



Manejo de excepciones

Qué es una excepción?

- Es la indicación de una “situación anormal”.
- Se produce cuando un método NO termina de la manera esperada, sino que termina de manera “excepcional” como consecuencia de un ERROR.
- Si se produce un ERROR y NO es tratado adecuadamente, el programa termina su ejecución.

Cómo se trata una excepción?

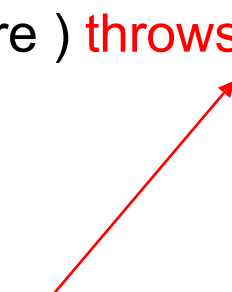
- ❶ Cuando se detecta una situación anormal, se debe **disparar** (anunciar) una excepción
- ❷ Se “atrapa” la excepción antes de que el programa deje de funcionar
- ❸ Se ejecuta una acción para:
 - 👍 Recuperar el programa del error
 - 👍 Pasar el control a otro método para que la procese
 - 👍 Informar al usuario qué fue lo que pasó

Cómo se trata una excepción?

- ❶ Cuando se detecta una situación anormal, se debe **disparar** (anunciar) una excepción

throws Exception

```
public void afiliarSocio (String cedula, String nombre ) throws Exception  
{  
....  
}
```



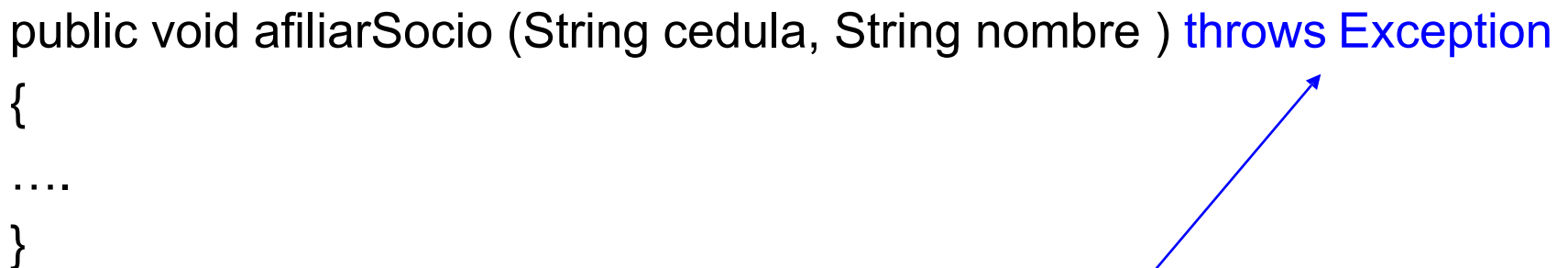
Se declara en la signature del método y quiere decir que es posible que este método lance una excepción en caso de detectar un error

Cómo se trata una excepción?

- ❶ Cuando se detecta una situación anormal, se debe **disparar** (anunciar) una excepción

throws Exception

```
public void afiliarSocio (String cedula, String nombre ) throws Exception  
{  
....  
}
```



Esto también quiere decir que es posible que este método no genere la excepción, sino que simplemente la deje pasar

Cómo se trata una excepción?


```
public void afiliarSocio (String cedula, String nombre ) throws Exception
{
    Socio s = buscarSocio( cedula );
    if ( s == null )
    {
        Socio nuevoSocio = new Socio (cedula, nombre );
        socios.add( nuevoSocio );
    }
    else
    {
        Exception e = new Exception ( “El socio ya existe” );
        throw e;
    }
}
```

← Revisar que no haya ya un socio con la misma cédula

Cómo se trata una excepción?

```
public void afiliarSocio (String cedula, String nombre ) throws Exception
{
    Socio s = buscarSocio( cedula );
    if ( s == null )
    {
        Socio nuevoSocio = new Socio (cedula, nombre );
        socios.add( nuevoSocio );
    }
    else
    {
        Exception e = new Exception ( “El socio ya existe” );
        throw e;
    }
}
```


En caso de que el socio NO exista, añadirlo al vector de socios



Cómo se trata una excepción?

```
public void afiliarSocio (String cedula, String nombre ) throws Exception
{
    Socio s = buscarSocio( cedula );
    if ( s == null )
    {
        Socio nuevoSocio = new Socio (cedula, nombre );
        socios.add( nuevoSocio );
    }
    else
    {
        Exception e = new Exception ( "El socio ya existe" );
        throw e;
    }
}
```

En caso de que el socio SI exista, generar una excepción



Cómo se trata una excepción?


