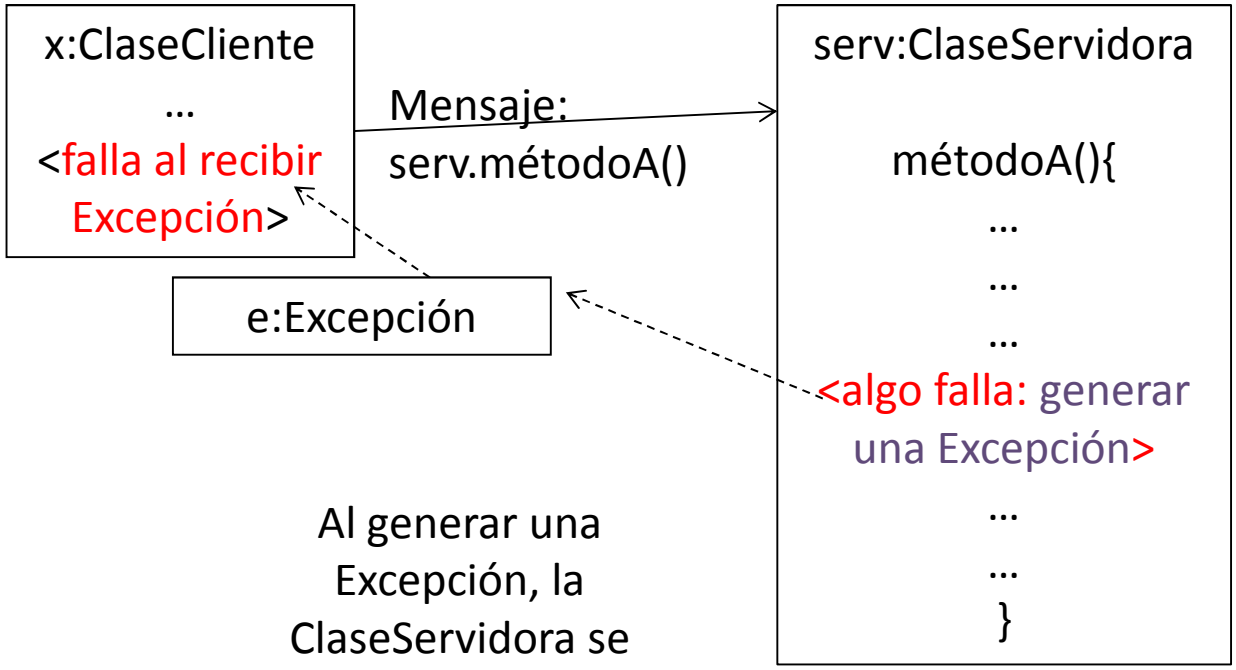
El programa se detiene y manda la pila de ejecución indicando la falla

Excepciones no fatales

- A veces no causa fin de programa
- Cuenta: al querer sacar más de lo que hay
- Al procesar un pedido mal hecho: se envía mensaje

- Generalmente la respuesta es un poco retorcida: un valor extraño (p. ej. Retiro negativo) o un “false” como regreso, sin detalles

