unelweb
Instituto de nuevas Tecnologías

Guía de

Orientada a Objetos

Objetivos

- Aprender los conceptos básicos de la programación orientada a objetos, comportamiento, relaciones y operaciones.
- Comprender por qué la programación orientada a objetos sirve para reutilizar código.
- Comprender los principios básicos de la programación orientada a objetos.

Puntos:

1. Objeto: definición y características generales.
2. Clase: definición y características generales.
3. Relación entre Objeto y Clase.
4. Atributos.
5. Métodos: definición y características generales.
6. Creación de Objetos y Métodos Constructores.
7. Métodos de destrucción de Objetos.
8. Encapsulamiento (ocultamiento de información)
9. Modularidad
10. Herencia
11. Polimorfismo

La Programación Orientada a Objetos (POO) es una metodología que basa la estructura de los programas en torno a los objetos. Los lenguajes de POO ofrecen medios y herramientas para describir los objetos manipulados por un programa. Más que describir cada objeto individualmente, estos lenguajes proveen una construcción (Clase) que describe a un conjunto de objetos que poseen las mismas propiedades.

OBJETO

Es una entidad (tangible o intangible) que posee características y acciones que realiza por sí solo o interactuando con otros objetos. Un objeto es una entidad caracterizada por sus atributos propios y cuyo comportamiento está determinado por las acciones o funciones que pueden modificarlo, así como también las acciones que requiere de otros objetos. Un objeto tiene identidad e inteligencia y constituye una unidad que oculta tanto datos como la descripción de su manipulación. Puede ser definido como una encapsulación y una abstracción: una encapsulación de atributos y servicios, y una abstracción del mundo real.

Para el contexto del Enfoque Orientado a Objetos (EOO) un objeto es una entidad que encapsula datos (atributos) y acciones o funciones que los manejan (métodos). También para el EOO un objeto se define como una instancia o particularización de una clase. Los objetos de interés durante el desarrollo de software no sólo son tomados de la vida real (objetos visibles o tangibles), también pueden ser abstractos. En general son entidades que juegan un rol bien definido en el dominio del problema. Un libro, una persona, un carro, un polígono, son apenas algunos ejemplos de objeto.

Cada objeto puede ser considerado como un proveedor de servicios utilizados por otros objetos que son sus clientes. Cada objeto puede ser a la vez proveedor y cliente. De allí que un programa pueda ser visto como un conjunto de relaciones entre proveedores clientes. Los servicios ofrecidos por los objetos son de dos tipos:

a. Los datos, que llamamos **atributos**.
b. Las acciones o funciones, que llamamos **métodos**.

Características Generales

a. *Un objeto se identifica por un nombre o un identificador único que lo diferencia de los demás***.** Ejemplo: el objeto Cuenta de Ahorros número 12345 es diferente al objeto Cuenta de Ahorros número 25789. En este caso el identificador que los hace únicos es el número de la cuenta.

b. *Un objeto posee estados.* El estado de un objeto está determinado por los valores que poseen sus atributos en un momento dado.

c. *Un objeto tiene un conjunto de métodos.* El comportamiento general de los objetos dentro de un sistema se describe o representa mediante sus operaciones o métodos. Los métodos se utilizarán para obtener o cambiar el estado de los objetos, así como para proporcionar un medio de comunicación entre objetos.

d. *Un objeto tiene un conjunto de atributos*. Los atributos de un objeto contienen valores que determinan el estado del objeto durante su tiempo de vida. Se implementan con variables, constantes y estructuras de datos (similares a los campos de un registro).

e. *Los objetos soportan encapsulamiento.* La estructura interna de un objeto normalmente está oculta a los usuarios del mismo. Los datos del objeto están disponibles solo para ser manipulados por los propios métodos del objeto. El único mecanismo que lo conecta con el mundo exterior es el paso de mensajes.

f. *Un objeto tiene un tiempo de vida dentro del programa o sistema que lo crea y utiliza.* Para ser utilizado en un algoritmo el objeto debe ser creado con una instrucción particular (*New* ó Nuevo) y al finalizar su utilización es destruido con el uso de otra instrucción o de manera automática.

CLASE

La clase es la unidad de modularidad en el EOO. La tendencia natural del individuo es la de clasificar los objetos según sus características comunes (clase). Por ejemplo, las personas que asisten a la universidad se pueden clasificar (haciendo abstracción) en estudiante, docente, empleado e investigador.

La clase puede definirse como la agrupación o colección de objetos que comparten una estructura común y un comportamiento común. Es una plantilla que contiene la descripción general de una colección de objetos. Consta de atributos y métodos que resumen las características y comportamientos comunes de un conjunto de objetos.
Todo objeto (también llamado instancia de una clase), pertenece a alguna clase. Mientras un objeto es una entidad concreta que existe en el tiempo y en el espacio, una clase representa solo una abstracción.

Todos aquellos objetos que pertenecen a la misma clase son descritos o comparten el mismo conjunto de atributos y métodos. Todos los objetos de una clase tienen el mismo formato y comportamiento.