- public void afiliarSocio (String cedula, String nombre) **throws Exception**
- {
- Socio s = buscarSocio(cedula);
if (s == null)
- {
 - Socio nuevoSocio = new Socio (cedula, nombre);
 - socios.add(nuevoSocio);
- }
- else**
- {**
- Exception e = new Exception (“El socio ya existe”);**
- throw e;**
- }**
- }**

1) Se crea un objeto de la clase Exception (con new)

Cómo se trata una excepción?

```
• public void afiliarSocio (String cedula, String nombre ) throws Exception
{
    Socio s = buscarSocio( cedula );
    if ( s == null )
    {
        Socio nuevoSocio = new Socio (cedula, nombre );
        socios.add( nuevoSocio );
    }
    else
    {
        • Exception e = new Exception ( “El socio ya existe” );
        • throw e;
    }
}
```

2) Se lanza la excepción (con la instrucción throw)



Cómo se trata una excepción?

- ❶ Cuando se detecta una situación anormal, se debe disparar (anunciar) una excepción
- ❷ Se “atrapa” la excepción antes de que el programa deje de funcionar
- ❸ Se ejecuta una acción para:
 - 👍 Recuperar el programa del error
 - 👍 Pasar el control a otro método para que la procese
 - 👍 Informar al usuario qué fue lo que pasó

Cómo se trata una excepción?

❷ Se “atrapa” la excepción antes de que el programa deje de funcionar

❸ Se ejecuta una acción para:

- 👍 Recuperar el programa del error
- 👍 Pasar el control a otro método para que la procese
- 👍 Informar al usuario qué fue lo que pasó

Cómo se trata una excepción?

② Se “atrapa” la excepción antes de que el programa deje de funcionar

instrucción try- ↑

{ Delimita la porción de código dentro de un método en la que necesitamos desviar la ejecución si una excepción ocurre allí

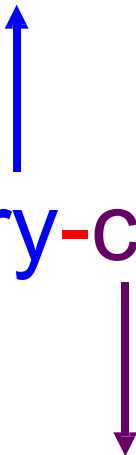
③ Se ejecuta una acción para:

- 👍 Recuperar el programa del error
- 👍 Pasar el control a otro método para que la procese
- 👍 Informar al usuario qué fue lo que pasó

Cómo se trata una excepción?

② Se “atrapa” la excepción antes de que el programa deje de funcionar

instrucción **try-catch** { Define el código que manejará el error



③ Se ejecuta una acción para:

- 👍 Recuperar el programa del error, o al menos
- 👍 Informar al usuario qué fue lo que pasó

Ejemplo en la interfaz

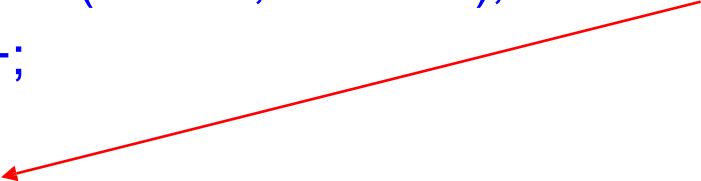
```
public void adicionarNuevoSocio (String cedula, String nombre )
{
    try
    {
        club.afiliarSocio (cedula, nombre ); ←
        totalSocios++;
    }
    catch (Exception x)
    {
        JOptionPane.showMessageDialog( this, "Mensaje", x.getMessage( ),
        JOptionPane.ERROR_MESSAGE);
    }
}
```

Aquí se puede
presentar una
excepción. Si todo
funciona bien, no se
ejecuta NINGUNA
instrucción del bloque
catch

Ejemplo

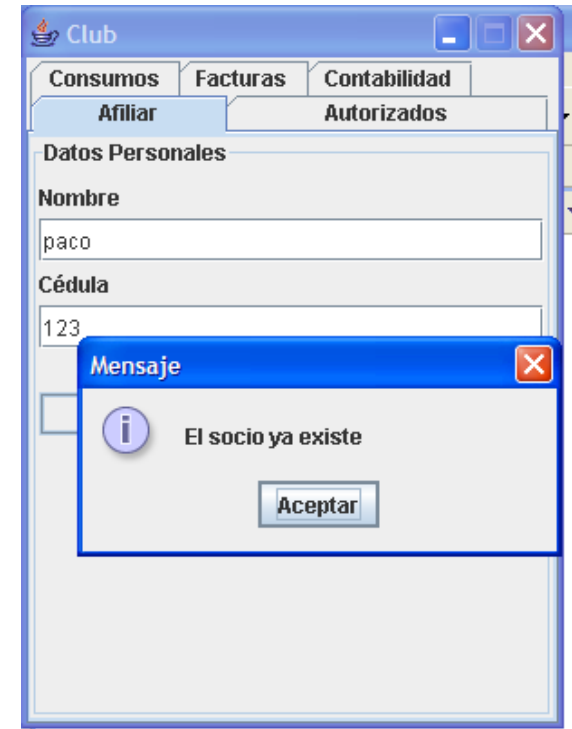
```
public void ejemplo (String cedula, String nombre )
{
    try
    {
        club.afiliarSocio (cedula, nombre );
        totalSocios++;
    }
    catch (Exception x)
    {
        JOptionPane.showMessageDialog( this, "Mensaje", x.getMessage( ),
        JOptionPane.ERROR_MESSAGE);
    }
}
```