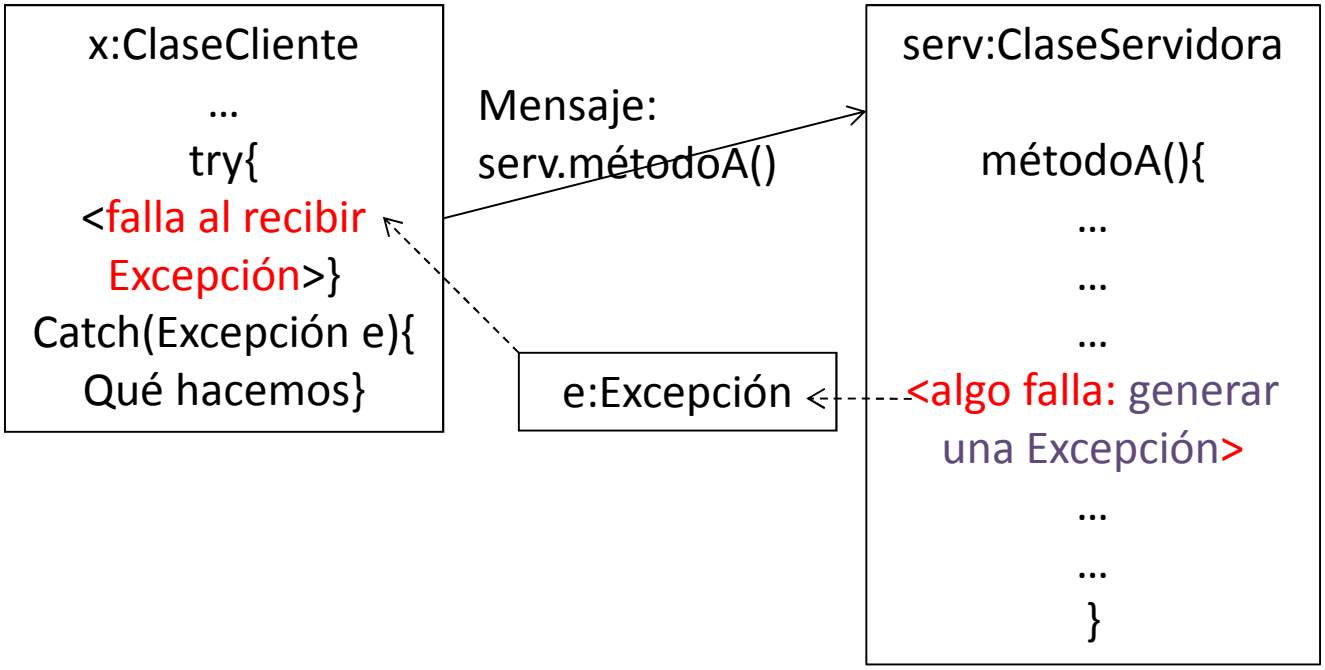
Manejo de excepción



Al generar una Excepción, la ClaseServidora se desentiende del problema y se lo deja a ClaseCliente

