Curso de Java

Programación Orientada a Objetos

Prof. Ing. Guido Acosta


Contenido de la clase

-Relaciones entre las clases

-Excepciones


Relaciones

-Las relaciones existentes entre las distintas clases nos indican como se comunican los objetos de estas clases entre sí.

-Un conjunto de objetos aislados tiene escasa capacidad para resolver un problema.


Tipos de relaciones

-Relaciones de asociación

-Relaciones de uso

-Relaciones de herencia

Relaciones de asociación

Al menos un atributo de la **ClaseB** es una referencia a un objeto de la **ClaseA**.

Relación: duradera


Relaciones de uso

La **ClaseB** necesita de un objeto de la **ClaseA** para llevar a cabo una funcionalidad.

Formas

-En un método de **B** se crea un objeto auxiliar de **A**.

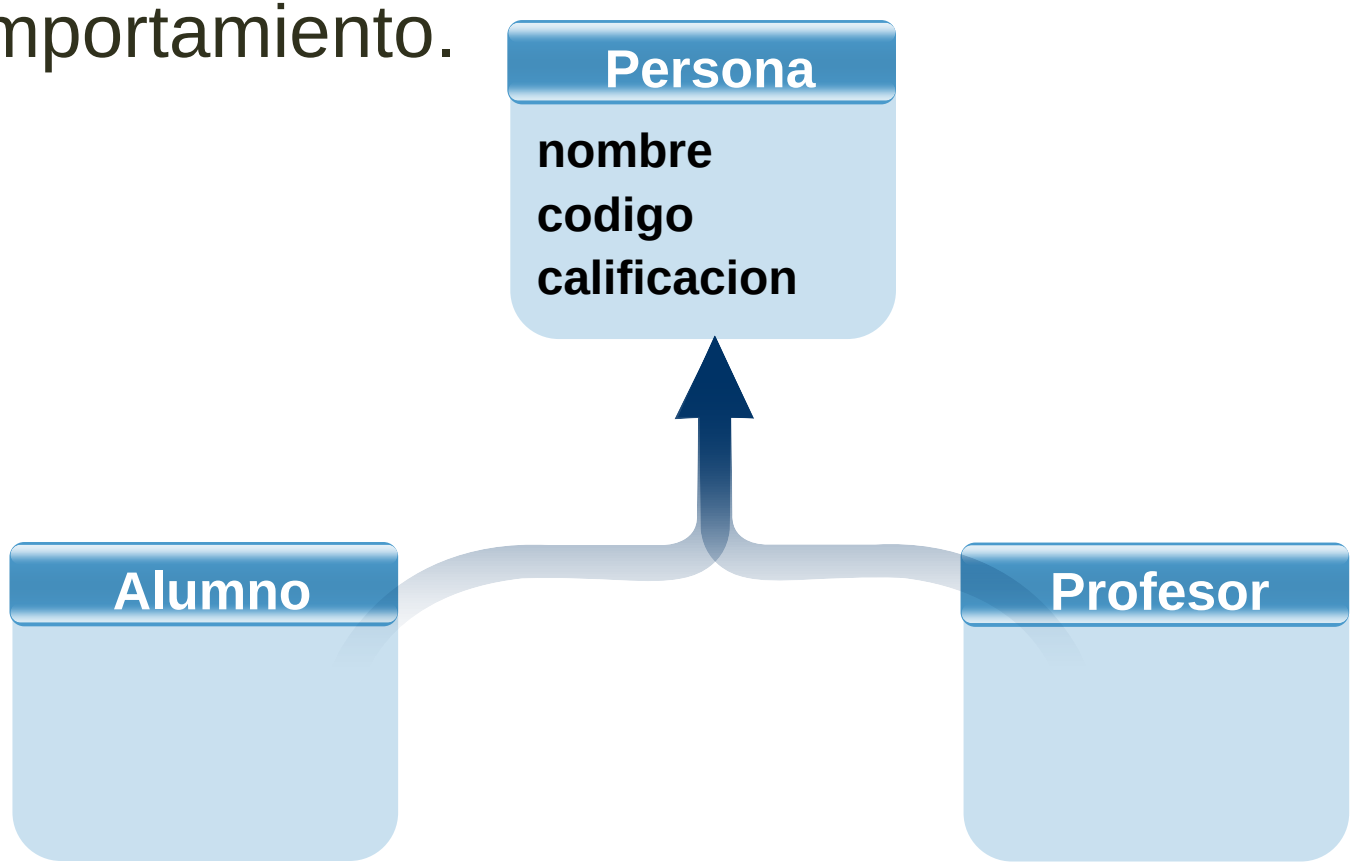
-En un método de **B** aparece un objeto de **A** como argumento.