Así se dice que cualquier excepción que se atrape la vamos a asignar a la variable de tipo Exception llamada x



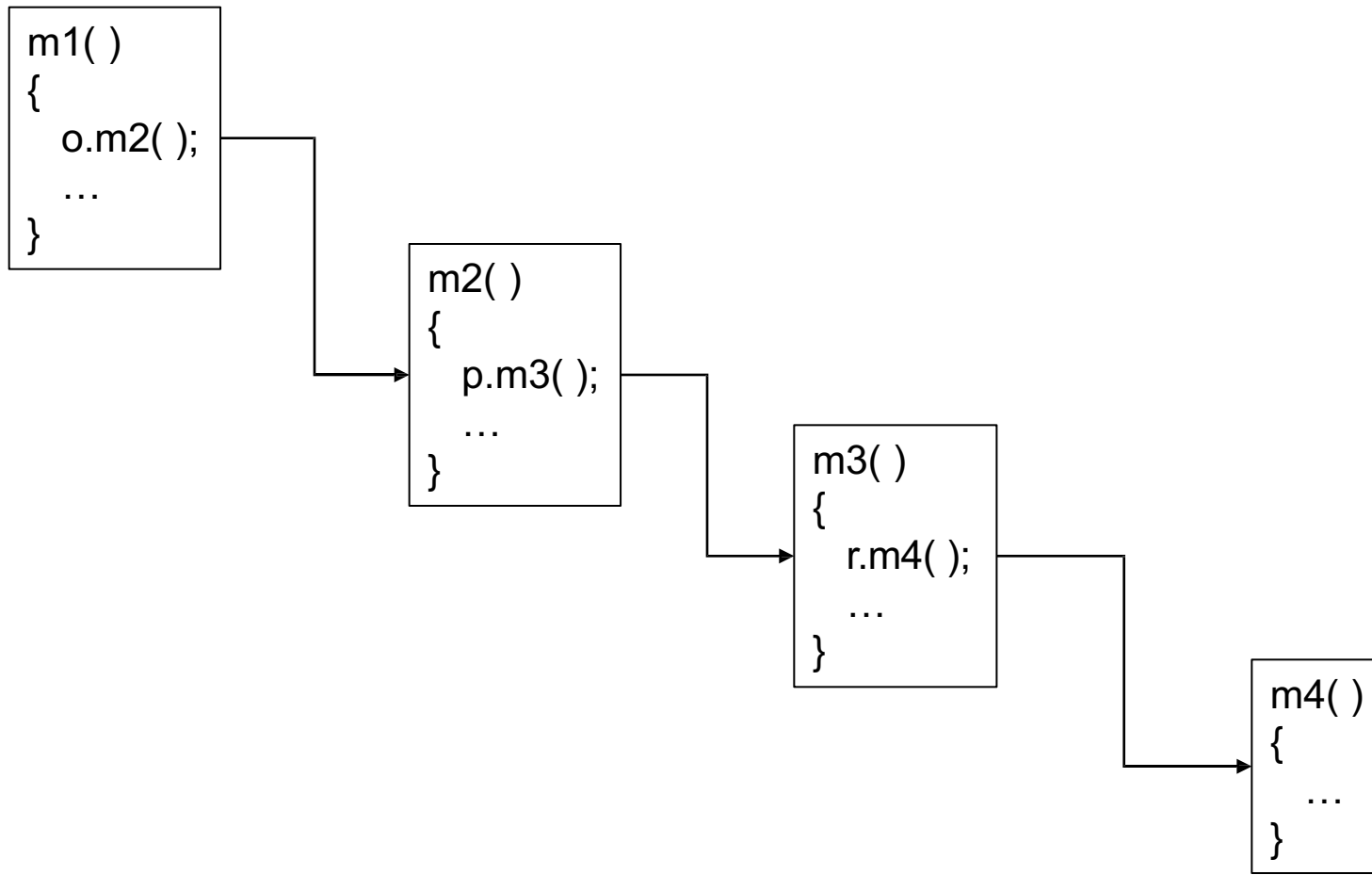
Ejemplo

```
public void ejemplo (String cedula, String nombre )
{
    try
    {
        club.afiliarSocio (cedula, nombre );
        totalSocios++;
    }
    catch (Exception x)
    {
        JOptionPane.showMessageDialog( this, "Mensaje", x.getMessage( ),
        JOptionPane.ERROR_MESSAGE);
    }
}
```