Su sintaxis algorítmica es:

```
Clase <Nombre de la Clase>
…
FClase <Nombre de la Clase>;
```

En Java sería de esta manera:

```
class [nombre de la clase] {
  [atributos o variables de la clase]
  [métodos o funciones de la clase]
  [main]
}
```

Características Generales

a. *Una clase es un nivel de abstracción alto.* La clase permite describir un conjunto de características comunes para los objetos que representa. Ejemplo: La clase Avión se puede utilizar para definir los atributos (tipo de avión, distancia, altura, velocidad de crucero, capacidad, país de origen, etc.) y los métodos (calcular posición en el vuelo, calcular velocidad de vuelo, estimar tiempo de llegada, despegar, aterrizar, volar, etc.) de los objetos particulares Avión que representa.

b. *Un objeto es una instancia de una clase.* Cada objeto concreto dentro de un sistema es miembro de una clase específica y tiene el conjunto de atributos y métodos especificados en la misma.

c. *Las clases se relacionan entre sí mediante una jerarquía.* Entre las clases se establecen diferentes tipos de relaciones de herencia, en las cuales la clase Hija (subclase) hereda los atributos y métodos de la clase Padre (superclase), además de incorporar sus propios atributos y métodos.

   Ejemplos:

   Superclase: Clase Avión.

   Subclases de Avión: Clase Avión Comercial, Avión de Combate, Avión de Transporte.

d. Los nombres o identificadores de las clases deben colocarse en singular (Clase Animal, Clase Carro, Clase Alumno).

RELACIÓN ENTRE OBJETO Y CLASE

Algorítmicamente, las clases son descripciones netamente estáticas o plantillas que describen objetos. Su rol es definir nuevos tipos conformados por atributos y operaciones.

Por el contrario, los objetos son instancias particulares de una clase. Las clases son una especie de molde de fábrica, en base al cual son construidos los objetos. Durante la ejecución de un programa sólo existen los objetos, no las clases.

La declaración de una variable de una clase NO crea el objeto. La asociación siguiente: <Nombre_Clase> <Nombre_Variable>; (por ejemplo, Rectángulo R), no genera o no crea automáticamente un objeto Rectángulo. Sólo indica que R será una referencia o una variable de objeto de la Clase Rectángulo.

La creación de un objeto debe ser indicada explícitamente por el programador de forma análoga a como inicializamos las variables con un valor dado sólo que para los objetos se hace a través de un método Constructor.

ATRIBUTO

Son los datos o variables que caracterizan al objeto y cuyos valores en un momento dado indican su estado.

Un atributo es una característica de un objeto. Mediante los atributos se define información oculta dentro de un objeto, la cual es manipulada solamente por los métodos definidos sobre dicho objeto. Un atributo consta de un nombre y un valor. Cada atributo está asociado a un tipo de dato, que puede ser simple (entero, real, lógico, carácter, string) o estructurado (arreglo, registro, archivo, lista, etc.).

Su sintaxis algorítmica es:

<Modo de Acceso> <Tipo de dato> <Nombre del Atributo>;

Los modos de acceso son:

a. ***Público:*** Atributos (o Métodos) que son accesibles fuera de la clase. Pueden ser llamados por cualquier clase, aun si no está relacionada con ella. Este modo de acceso también se puede representar con el símbolo **+.**

b. ***Privado:*** Atributos (o Métodos) que sólo son accesibles dentro de la implementación de la clase. También se puede representar con el símbolo **–.**

c. ***Protegido:*** Atributos (o Métodos) que son accesibles para la propia clase y sus clases hijas (subclases). También se puede representar con el símbolo **#.**

MÉTODO

Son las operaciones (acciones o funciones) que se aplican sobre los objetos y que permiten crearlos, cambiar su estado o consultar el valor de sus atributos. Los métodos constituyen la secuencia de acciones que implementan las operaciones sobre los objetos. La implementación de los métodos no es visible fuera de objeto.

La sintaxis algorítmica de los métodos expresados como funciones y acciones es:

- Para funciones se pueden usar cualquiera de estas dos sintaxis:

  <Modo de Acceso> Función <Nombre> [(Lista Parámetros)]: <Descripción del Tipo de datos>

- Para acciones:

  <Modo de Acceso> Acción <Nombre> [(Lista Parámetros)]

donde los parámetros son opcionales.

Características Generales

a. *Cada método tiene un nombre, cero o más parámetros (por valor o por referencia) que recibe o devuelve y un algoritmo con el desarrollo del mismo.*

b. *En particular se destaca el método constructor, que no es más que el método que se ejecuta cuando el objeto es creado.* Este constructor suele tener el mismo nombre de la Clase/Objeto, pero aunque es una práctica común, el método constructor no necesariamente tiene que llamarse igual a la clase (al menos, no en pseudo-código). Es un método que recibe cero o más parámetros y, lo usual, es que inicialicen los valores de los atributos del objeto.

c. En lenguajes como Java y C++ se puede definir más de un método constructor, que normalmente se diferencian entre sí por la cantidad de parámetros que reciben.

d. *Los métodos se ejecutan o activan cuando el objeto recibe un mensaje*, enviado por un objeto o clase externo al que lo contiene, o por el mismo objeto de manera local.