Relaciones de uso

Relación: temporal

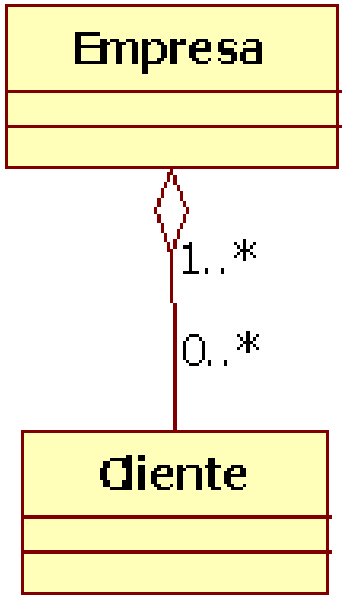
Relaciones de herencia

-Permite que una clase obtenga la funcionalidad de otra, añadiendo nuevos atributos y/o modificando su comportamiento.



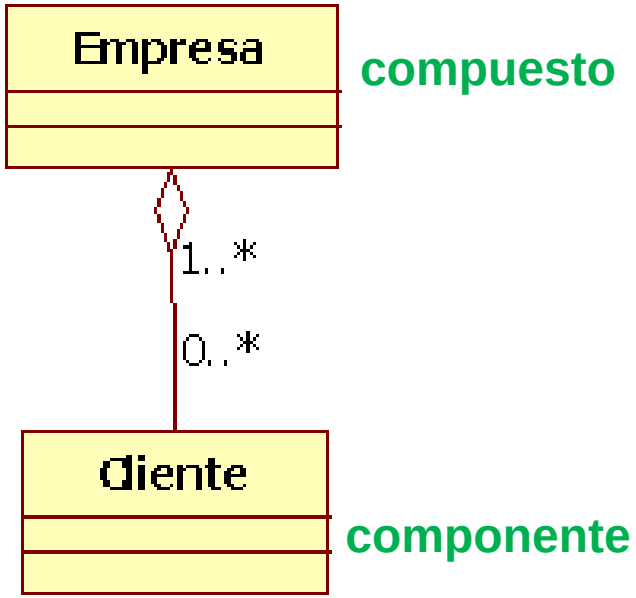
Tipo de asociación – Agregación

-La agregación es un tipo de asociación que indica que una clase es parte de otra clase (composición débil).



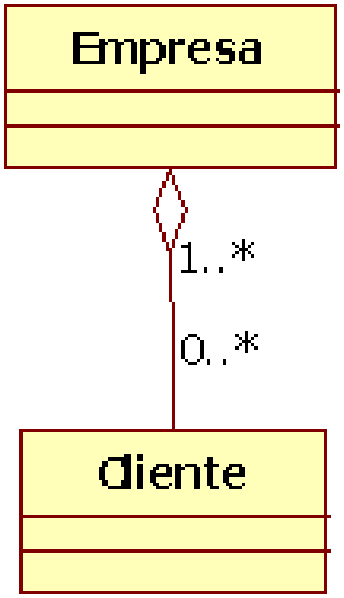
Tipo de asociación – Agregación (2)

-Los componentes pueden ser compartidos por varios compuestos (de la misma asociación de agregación o de varias asociaciones de agregación distintas)



-La destrucción del compuesto no conlleva la destrucción de los componentes.

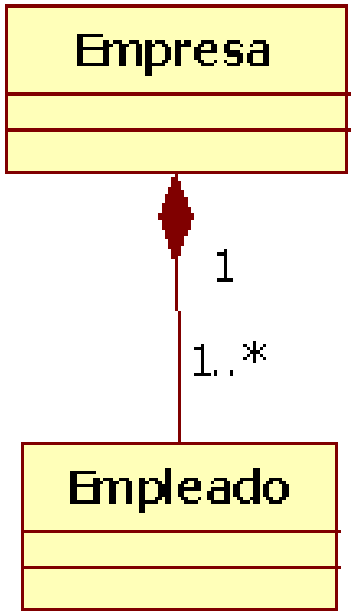
Tipo de asociación – Agregación (3)



- Tenemos una clase **Empresa**.
- Tenemos una clase **Cliente**.
- Una empresa agrupa a varios clientes.

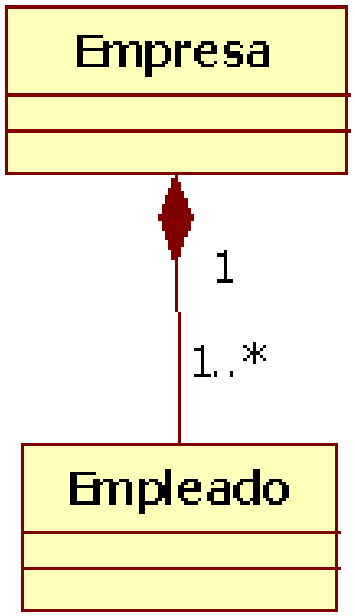
Tipo de asociación – Composición

-La composición es un tipo de asociación que indica que una clase es parte de otra clase (composición fuerte).