Puede ser generada por alguna clase de Java o del usuario

Manejo de excepción

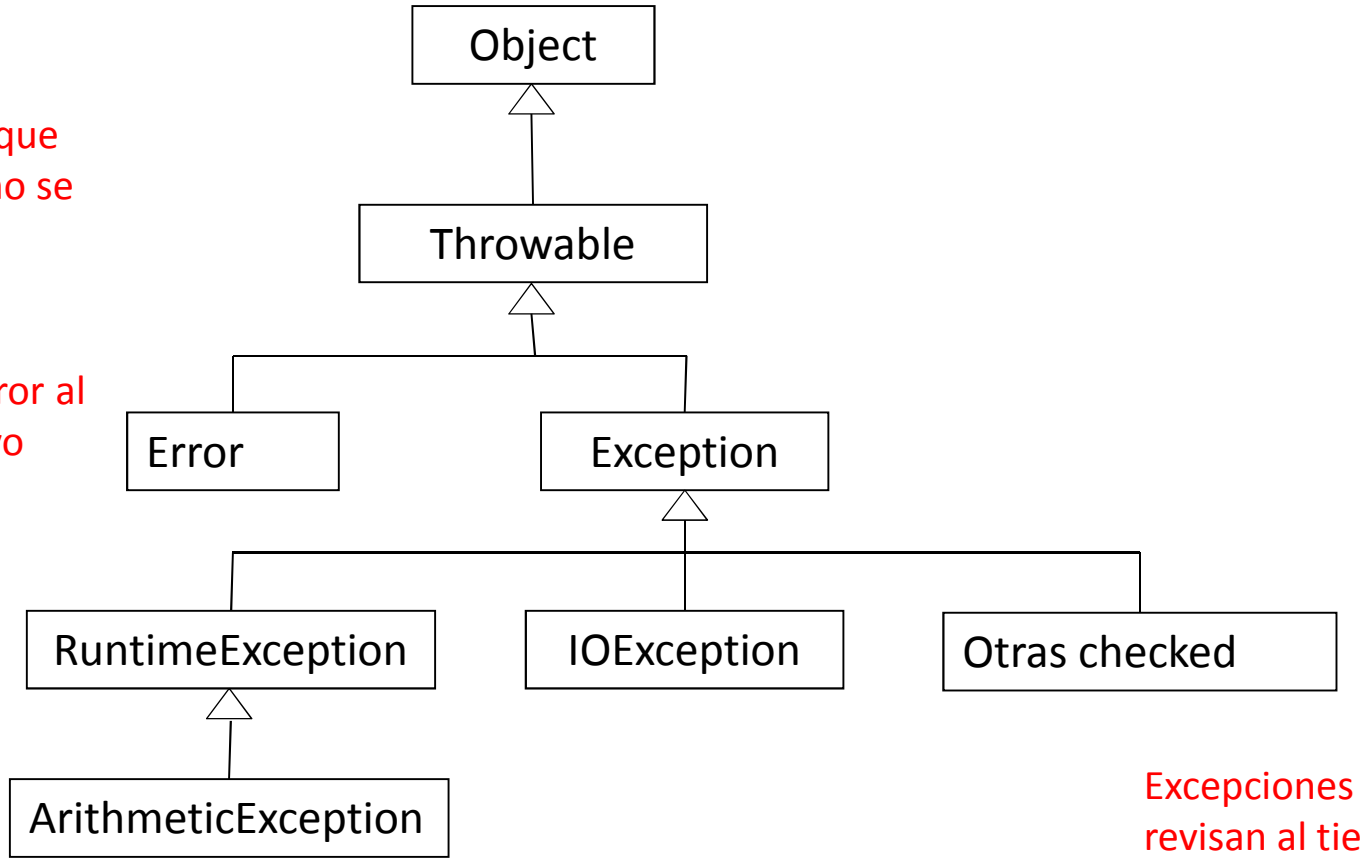


Atrapando la excepción

```
try{
        acción segura;
        acción riesgosa;
        acción riesgosa;
        acción segura;
        …
    }
    catch(NombreExcepción1   e1)        //e1 es objeto Excepción1
            { acción a tomar en caso de excepción1}
     catch(NombreExcepción2  e2)        //e2 es objeto Excepción2
            { acción a tomar en caso de excepción2}
     …
     finally {acción que debe tomarse en cualquier caso}
```

Los objetos Excepción

Fallas graves que usualmente no se atrapan; son externas a la aplicación. Ejemplo IOError al leer en archivo que sí existe



Excepciones que no se revisan al tiempo de compilar, ocurren imprevistas al correr. Ejemplos: NullPointer, IndexOutOfBounds.

Excepciones que se revisan al tiempo de compilar, exigiendo manejo; InterruptedException, etc.

Los objetos Excepción

- Campo principal: mensaje
- Opcional: causa (es Throwable, es como excepción de excepción)
- Pueden agregarse elementos

- Métodos: getMessage, getStackTrace, printStackTrace, toString

¿Por qué usar excepciones?

- Suponga clase Cuenta:
  - Atributos: saldo, límite diario, activa
  - Método **retira(cant)**
  - regresa cant si es menor o igual a saldo
  - Regresa 0 de otro modo
- ¿Qué pasa si …?
  - Cant =0
  - Cant < 0
  - Excede límite de cajero
  - Excede límite diario
  - No está activo
  - Si es crédito y excede límite

¿Por qué usar excepciones?

- Solución antigua:
  - Regresa cant si es válido
  - -1 si es cero o negativo
  - -2 si sobrepasa saldo
  - -3 si rebasa límite
  - -4 si rebasa límite diario
  - -5 si no está activo
- Quien mandó el mensaje debe verificar la respuesta y tomar decisiones, que quedan integradas en el código de las situaciones normales;
- Si cambian reglas de banco, debe revisarse todo, la asignación de números, las preguntas, etc.
- Al manejarlo como excepción, permite ignorar problemas o atenderlos, pero siempre como algo especial, no de diario

¿Por qué usar excepciones?

- Otra Solución antigua: enviar mensajes de error, ¿quién los leerá?
- Las clases no «leen» mensajes y tiene los mismos problemas que el envío de números negativos.

- El problema se complica si el valor de retorno es un objeto, ¿qué objeto regresará para avisar que está mal?

¿Por qué usar excepciones?

- Realmente, esas situaciones no cumplen lo que espera la clase Cuenta, es decir, no cumplen con el contrato entre cliente y servidor.
- La excepción es un mecanismo más claro y flexible.

Manejo de excepciones

Si en un método puede ocurrir una excepción:

a. Se puede atrapar ahí (try … catch)
b. Se puede posponer (throws XException)

La clase que llamó al método que hace el throw

a. Puede atraparla (try … catch)
b. Puede posponer la atención (throws XException)