CREACIÓN DE OBJETOS Y MÉTODOS CONSTRUCTORES

Cada objeto o instancia de una clase debe ser creada explícitamente a través de un método u operación especial denominado **constructor.** Los atributos de un objeto toman valores iniciales dados por el constructor. Por convención, el método constructor tiene el mismo nombre de la clase y no se le asocia un modo de acceso (**es público**).

Algunos lenguajes proveen un método constructor por defecto para cada clase y/o permiten la definición de más de un método constructor.

MÉTODO DE DESTRUCTORES DE OBJETOS

Los objetos que ya no son utilizados en un programa, ocupan inútilmente espacio de memoria, que es conveniente recuperar en un momento dado. Según el lenguaje de programación utilizado esta tarea es dada al programador o es tratada automáticamente por el procesador o soporte de ejecución del lenguaje.

El siguiente ejemplo describe cómo implementar una clase con sus atributos y métodos en (Pseudo-Código):

```
Clase CuentaBancaria
// atributos
Privado Entero Saldo;
Privado String NroCuenta;
Privado String Titular;
// métodos

Acción CuentaBancaria(Entero montoInicial; String num, nombre)
// asigna a los atributo de la clase sus valores iniciales
Saldo = montoInicial;
NroCuenta = num;
Titular = nombre;
Facción;

Público Acción depositar(Entero cantidad)
// incrementa el saldo de la cuenta
Saldo = Saldo + cantidad;
Facción;
```

```
Público Acción retirar(Entero cantidad)
// disminuye el saldo de la cuenta
Saldo = Saldo - cantidad;
Facción;

Público Función obtenerSaldo: Entero
// permite conocer el monto disponible o saldo de la cuenta
Retornar(saldo);
FFunción;
FinClase CuentaBancaria
```

Ahora veamos un ejemplo en java de la implementación de la clase persona:

```
public class Persona {
      private String nombre;
      Private int edad;
      private String ciudad;
      private int telefono;
      public Persona (Sting n, int e, String c, int t){
            this.nombre = n;
            this.edad = e;
            this.ciudad = c;
            this.telefono = t;
      }
      public Persona (){ //Constructor sin param.
            this.nombre = “”; //nombre vacio
            this.edad = 0; //edad cero
            this.ciudad = “”; //ciudad vacia
            this.telefono = 0; //teléfono cero
      }
      public void cambiarEdad (int e){
```

```
            this.edad = e;
        }
        public String consultarNombre (){
            return nombre;
        }
        public int consultarEdad (){
            return edad;
        }
        public String consultarCiudad (){
            return ciudad;
        }
        public int consultarTelefono (){
            return telefono;
        }
}
```

ENCAPSULAMIENTO (OCULTAMIENTO DE INFORMACIÓN)

Es la propiedad del EOO que permite ocultar al mundo exterior la representación interna del objeto. Esto quiere decir que el objeto puede ser utilizado, pero los datos esenciales del mismo no son conocidos fuera de él.

La idea central del encapsulamiento es esconder los detalles y mostrar lo relevante. Permite el ocultamiento de la información, separando el aspecto correspondiente a la especificación de la implementación; de esta forma, distingue el "qué hacer" del "cómo hacer". La especificación es visible al usuario, mientras que la implementación se le oculta.

El encapsulamiento en un sistema orientado a objeto se representa en cada clase u objeto, definiendo sus atributos y métodos con los siguientes modos de acceso:

a. *Público (+)*: Atributos o Métodos que son accesibles fuera de la clase. Pueden ser llamados por cualquier clase, aun si no está relacionada con ella.

b. *Privado (-)*: Atributos o Métodos que solo son accesibles dentro de la implementación de la clase.

c. *Protegido (#)*: Atributos o Métodos que son accesibles para la propia clase y sus clases hijas (subclases).

Los atributos y los métodos que son públicos constituyen la interfaz de la clase, es decir, lo que el mundo exterior conoce de la misma. Normalmente lo usual es que se oculten los atributos de la clase y solo sean visibles los métodos, incluyendo entonces algunos de consulta para ver los valores de los atributos. El método constructor (Nuevo, New) siempre es Público.

MODULARIDAD

Es la propiedad que permite tener independencia entre las diferentes partes de un sistema. La modularidad consiste en dividir un programa en módulos o partes, que pueden ser compilados separadamente, pero que tienen conexiones con otros módulos. En un mismo módulo se suele colocar clases y objetos que guarden una estrecha relación. El sentido de modularidad está muy relacionado con el ocultamiento de información.

HERENCIA

Es el proceso mediante el cual un objeto de una clase adquiere propiedades definidas en otra clase que lo preceda en una jerarquía de clasificaciones. Permite la definición de un nuevo objeto a partir de otros, agregando las diferencias entre ellos (Programación Diferencial), evitando repetición de código y permitiendo la reusabilidad.
Las clases heredan los datos y métodos de la superclase. Un método heredado puede ser sustituido por uno propio si ambos tienen el mismo nombre.