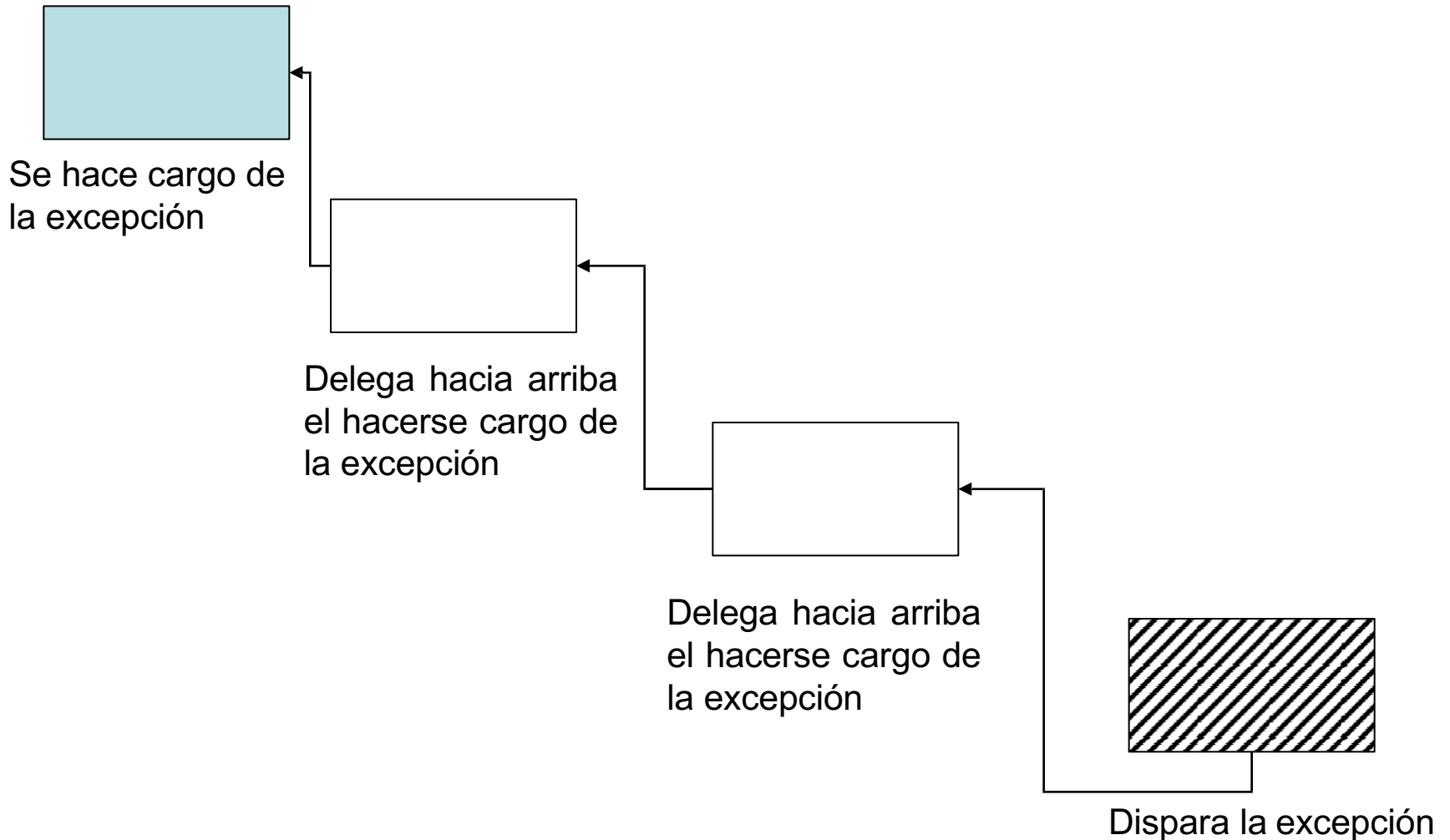


Bloque catch = código que maneja el error. Aquí podemos usar la variable x para pedirle información sobre el error