Y así sucesivamente

Ejemplo en etapas

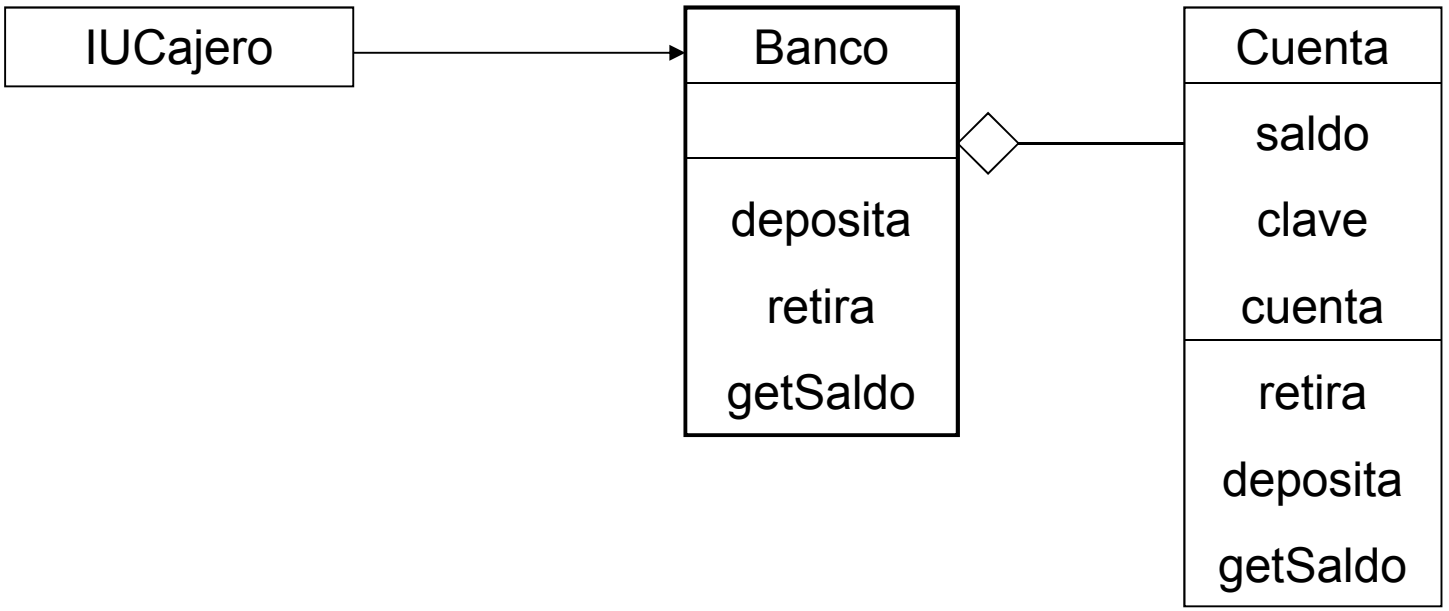
A continuación se presenta un ejemplo en etapas. Para una variante de un problema de cuentas bancarias, se tiene una interfaz gráfica para hacer depósitos, retiros y consultar saldo. Hay un botón INICIA que crea las cuentas 99001, con clave alfa11; 99002 con clave beta22; 99003, con clave gama33.

La primera versión no usa excepciones y puede verse que algunos casos dan resultados confusos, al no distinguir la causa de una negativa.

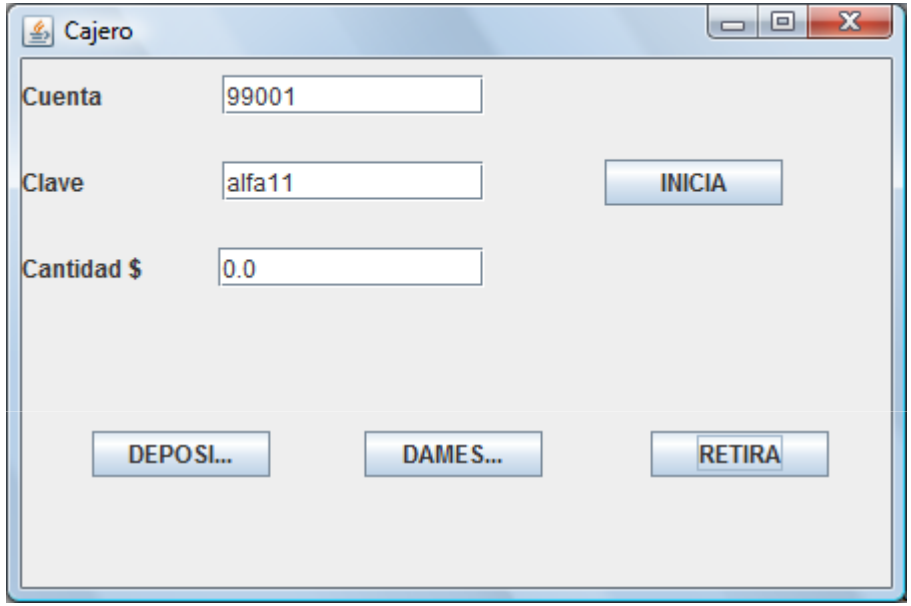
La segunda versión usa una excepción llamada ExcepciónBanco y la lanza la clase Cuenta cuando los parámetros no son aceptables. Se espera que la atrape la clase Banco. Si no lo hace se producirá una excepción en el programa.