La herencia puede ser simple (cada clase tiene sólo una superclase) o múltiple (cada clase puede tener asociada varias superclases). La clase Docente y la clase Estudiante heredan las propiedades de la clase Persona (superclase, herencia simple). La clase Preparador (subclase) hereda propiedades de la clase Docente y de la clase Estudiante (herencia múltiple).
En java no existe herencia múltiple.

POLIMORFISMO

Es una propiedad del EOO que permite que un método tenga múltiples implementaciones, que se seleccionan en base al tipo objeto indicado al solicitar la ejecución del método.

El polimorfismo operacional o sobrecarga operacional permite aplicar operaciones con igual nombre a diferentes clases o están relacionados en términos de inclusión. En este tipo de polimorfismo, los métodos son interpretados en el contexto del objeto particular, ya que los métodos con nombres comunes son implementados de diferente manera dependiendo de cada clase. Por ejemplo, el área de un cuadrado,

rectángulo y círculo, son calculados de manera distinta; sin embargo, en sus clases respectivas puede existir la implementación del área bajo el nombre común Área. En la práctica y dependiendo del objeto que llame al método, se usara el código correspondiente.

Ejemplos:

Superclase: Clase Animal
Subclases: Clases Mamífero, Ave, Pez.

Se puede definir un método Comer en cada subclase, cuya implementación cambia de acuerdo a la clase invocada, sin embargo, el nombre del método es el mismo.

Mamifero.Comer , Ave.Comer , Pez.Comer

Otro ejemplo de polimorfismo es el operador +. Este operador tiene dos funciones diferentes de acuerdo al tipo de dato de los operandos a los que se aplica. Si los dos elementos son numéricos, el operador + significa suma algebraica de los mismos, en cambio sí, por lo menos uno de los operandos es un String o Caracter, el operador es la concatenación de cadenas de caracteres.

Otro ejemplo de sobrecarga es cuando tenemos un método definido originalmente en la clase madre, que ha sido adaptado o modificado en la clase hija. Por ejemplo, un método Comer para la clase Animal y otro Comer que ha sido adaptado para la clase Ave, quien está heredando de la clase Animal.

Puntos

1. Tipos de excepciones
2. Manejo de Excepciones
3. Captura de Excepciones

En Java los errores son conocidos como e**xcepciones**. Cuando se produce una excepción, se crea un objeto del mismo tipo de la excepción. En lenguaje Java las excepciones pueden manejarse con las clases que extienden el paquete Throwable de manera directa o indirecta, pero existen diversos tipos de excepciones y formas para manejarlas. La clase Throwable, es la superclase de todas las clases de manejo de errores.

Cuando se genera una excepción, el programa Java busca un manejador para el error (handler). El handler es una porción del código que indica el tipo de objeto correspondiente al error que se van a ejecutar luego de que ocurrió el error.

Se pueden dar varios tipos de errores:

- Al tratar de acceder a elementos de arreglos con un índice mayor al del último elemento del arreglo.
- Divisiones para cero.
- Manejo de archivos. No existe, no se tiene suficientes permisos, etc.
- Errores accediendo a bases de datos.
- Errores definidos por el usuario. Tarjeta incorrecta, excede cupo de transferencia, etc.

TIPOS DE EXCEPCIONES

- Checked: son las excepciones que revisan el compilador.
- Unchecked: son las excepciones que no revisa el compilador y se dan en tiempo de ejecución. Ejemplo: RuntimeException.

MANEJO DE EXCEPCIONES

Cuando un programa Java viola las restricciones semánticas del lenguaje (se produce un error), la máquina virtual Java comunica este hecho al programa mediante una excepción. La aplicación Java no debe morir y generar un Core (o un crash en caso del DOS), al contrario, se lanza (throw) una excepción y se captura (catch) para resolver esa situación de error.

Muchas clases de errores pueden provocar una excepción, desde un desbordamiento de memoria o un disco duro estropeado, hasta un disquete protegido contra escritura, un intento de dividir por cero o intentar acceder a un vector fuera de sus límites. Cuando esto ocurre, la máquina virtual Java crea un objeto de la clase exception o error y se notifica el hecho al sistema de ejecución.

Se dice que se ha lanzado una excepción ("Throwing Exception").

Un método se dice que es capaz de tratar una excepción ("Catch Exception") si ha previsto el error que se ha producido y prevé también las operaciones a realizar para "recuperar" el programa de ese estado de error. En el momento en que es lanzada una excepción, la máquina virtual Java recorre la pila de llamadas de métodos en busca de alguno que sea capaz de tratar la clase de excepción lanzada.

Para ello, comienza examinando el método donde ese ha producido la excepción; si este método no es capaz de tratarla, examina el método desde el que se realizó la llamada al método donde se produjo la excepción y así sucesivamente hasta llegar al último de ellos. En caso de que ninguno de los métodos de la pila sea capaz de tratar la excepción, la máquina virtual Java muestra un mensaje de error y el programa termina.