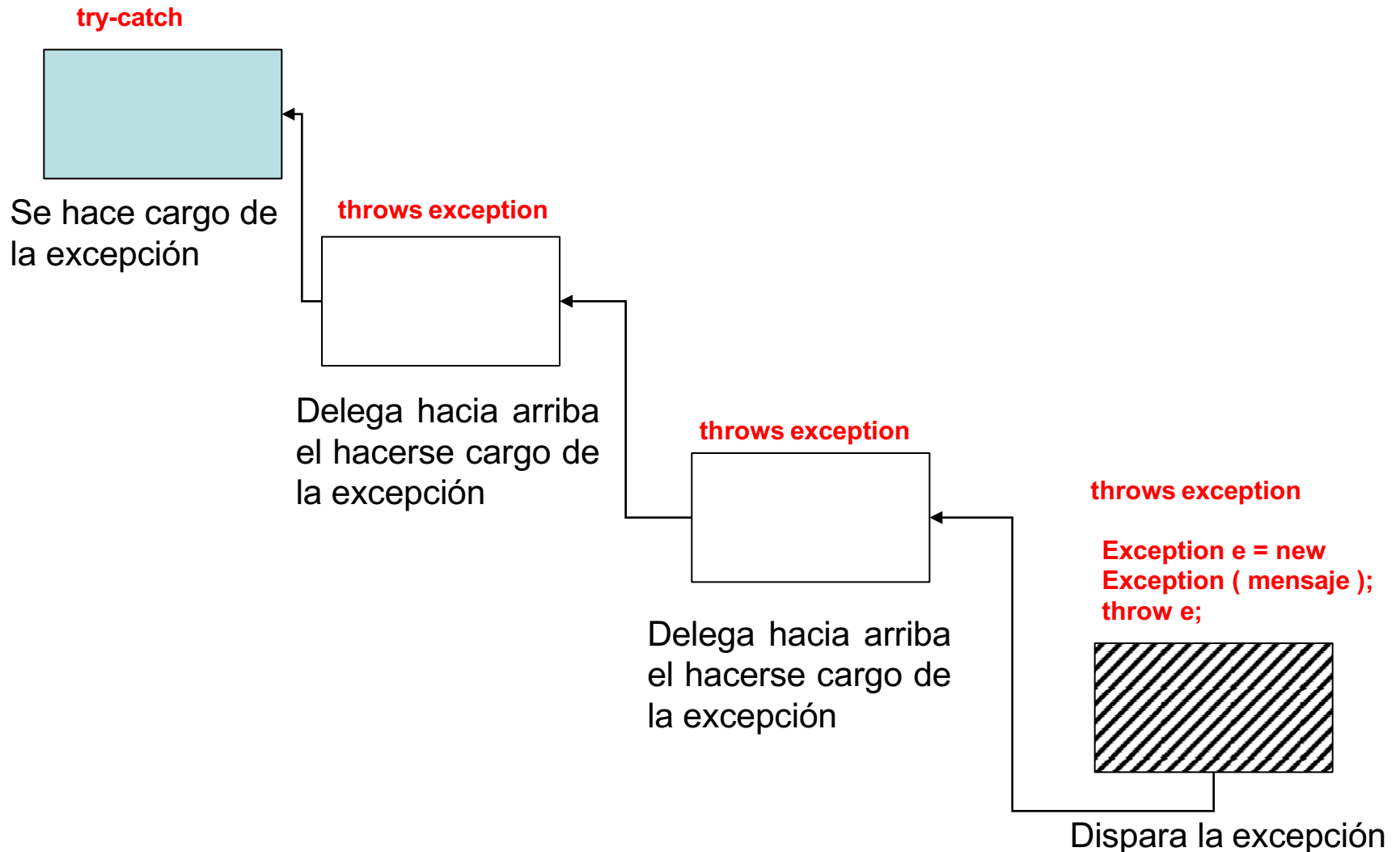
Recuperación de una situación anormal



Recuperación de una situación anormal



Recuperación de una situación anormal



Mas sobre excepciones ...

- Existen diferentes clases de excepciones en java (NO SOLO la clase Exception). Ejemplos:
 - NullPointerException
 - IndexOutOfBoundsException
 - ArithmeticException
 - ...

Mas sobre excepciones ...

- Los métodos de las clases nativas de java disparan excepciones. Ejemplos:

Clase	Método	Excepción
ArrayList	add (indice,objeto)	IndexOutOfBoundsException (si indice no es válido)
String	contentEquals(cadena)	NullPointerException (se cadena es null)

Mas sobre excepciones ...