Tipo de asociación – Composición(2)

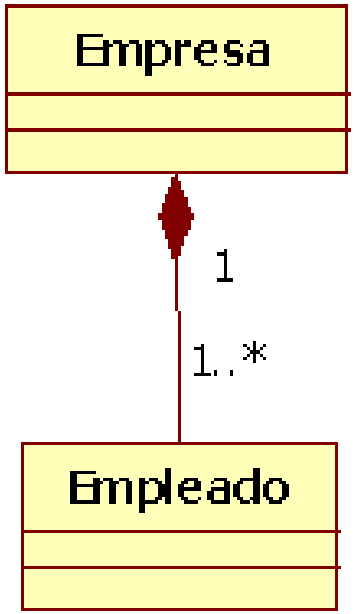
-La vida de la clase contenida debe coincidir con la vida de la clase contenedora.



-Los componentes constituyen una parte del objeto compuesto.

-La supresión del objeto compuesto conlleva la supresión de los componentes.

Tipo de asociación – Composición(2)



-Tenemos una clase **Empresa**.
-Un objeto Empresa está a su vez compuesto por uno o varios objetos del tipo empleado.
-El tiempo de vida de los objetos Empleado depende del tiempo de vida de Empresa, ya que si no existe una Empresa no pueden existir sus empleados.

Ejercicio de programación

-Gestión académica – Parte 1


Contenido de la clase

-Relaciones entre las clases

-**Excepciones**

-Sistema de Gestión Académica

Excepción

Una **excepción** es un evento que ocurre durante la ejecución del programa que interrumpe el flujo normal de las sentencias.

Si hay un error, la aplicación no debería morir. Se debería lanzar una excepción que el sistema pueda manejar para solucionar el problema.

Excepción

Una **excepción** es un evento que ocurre durante la ejecución del programa que interrumpe el flujo normal de las sentencias.

Si hay un error, la aplicación no debería morir. Se debería lanzar una excepción que el sistema pueda manejar para solucionar el problema.

Robustez


Excepción - Ejemplos

-El programa intenta acceder a los miembros de un objeto inexistente.

-El usuario escribe una palabra cuando se esperaba un número.

-El programa intenta leer un archivo que no existe.

-El programa intenta realizar una división por cero.

Excepción

-Cuando se prevé el posible lanzamiento de una excepción y se toman las medidas al respecto en el código de nuestra aplicación, se captura la excepción

```
try {
        System.out.println("Ingrese un nro:");
        numero = Integer.parseInt(lector.readLine());
} catch (Exception e) {
    // TODO: handle exception
}
```

-El manejo de una excepción consiste en capturar una excepción y tomar medidas adecuadas al respecto.

Excepción

Programa

```
leerFichero {
        abrir el fichero;
        determinar su tamaño;
        asignar suficiente memoria;
        leer el fichero a la memoria;
        cerrar el fichero;
}
```


Excepción

Situaciones no manejadas

¿Qué sucede si no se puede abrir el fichero?
¿Qué sucede si no se puede determinar la longitud del fichero?
¿Qué sucede si no hay suficiente memoria libre?
¿Qué sucede si la lectura falla?
¿Qué sucede si no se puede cerrar el fichero?


Excepción

Manejo tradicional de errores

```
codigodeError leerFichero {
    inicializar codigodeError = 0;
    abrir el fichero;
        if (ficheroAbierto) {
            determinar la longitud del fichero;
            if (obtenerLongitudDelFichero) {
                asignar suficiente memoria;
                if (obtenerSuficienteMemoria) {
                    leer el fichero a memoria;
                        if (falloDeLectura) {
                            codigodeError = -1;
                        }
                } else {
                        codigodeError = -2;
            }  else {
                codigodeError = -3;
…
```

Excepción

Manejo en Java

```
leerFichero {
    try {
        abrir el fichero;
        determinar su tamaño;
        asignar suficiente memoria;
        leer el fichero a la memoria;
        cerrar el fichero;
    } catch (falloAbrirFichero) {
            hacerAlgo;
    } catch (falloDeterminacionTamaño) {
            hacerAlgo;
    } catch (falloAsignaciondeMemoria) {
            hacerAlgo;
    } catch (falloLectura) {
            hacerAlgo;
    } catch (falloCerrarFichero) {
            hacerAlgo;
    } }…
```