Los programas escritos en Java también pueden lanzar excepciones explícitamente mediante la instrucción throw, lo que facilita la devolución de un "código de error" al método que invocó el método que causó el error.

Ejemplo:

```
public class Excepciones1 {
public static void main(String[] arg) {
      metodo();
 }

 static void metodo() {
 int divisor = 0;
 int resultado = 100/divisor;
 System.out.println("Resultado: " + resultado);
 System.out.println("Una Suma:" + (3+4));
       }
}
```

Si intentamos ejecutar este programa obtendremos el siguiente resultado:

Exception in thread ''main'' java.lang.ArithmeticException: / by zero ……

Lo que ha ocurrido es que la máquina virtual Java ha detectado una condición de error y ha creado un objeto de la clase java.lang.ArithmeticException. Como el método donde se ha producido la excepción no es capaz de tratarla, se trata por la máquina virtual Java, que muestra el mensaje de error anterior y finaliza la ejecución del programa.

CAPTURA DE EXCEPCIONES

Un manejador de excepciones es una porción de código que se va a encargar de tratar las posibles excepciones que se puedan generar. En Java, de forma similar a C++ se pueden tratar las excepciones previstas por el programador utilizando unos mecanismos, los manejadores de excepciones, que se estructuran en tres bloques:

a. **El bloque try:** Lo primero que hay que hacer para que un método sea capaz de tratar una excepción generada por la máquina virtual Java es encerrar las instrucciones susceptibles de generarla en un bloque try.

```
try {
 Bloque de Instrucciones
}
```

Cualquier excepción que se produzca dentro del bloque try será analizado por el bloque o bloques catch que se verá en el punto siguiente. En el momento en que se produzca la excepción, se abandona el bloque try y, por lo tanto, las instrucciones que sigan al punto donde se produjo la excepción no serán ejecutadas. Cada bloque try debe tener asociado al menos un bloque catch.

b. **El bloque catch:** Por cada bloque try pueden declararse uno o varios bloques catch, cada uno de ellos capaz de tratar un tipo de excepción.

```
try {
      Bloque de Instrucciones
 } catch (TipoExcepción nombreVariable) {
     Bloque de Instrucciones del primer catch
} catch (TipoExcepción nombreVariable) {
```

```
        Bloque de Instrucciones del segundo catch
    } ...
```

Para declarar el tipo de excepción que es capaz de tratar un bloque catch, se declara un objeto cuya clase es la clase de la excepción que se desea tratar o una de sus superclases.

Ejemplo:

```
class ExcepcionTratada {
    public static void main(String[] arg) {
        int i=5; j=0;
        try {
            int k=i/j;
            System.out.println("Esto no se va a
            ejecutar");
        }catch (ArithmeticException ex) {
            System.out.println("Has intentado dividir por
            cero");
        }
    System.out.println("Fin del programa");
    }
}
```

c. **El bloque finally** (no existente en C++): El bloque finally se utiliza para ejecutar un bloque de instrucciones sea cual sea la excepción que se produzca. Este bloque se ejecutará en cualquier caso, incluso si no se produce ninguna excepción. Sirve para no tener que repetir código en el bloque try y en los bloques catch.

```
try {
     Bloque de Instrucciones del try
}
```

```
catch (TipoExcepción nombreVariable) {
      Bloque de Instrucciones del primer catch
}
catch (TipoExcepción nombreVariable) {
      Bloque de Instrucciones del segundo catch
} .....
}
finally {
      Bloque de Instrucciones de finally
}
```

Todos los métodos Java utilizan la sentencia throw para lanzar una excepción. Esta sentencia requiere un sólo argumento (un objeto Throwable).

Este programa lee un fichero (fichero.txt), y lee su contenido en forma de números.

Si alguno de los números leídos es negativo, lanza una excepción MiExcepcion. Además gestiona la excepción IOException, que es una excepción de las que Java incluye y que se lanza si hay algún problema en una operación de entrada/salida.

Ambas excepciones son gestionadas, imprimiendo su contenido (cadena de error) por pantalla.

```
// Creo una excepción personalizada

class MiExcepcion extends Exception {
MiExcepcion(){
super(); // constructor por defecto de Exception
 }
```

```
MiExcepcion( String cadena ){
super( cadena ); // constructor param. de Exception
}
}

// Esta clase lanzará la excepción
class Lanzadora {
void lanzaSiNegativo( int param ) throws MiExcepcion {
      if ( param < 0 )
      throw new MiExcepcion( "Numero negativo" );
      }
}
class Excepciones {
      public static void main( String[] args ) {
      // Para leer un fichero
      Lanzadora lanza = new Lanzadora();
      FileInputStream entrada = null;
      int leo;
      try {
entrada = new FileInputStream( "fich.txt" );
      while ( ( leo = entrada.read() ) != -1 )
      lanza.lanzaSiNegativo( leo );
      entrada.close();
                  System.out.println( "Todo fue bien" );
      } catch ( MiExcepcion e ){ // Personalizada
      System.out.println( "Excepción: " + e.getMessage() );
      } catch ( IOException e ){ // Estándar
      System.out.println( "Excepcion: " + e.getMessage() );
      } finally {
      if ( entrada != null )
      try {
      entrada.close(); // Siempre queda cerrado
      } catch ( Exception e ) {
      System.out.println( "Exception: " + e.getMessage() );
      }
      System.out.println( "Fichero cerrado." );
```

```
        }
        }
}
```