- Para atrapar y manejar diferentes excepciones en un solo método:
 - 1 try + varios catch (1 por cada clase de excepción).

```
public void ejemplo (String cedula, String nombre )
{
    try
    {
        club.afiliarSocio (cedula, nombre);
        totalSocios++;
    }
    catch (NullPointerException e)
    {
        JOptionPane.showMessageDialog( this, "ERROR DE NULL", e.getMessage( ),
        JOptionPane.ERROR_MESSAGE);
    }
    catch (Exception e)
    {
        JOptionPane.showMessageDialog( this, "ERROR DE DUPLICACION", e.getMessage( ),
        JOptionPane.ERROR_MESSAGE);
    }
}
```

Contrato de un método

Contrato de un método

- Establece:



- Bajo qué condiciones (**suposiciones**) el método tendrá éxito



- Cuál será el **resultado** una vez que termine su ejecución

Ejemplo de contrato de un método

public void afiliarSocio(String cedula, String nombre) throws exception



- Bajo qué condiciones (**suposiciones**) el método tendrá éxito



- La lista de socios ya fue creada (puede ser vacía)
- La cédula no es vacía
- El nombre no es vacío
- No se ha verificado que ya exista un socio con esa cédula

PRECONDICION

Ejemplo de contrato de un método

`public void afiliarSocio(String cedula, String nombre) throws exception`



- Cuál será el **resultado** una vez que termine su ejecución



POSTCONDICION



• Todo funcionó bien y el socio se afilió al club

ó

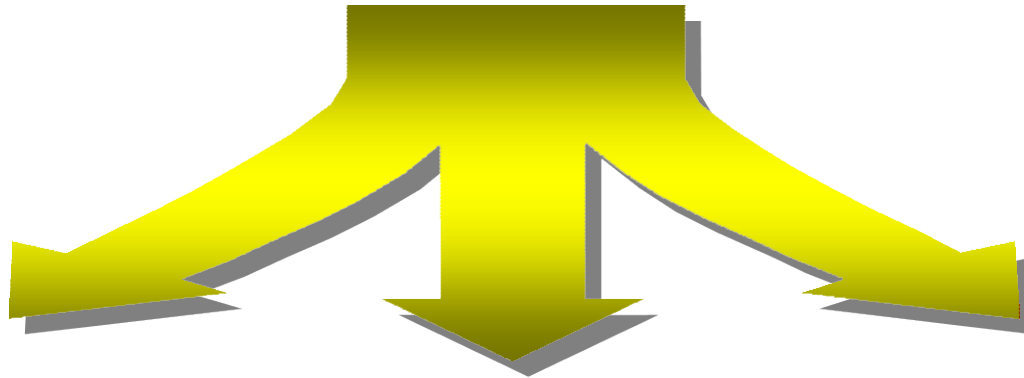
EXCEPCION



• Se produjo un error y se informó el problema con una excepción. El socio no quedó afiliado al club

Precondición

- Conjunto de **suposiciones**, expresadas como **condiciones** que deben ser verdaderas para que el método se ejecute con éxito.
- Las **suposiciones** (**condiciones**) están relacionadas con:



Atributos

Asociaciones

Parámetros
de entrada

Precondición

- Tarea No. 4: identificar precondiciones del método registrarConsumo de la clase Socio

```
public void registrarConsumo( String nombreA, String concepto, double valor )  
{  
    Factura nuevaFactura = new Factura( nombreA, concepto, valor );  
    facturas.add( nuevaFactura );  
}
```

Suposiciones sobre el parámetro nombreA	
Suposiciones sobre el parámetro concepto	
Suposiciones sobre el parámetro valor	
Suposiciones sobre el estado del objeto (valores de los atributos)	
Suposiciones sobre el estado de los objetos con los cuales existe una asociación	

Precondición

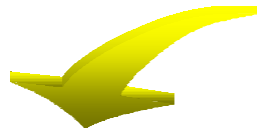
- Tarea No. 4: identificar precondiciones del método registrarConsumo de la clase Socio

```
public void registrarConsumo( String nombreA, String concepto, double valor )
{
    Factura nuevaFactura = new Factura( nombreA, concepto, valor );
    facturas.add( nuevaFactura );
}
```

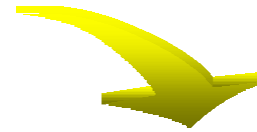
Suposiciones sobre el parámetro nombreA	nombreA != null, nombreA != “ ”, nombreA existe en la lista de autorizados del socio
Suposiciones sobre el parámetro concepto	concepto != null, concepto != “ ”
Suposiciones sobre el parámetro valor	valor >= 0
Suposiciones sobre el estado del objeto (valores de los atributos)	
Suposiciones sobre el estado de los objetos con los cuales existe una asociación	facturas != null

Postcondición

- **Resultado**, expresado como un conjunto de **condiciones**, que deben ser verdaderas después de que el método ha sido ejecutado (SIEMPRE Y CUANDO NO SE HAYA LANZADO UNA EXCEPCIÓN).
- El **resultado** (**condiciones**) están relacionadas con:



Valor de retorno
(si el método
retorna algo)



Estado del objeto (valor
de sus atributos) después
de la ejecución del
método

El contrato queda establecido de la siguiente manera ...