La tercera versión es semejante a la segunda, pero la clase Banco declina atrapar la excepción, indicándolo con la expresión “throws ExcepciónBanco”. Entonces, debe atraparla la clase interfaz de usuario IUCajero.

Ejemplo


IUCajero
Banco
deposita
retira
getSaldo
Cuenta
saldo
clave
cuenta
retira
deposita
getSaldo

Ventana del ejemplo.



Cuenta simple, sin excepciones

```
public double getSaldo(String cla){
// solo se informa al cliente que sepa la clave
        if (clave.equals(cla))
                return saldo;
        else
                return -9999.99;
   }
```

```
public double retira(String cla, double kant){
        //debe tener la clave para retirar
        //debe estar activa
        //debe tener saldo suficiente para retirar
        double ret=0;
        if (clave.equals(cla)){
                if (activo){
                        if (saldo >= kant){
                                ret = kant;
                        }
                }
        }
        return ret;
    }
```

Excepción generada en Cuenta

```
public double getSaldo(String cla){
        // solo se informa al cliente que sepa la clave
        if (clave.equals(cla))
                return saldo;
        else
                throw new ExcepciónBanco("clave
    equivocada");
    }
```

```
public double retira(String cla, double kant){
        double ret=0;
        if (clave.equals(cla)){
                if (activo){
                        if (saldo >= kant){
                                ret = kant;
                                return ret;
                                }
                        else throw new ExcepciónBanco("saldo
    insuficiente");
                }
                else throw new ExcepciónBanco("cuenta
    inactiva");
        }
        else throw new ExcepciónBanco("clave equivocada");
    }
```

Clase Excepción

```
public class ExcepciónBanco extends
   RuntimeException {
public ExcepciónBanco(String mensaje){
super(mensaje);
}
}
```

Captura de excepción en banco

```
public double getSaldo(String cta, String kla){
        double resp=0;
        int res = busca(cta);
        if (res>=0){
                CuentaE kue = listaCuentas.get(res);
                try{
                        resp = kue.getSaldo(kla);
                }catch(ExcepciónBanco eb){
                        JOptionPane.showMessageDialog(null,
   "No puedo dar saldo: "+eb.getMessage());
                }
        }
        return resp;
   }
```

```
public double retira(String cta, String kla, double k){
        double resp=0;
        int res = busca(cta);
        if (res>=0){
                CuentaE kue = listaCuentas.get(res);
                try{
                        resp = kue.retira(kla, k);
                }catch(ExcepciónBanco eb){
                        JOptionPane.showMessageDialog(null,
"No se pudo retirar: "+eb.getMessage());
                }
        }
        return resp;
}
```

Banco evade atraparla

```
    public double retira(String cta, String kla, double k)throws
    ExcepciónBanco{
            double resp=0;
            int res = busca(cta);
            if (res>=0){
                    CuentaE kue = listaCuentas.get(res);
//                  try{
                            resp = kue.retira(kla, k);
//                  }catch(ExcepciónBanco eb){
//                          JOptionPane.showMessageDialog(null,
    "No se pudo retirar: "+eb.getMessage());
//                  }
            }
            return resp;
    }
```

```
    public double getSaldo(String cta, String kla)throws
    ExcepciónBanco{
        double resp=0;
        int res = busca(cta);
        if (res>=0){
            CuentaE kue = listaCuentas.get(res);
//          try{
                resp = kue.getSaldo(kla);
//          }catch(ExcepciónBanco eb){
//              JOptionPane.showMessageDialog(null,
    "No puedo dar saldo: "+eb.getMessage());
//          }
        }
        return resp;
    }
```

La atrapa Interfaz

```
JButton btnNewButton = new JButton("DAMESaldo");
    btnNewButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
        try{

        textField_2.setText(""+miBanco.getSaldo(textField.getTex
    t(), textField_1.getText()));
                    }catch(ExcepciónBanco eb){

        JOptionPane.showMessageDialog(null,"No pudo dar
    saldo: "+eb.getMessage());
                    }
                    }
        });
```

```
        JButton btnNewButton_1 = new JButton("RETIRA");
        btnNewButton_1.addActionListener(new ActionListener()
{
                public void actionPerformed(ActionEvent arg0) {
                        try{

        textField_2.setText(""+miBanco.retira(textField.getText(),
textField_1.getText(),
Double.parseDouble(textField_2.getText())));
                }catch(ExcepciónBanco eb){

        JOptionPane.showMessageDialog(null,"No pudo retirar:
"+eb.getMessage());
                }
                }
        });
```