La salida de este programa, suponiendo un número negativo sería:

```
Excepción: Numero negativo
Fichero cerrado
```

En el caso de que no hubiera ningún número negativo sería:

```
Todo fue bien
Fichero cerrado
```

En el caso de que se produjese un error de E/S, al leer el primer número, sería:

```
Excepción: java.io.IOException
Fichero cerrado
```

Objetivo

Conocer las aplicaciones Java utilizando las interfaces AWT y SWING.

Puntos:

- AWT
- SWING
- Estructura básica de una aplicación SWING
- Principales aspectos de una aplicación SWING
- Diálogos
- Algunos componentes de SWING
- Diferencias entre SWING y AWT

AWT

El **AWT** es el acrónimo del X Window Toolkit para Java, donde X puede ser cualquier cosa: Abstract, Alternative, Awkward, Another; aunque parece que Sun se decanta por ***Abstracto***, seriedad por encima de todo. Se trata de una biblioteca de clases Java para el desarrollo de Interfaces de Usuario Gráficas. La versión del AWT que Sun proporciona con el JDK se desarrolló en sólo dos meses y es la parte más débil de todo lo que representa Java como lenguaje. El entorno que ofrece es demasiado simple, no se han tenido en cuenta las ideas de entornos gráficos novedosos, sino que se ha ahondado en estructuras orientadas a eventos, llenas de callbacks y sin soporte alguno del entorno para la construcción gráfica; veremos que la simple acción de colocar un dibujo sobre un botón se vuelve una tarea bastante complicada. Quizá la presión de tener que lanzar algo al mercado haya tenido mucho que ver en la pobreza de AWT.

La estructura básica del AWT se basa en *Componentes* y *Contenedores*. Estos últimos contienen Componentes posicionados a su respecto y son Componentes a su vez, de forma que los eventos pueden tratarse tanto en Contenedores como en Componentes, corriendo por cuenta del programador (todavía no hay herramientas de composición visual) el encaje de todas las piezas, así como la seguridad de tratamiento de los eventos adecuados.

Ejemplo de código PanelSolo:

```
package paneles;
import java.awt.*;
public class PanelSolo {
    public static void main(String[] args) {
```

```
        Frame f = new Frame("Ejemplo");
        Panel p0 = new Panel();
        Panel p1 = new Panel();
        Panel p2 = new Panel();
        Panel p3 = new Panel();
        Panel p4 = new Panel();
        f.add(p0,"North");
        f.add(p1,"West");
        f.add(p2,"South");
        f.add(p3,"Center");
        f.add(p4,"East");
       // Coloreo para distinguir los paneles.
        p0.setBackground(Color.blue);
        p3.setBackground(Color.white);
        p4.setBackground(Color.red);
        f.pack();
       f.setVisible(true);
     }
}
```

SWING

El paquete Swing es parte de la JFC (Java Foundation Classes) en la plataforma Java. La JFC provee facilidades para ayudar a la gente a construir GUIs. Swing abarca componentes como botones, tablas, marcos, etc.

Las componentes Swing se identifican porque pertenecen al paquete *javax.swing*. Swing existe desde la JDK 1.1 (como un agregado). Antes de la existencia de Swing, las interfaces gráficas con el usuario se realizaban a través de AWT (Abstract Window Toolkit), de quien Swing hereda todo el manejo de eventos. Usualmente, para toda componente AWT existe una componente Swing que la reemplaza, por ejemplo, la clase Button de AWT es reemplazada por la clase JButton de Swing (el nombre de todas las componentes Swing comienza con "J").

Las componentes de Swing utilizan la infraestructura de AWT, incluyendo el modelo de eventos AWT, el cual rige cómo una componente reacciona a eventos tales como, eventos de teclado, mouse, etc... Es por esto, que la mayoría de los programas Swing necesitan importar dos paquetes AWT: *java.awt.** y *java.awt.event.**.

Importante: Como regla, los programas no deben usar componentes pesados de AWT junto a componentes Swing, ya que los componentes de AWT son siempre pintados sobre los de Swing. (Por componentes pesados de AWT se entiende Menú, ScrollPane y todas las componentes que heredan de las clases Canvas y Panel de AWT).

ESTRUCTURA BÁSICA DE UNA APLICACIÓN SWING

Una aplicación Swing se construye mezclando componentes con las siguientes reglas:

Debe existir, al menos, un contenedor de alto nivel (Top-Level Container), que provee el soporte que las componentes Swing necesitan para el pintado y el manejo de eventos.

Otras componentes del contenedor de alto nivel (éstas pueden ser contenedores o componentes simples).