- “Si todas las **condiciones** de la **precondición** se cumplen antes de llamar el método, éste asume el compromiso de llegar a cumplir todas las **condiciones** incluidas en la **postcondición**”



- Si alguna de las **precondiciones** no se cumple, el método no está obligado a cumplir la **postcondición**

Postcondición

- Tarea No. 5: identificar postcondiciones de los métodos registrarConsumo y existeAutorizado de la clase Socio

```
public void registrarConsumo( String nombreA, String concepto, double valor )  
{  
    Factura nuevaFactura = new Factura( nombreA, concepto, valor );  
    facturas.add( nuevaFactura );  
}
```

Descripción del estado del objeto (valores de los atributos) que ejecutó el método, expresada como una lista de condiciones que deben ser verdaderas	
--	--

Postcondición

- Tarea No. 5: identificar postcondiciones de los métodos registrarConsumo y existeAutorizado de la clase Socio

```
public boolean existeAutorizado( String nombreAutorizado )
{
    boolean encontro = false;
    int numAutorizados = autorizados.size();
    for ( int i = 0; i < numAutorizados && !encontro; i++ )
    {
        String a = ( String ) autorizados.get( i );
        if ( a.equals( nombreAutorizado ) )
            encontro = true;
    }
    return encontro;
}
```

Descripción del retorno del método, expresada como una lista de condiciones que deben ser verdaderas	
--	--

Preguntas típicas que surgen en el momento de definir un contrato

- Un método debe verificar en algún punto las condiciones que hacen parte de la precondición?

- Un método debe verificar en algún punto las condiciones que hacen parte de la precondición?
- R// NO. Lo que aparece en la precondición se debe suponer como cierto y se debe utilizar como si lo fuera. Si algo falla en la ejecución por culpa de eso, es responsabilidad de aquel que hizo la llamada al método sin cumplir el contrato.

- Cómo se manejan las excepciones en los contratos?

- Cómo se manejan las excepciones en los contratos?
- R// Un contrato sólo debe decir que lanza una excepción cuando, aún cumpliéndose todo lo pedido en la precondition, es imposible llegar a cumplir la postcondición. Eso quiere decir que ninguna excepción puede asociarse con el no cumplimiento de una precondition.

- Qué se incluye entonces en la precondition?

- Qué se incluye entonces en la precondition?
- R// En la precondition solo se deben incluir condiciones que resulten fáciles de garantizar por parte de aquél que utiliza el método.
- Si se imponen verificaciones excesivamente costosas en tiempo antes de usar el método, terminamos construyendo programas ineficientes.
- Si se quiere asegurar de algo en la ejecución del método, basta con eliminarlo de la precondition y lanzar una excepción si no se cumple.

- Por qué es inconveniente verificar todo dentro de los métodos ?

- Por qué es inconveniente verificar todo dentro de los métodos ?
- R// Por eficiencia. Es mucho mejor repartir las responsabilidades de verificar las cosas entre el que hace el llamado y el que hace el método.
- Si en el contrato queda claro quién se encarga de qué, es más fácil y eficiente resolver los problemas.

Documentación de los
contratos con javadoc

Documentación de los contratos con javadoc

El elemento ...	Se documenta con ...
Contrato completo	<pre>/** (al inicio) * (en cada línea) */ (al final)</pre>
Descripción general del método	<pre>* Comentario (va al comienzo de la descripción del contrato)</pre>

```
/**
 * Este método afilia un nuevo socio al club.<br>
 */
public void afiliarSocio( String cedula, String nombre ) throws Exception
{
}
```

Documentación de los contratos con javadoc

El elemento ...	Se documenta con ...
Contrato completo	<pre>/** (al inicio) * (en cada línea) */ (al final)</pre>
Descripción general del método	<pre>* Comentario (va al comienzo de la descripción del contrato)</pre>
Precondición	<pre>* pre: Frase con la precondición.
</pre>

```
/**
 * Este método afilia un nuevo socio al club
 * <b>pre:</b> La lista de socios no es null pero puede ser vacía.<br>
 */
public void afiliarSocio( String cedula, String nombre ) throws Exception
{
}
```

Documentación de los contratos con javadoc

El elemento ...	Se documenta con ...
Contrato completo	<code>/**</code> (al inicio) <code>*</code> (en cada línea) <code>*/</code> (al final)
Descripción general del método	<code>* Comentario</code> (va al comienzo de la descripción del contrato)
Precondición	<code>* pre:</code> Frase con la precondición.
Poscondición	<code>* post:</code> Frase con la poscondición.

```
/**
 * Este método afilia un nuevo socio al club
 * <b>pre:</b> La lista de socios no es null pero puede ser vacía.<br>
 * <b>pos:</b> Crea un nuevo socio en el club con los datos dados.<br>
 */
public void afiliarSocio( String cedula, String nombre ) throws Exception
{
}
```