Bloque try…catch

```
// Bloque 1
try{
    // Bloque 2
} catch (Exception error) {
    // Bloque 3
}
// Bloque 4
```

try…catch – Sin excepción

```
// Bloque 1
try{
    // Bloque 2
} catch (Exception error) {
    // Bloque 3
}
// Bloque 4
```

try…catch – Sin excepción

```
// Bloque 1
try{
    // Bloque 2
} catch (Exception error) {
    // Bloque 3
}
// Bloque 4
```

try…catch – Sin excepción

```
// Bloque 1
try{
    // Bloque 2
} catch (Exception error) {
    // Bloque 3
}
// Bloque 4
```

try…catch – Con excepción en bloque 2

```
// Bloque 1
try{
    // Bloque 2
} catch (Exception error) {
    // Bloque 3
}
// Bloque 4
```

try…catch – Con excepción en bloque 2

```
// Bloque 1
try{
    // Bloque 2
} catch (Exception error) {
    // Bloque 3
}
// Bloque 4
```

try…catch – Con excepción en bloque 2

```
// Bloque 1
try{
    // Bloque 2
} catch (Exception error) {
    // Bloque 3
}
// Bloque 4
```

try…catch – Con excepción en bloque 2

```
// Bloque 1
try{
    // Bloque 2
} catch (Exception error) {
    // Bloque 3
}
// Bloque 4
```

try…catch – Con excepción en bloque 1

```
// Bloque 1
try{
    // Bloque 2
} catch (Exception error) {
    // Bloque 3
}
// Bloque 4
```

try…catch – Con excepción en bloque 1

```
// Bloque 1
try{
    // Bloque 2
} catch (Exception error) {
    // Bloque 3
}
// Bloque 4
```

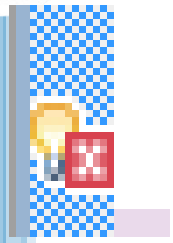

Detectar varias excepciones

```
// Bloque 1
try{
    // Bloque 2
} catch (NumberFormatException e2) {
    // Bloque 3
} catch (IOException e1) {
    // Bloque 4
} catch (Exception error) {
    // Bloque 5
} finally {
 // Bloque 6
}
// Bloque 7
```

Excepciones en Eclipse

```
System.out.println("Ingrese un nro:");
numero = Integer.parseInt(lector.readLine());
```

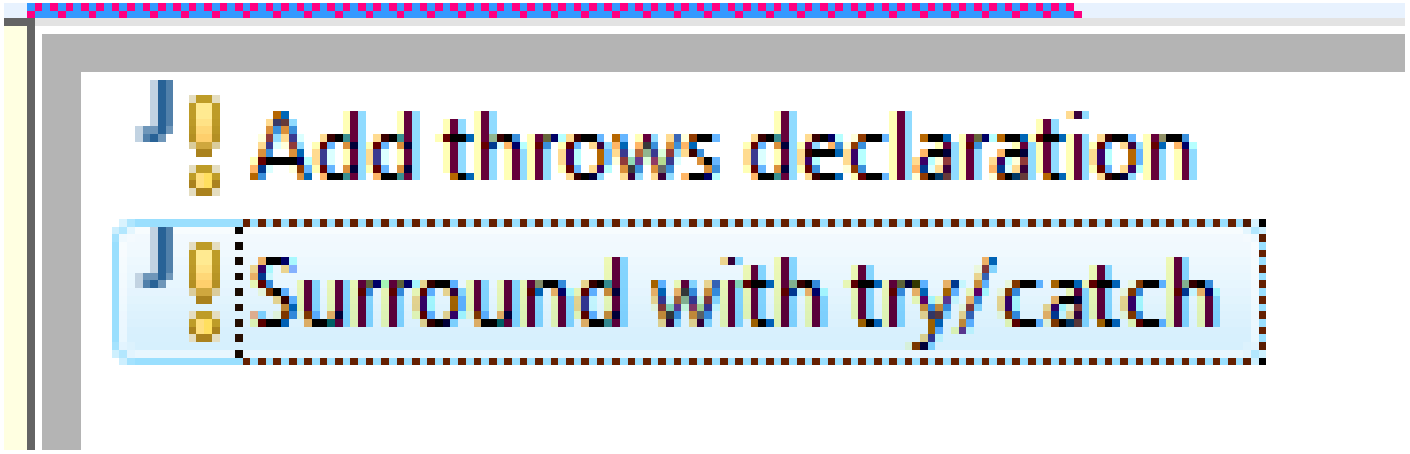
Excepciones en Eclipse



```
System.out.
```

Unhandled exception type IOExceptionn

Excepciones en Eclipse



Surround with try/catch

```
try {
    numero = Integer.parseInt(lector.readLine());
} catch (NumberFormatException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Add throws declaration

```
public static void main(String[] args) throws
        NumberFormatException, IOException{


    System.out.println("Ingrese un nro:");
    numero = Integer.parseInt(lector.readLine());
```

Propagación de excepciones