Ejemplo 1:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class HolaMundoSwing {
      public static void main(String[] args) {
            JFrame frame = new JFrame("HolaMundoSwing");
            final JLabel label = new JLabel("Hola Mundo");
            frame.getContentPane().add(label);

            //frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

            frame.addWindowListener(new java.awt.event.WindowAdapter(){
            public void windowClosing(WindowEvent e){
            System.exit(0);
            }
      }
      );
```

```
        frame.pack();
        frame.setVisible(true);
        }
}
```

En las primeras líneas se importan los paquetes necesarios.

```
        import javax.swing.*;
        import java.awt.*;
        import java.awt.event.*;
```

Luego se declara la clase *HolaMundoSwing* y en el método *main* se setea el top level container:

```
        public class HolaMundoSwing {
              public static void main(String[] args) {
              JFrame frame = new JFrame("HolaMundoSwing");
              ...
              frame.pack();
              frame.setVisible(true);
              }
        }
```

En el ejemplo, sólo hay un contenedor de alto nivel, un JFrame. Un frame implementado como una instancia de la clase JFrame es una ventana con decoraciones, tales como, borde, título y botones como íconos y para cerrar la ventana. Aplicaciones con un GUI típicamente usan, al menos, un frame. Además, el ejemplo tiene un componente, una etiqueta que dice "Hola Mundo".

final JLabel label = new JLabel("Hola Mundo"); **//construye el JLabel.**

frame.getContentPane().add(label); **//agrega el label al frame.**

Para que el botón de cerrar cierre la ventana, hay dos opciones:

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
//JDK 1.3
frame.addWindowListener(new     java.awt.event.WindowAdapter(){
//versiones anteriores
      public void windowClosing(WindowEvent e){
            System.exit(0);
      }
}
);
```

PRINCIPALES ASPECTOS DE UNA APLICACIÓN SWING

Como ya se dijo antes, cada aplicación Swing debe tener al menos un *top-level container* que contendrá toda la aplicación, estos pueden ser:

- **javax.swing.JFrame**: Una ventana independiente.
- **javax.swing.JApplet**: Un applet.

DIÁLOGOS

Los Diálogos son ventanas de interacción sencilla con el usuario como por ejemplo:

- java.swing.JOptionPane: Ventana de diálogo tipo SI_NO, SI_NO_CANCELAR, ACEPTAR, etc...
- java.swing.JFileChooser: Ventana para elegir un archivo.

- java.swing.JColorChooser.

A un contenedor se le pueden agregar otros contenedores o componentes simples.

ALGUNOS COMPONENTES DE SWING

a. **JFrame y JDialog:** Los dos tipos de ventanas principales que tenemos en Java son JFrame y JDialog. Hay varias diferencias entre ellas y en función de estas diferencias vamos a ver para qué sirven. Si instanciamos un JFrame, en la barra de abajo de Windows nos aparece un nuevo "boton" correspondiente a nuestra aplicación. Si instanciamos un JDialog, no aparece nada.

   Un JFrame tiene un método setIconImage() para cambiar el icono por defecto de la taza de café. JDialog no tiene este método. Un JDialog admite otra ventana (JFrame o JDialog) como padre en el constructor. JFrame no admite padres. Un JDialog puede ser modal, un JFrame no. Todo esto nos indica lo siguiente: Un JFrame debe ser la ventana principal de nuestra aplicación y sólo debe haber una.

   Las ventanas secundarias de nuestra aplicación deben ser JDialog. Los motivos de esto son los siguientes. Al mostrar el JFrame un botón en la barra de herramientas de windows y tener método para cambiar el icono, es la ventana ideal como ventana principal de nuestra aplicación y sólo debe haber una. Nos permite cambiar el icono y sólo debe haber un botón en la barra de herramientas de Windows para nuestra aplicación. Si usamos un JDialog como ventana principal, no tenemos botón en la barra de herramientas y no hay forma fácil de cambiarle el icono.

Los JDialog son ideales para ventanas secundarias porque admiten una ventana padre. Si la VentanaA es padre del JDialogB, entonces el JDialogB siempre estará por delante de VentanaA, nunca quedará por detrás. Lo ideal es que hagamos nuestras ventanas secundarias como JDialog cuyo padre sea el JFrame principal. De esta forma losJDialog siempre serán visibles por encima del JFrame y no se irán detrás ni quedarán ocultos por el JFrame.

Otra ventaja de admitir un padre es que heredan el icono de él. Si hemos cambiado el icono del **JFrame** con el método **setIconImage()**, entonces todos los **JDialog** que hagamos como hijos de este **JFrame** heredarán el icono. Todas las ventanas de nuestra aplicación tendrán el mismo icono en lugar de la taza de café por defecto.

b. **JOptionPane:** Para hacer ventanas sencillas que avisen de un error al usuario y tengan un botón de "Cerrar", o que le pidan una confirmación para hacer algo (como borrar un fichero) y tengan dos botones de "Si" y "No", o para pedirle que elija una opción entre varias disponibles... tenemos suerte. No hay que construir la ventana.

   La clase **JOptionPane** de Java tiene métodos **showConfirmDialog(), showInputDialog(), showOptionDialog(), showMessageDialog()**. Estos métodos mostrarán una ventana modal que pide al usuario una confirmación, que le pide un dato o le muestra un mensaje. En caso de que el usuario tenga que elegir algo, el método usado devuelve el valor elegido. Por ejemplo, para mostrar un aviso de error, nos basta esta simple línea de código:

```
JOptionPane.showMessageDialog(ventanaPadre, "mensaje de error", "título de la ventana", JOptionPane.ERROR_MESSAGE);
```

Esto mostrará el mensaje de error y detendrá la ejecución del código hasta que el usuario cierre la ventana.

Ahora viene el dato importante. Esta ventana es modal y admite un padre. Si ya hay una ventana modal visible en pantalla, deberías pasar esa ventana como padre de esta para no tener problemas.

DIFERENCIAS ENTRE SWING Y AWT

Los componentes Swing, están escritos, se manipulan y se despliegan completamente en Java (ofrecen mayor portabilidad y flexibilidad). Por ello se les llama componentes puros de Java. Como están completamente escritos en Java y no les afectan las complejas herramientas GUIs de la plataforma en la que se utilizan, también se les conoce comúnmente como componentes ligeros. De hecho, de las principales diferencias entre los componentes de java.awt y de javax.swing es que los primeros están enlazados directamente a las herramientas de la interfaz gráfica de usuario de la plataforma local. Por lo tanto, un programa en Java que se ejecuta en distintas plataformas Java tiene una apariencia distinta e incluso, algunas veces hasta la interacciones del usuario son distintas en cada plataforma (a la apariencia y a la forma en que el usuario interactúa con el programa se les conoce como la "apariencia visual del programa").

Sin embargo, los componentes Swing permiten al programador especificar una apariencia visual distinta para cada plataforma, una apariencia visual uniforme entre todas las plataformas, o incluso puede cambiar la apariencia visual mientras el programa se ejecuta.

A los componentes AWT que se enlazan a la plataforma local se les conoce como componentes pesados (dependen del sistema de ventanas de la plataforma local para determinar su funcionalidad y su apariencia visual). Cada componente pesado tiene un componente asociado (del paquete java.awt.peer), el cual es responsable de las interacciones entre el componente pesado y la plataforma local para mostrarlo y manipularlo.

Varios componentes Swing siguen siendo pesados. En particular las subclases de java.awt.Window, que muestran ventanas en la pantalla, aun requieren de una interacción directa con el sistema de ventanas local (en consecuencia, los componentes pesados de Swing son menos flexibles).

Otras ventajas de Swing respecto a AWT son:

- Amplia variedad de componentes: En general las clases que comiencen por "J" son componentes que se pueden añadir a la aplicación. Por ejemplo: JButton.

- Aspecto modificable (look and feel): Se puede personalizar el aspecto de las interfaces o utilizar varios aspectos que existen por defecto (Metal Max, Basic Motif, Window Win32).

- Arquitectura Modelo-Vista-Controlador: Esta arquitectura da lugar a todo un enfoque de desarrollo, muy arraigado en los entornos gráficos de usuario, realizados con técnicas orientadas a objetos. Cada componente tiene asociado una clase de modelo de datos y una interfaz que utiliza. Se puede crear un

modelo de datos personalizado para cada componente, con sólo heredar de la clase Model.

- Gestión mejorada de la entrada del usuario: Se pueden gestionar combinaciones de teclas en un objeto KeyStroke y registrarlo como componente. El evento se activará cuando se pulse dicha combinación si está siendo utilizado el componente, la ventana en que se encuentra, o algún hijo del componente.

- Objetos de acción (action objects): Estos objetos cuando están activados (enabled) controlan las acciones de varios objetos componentes de la interfaz. Son hijos de ActionListener.

- Contenedores anidados: Cualquier componente puede estar anidado en otro. Por ejemplo, un gráfico se puede anidar en una lista.

- Escritorios virtuales: Se pueden crear escritorios virtuales o "interfaz de múltiples documentos" mediante las clases JDesktopPane y JInternalFrame.

- Bordes complejos: Los componentes pueden presentar nuevos tipos de bordes. Además el usuario puede crear tipos de bordes personalizados.

- Diálogos personalizados: Se pueden crear multitud de formas de mensajes y opciones de diálogo con el usuario, mediante la clase JOptionPane.

- Clases para diálogos habituales: Se puede utilizar JFileChooser para elegir un fichero, y JColorChooser para elegir un color.

- Componentes para tablas y árboles de datos: Mediante las clases JTable y JTree.

- Potentes manipuladores de texto: Además de campos y áreas de texto, se presentan campos de sintaxis oculta JPassword, y texto con múltiples fuentes JTextPane. Además hay paquetes para utilizar ficheros en formato HTML o RTF.-Capacidad para "deshacer". En gran variedad de situaciones se pueden deshacer las modificaciones que se realizaron.