Documentación de los contratos con javadoc

El elemento ...	Se documenta con ...
Contrato completo	/** (al inicio) * (en cada línea) */ (al final)
Descripción general del método	* Comentario (va al comienzo de la descripción del contrato)
Precondición	* pre: Frase con la precondición.
Poscondición	* post: Frase con la poscondición.
Parámetros de entrada	@param nombre del parámetro, descripción, suposiciones que hace el método sobre el valor del parámetro

```
/**
 * Este método afilia un nuevo socio al club
 * <b>pre:</b> La lista de socios no es null pero puede ser vacía.<br>
 * <b>pos:</b> Crea un nuevo socio en el club con los datos dados.<br>
 * @param cedula Es la cédula del nuevo socio. No se sabe si ya existe un socio en
 * el club con esta cédula. Se tiene que: cedula != null
 * @param nombre Es el nombre del nuevo socio. Se tiene que: nombre != null
 * /
public void afiliarSocio( String cedula, String nombre ) throws Exception
{
}
```


Documentación de los contratos con javadoc

El elemento ...	Se documenta con ...
Contrato completo	/** (al inicio) * (en cada línea) */ (al final)
Descripción general del método	* Comentario (va al comienzo de la descripción del contrato)
Precondición	* pre: Frase con la precondición.
Poscondición	* post: Frase con la poscondición.
Parámetros de entrada	@param nombre del parámetro, descripción, suposiciones que hace el método sobre el valor del parámetro
Retorno del método (si el método devuelve algún valor (no es void))	@return descripción del valor de retorno, condiciones que cumple este valor

```
/**
 * Este método afilia un nuevo socio al club
 * <b>pre:</b> La lista de socios no es null pero puede ser vacía.<br>
 * <b>pos:</b> Crea un nuevo socio en el club con los datos dados.<br>
 * @param cedula Es la cédula del nuevo socio. No se sabe si ya existe un socio en el club con esta cédula. Se tiene
 *
 *         que: cedula != null
 * @param nombre Es el nombre del nuevo socio. Se tiene que: nombre != null
 */
public void afiliarSocio( String cedula, String nombre ) throws Exception
{
}
```

Documentación de los contratos con javadoc

El elemento ...	Se documenta con ...
Contrato completo	<code>/**</code> (al inicio) <code>*</code> (en cada línea) <code>*/</code> (al final)
Descripción general del método	<code>* Comentario</code> (va al comienzo de la descripción del contrato)
Precondición	<code>* pre: Frase con la precondición.
</code>
Poscondición	<code>* post: Frase con la poscondición.
</code>
Parámetros de entrada	<code>@param</code> nombre del parámetro, descripción, suposiciones que hace el método sobre el valor del parámetro
Retorno del método (si el método devuelve algún valor (no es void))	<code>@return</code> descripción del valor de retorno, condiciones que cumple este valor
Excepciones	<code>@throws</code> tipo de la excepción, descripción de cuando puede ser disparada

```
/**
 * Este método afilia un nuevo socio al club
 * <b>pre:</b> La lista de socios no es null pero puede ser vacía.<br>
 * <b>pos:</b> Crea un nuevo socio en el club con los datos dados.<br>
 * @param cedula Es la cédula del nuevo socio. No se sabe si ya existe un socio en el club con esta cédula. Se tiene que:
 *
 *         cedula != null
 * @param nombre Es el nombre del nuevo socio. Se tiene que: nombre != null
 * @throws Exception si el socio ya existía dispara una excepción indicando que el socio ya
 *         existe.
 */
public void afiliarSocio( String cedula, String nombre ) throws Exception
{
}
```

Documentación de los contratos con javadoc

```
/**
 * Este método afilia un nuevo socio al club.<br>
 * <b>pre:</b> La lista de socios no es null pero puede ser vacía.<br>
 * <b>pos:</b> Crea un nuevo socio en el club con los datos dados.<br>
 * @param cedula Es la cédula del nuevo socio. No se sabe si ya existe un socio en
 *               el club con esta cédula. Se tiene que: cedula != null
 * @param nombre Es el nombre del nuevo socio. Se tiene que: nombre != null
 * @throws Exception si el socio ya existía dispara una exception indicando que el
 *               socio ya existe.
 * /
public void afiliarSocio( String cedula, String nombre ) throws Exception
{
}
}
```

Cómo generar y leer el archivo en formato html

- ❶ Ejecutar el archivo **doc.bat** que se encuentra en el directorio **bin** del proyecto (haciendo doble click)

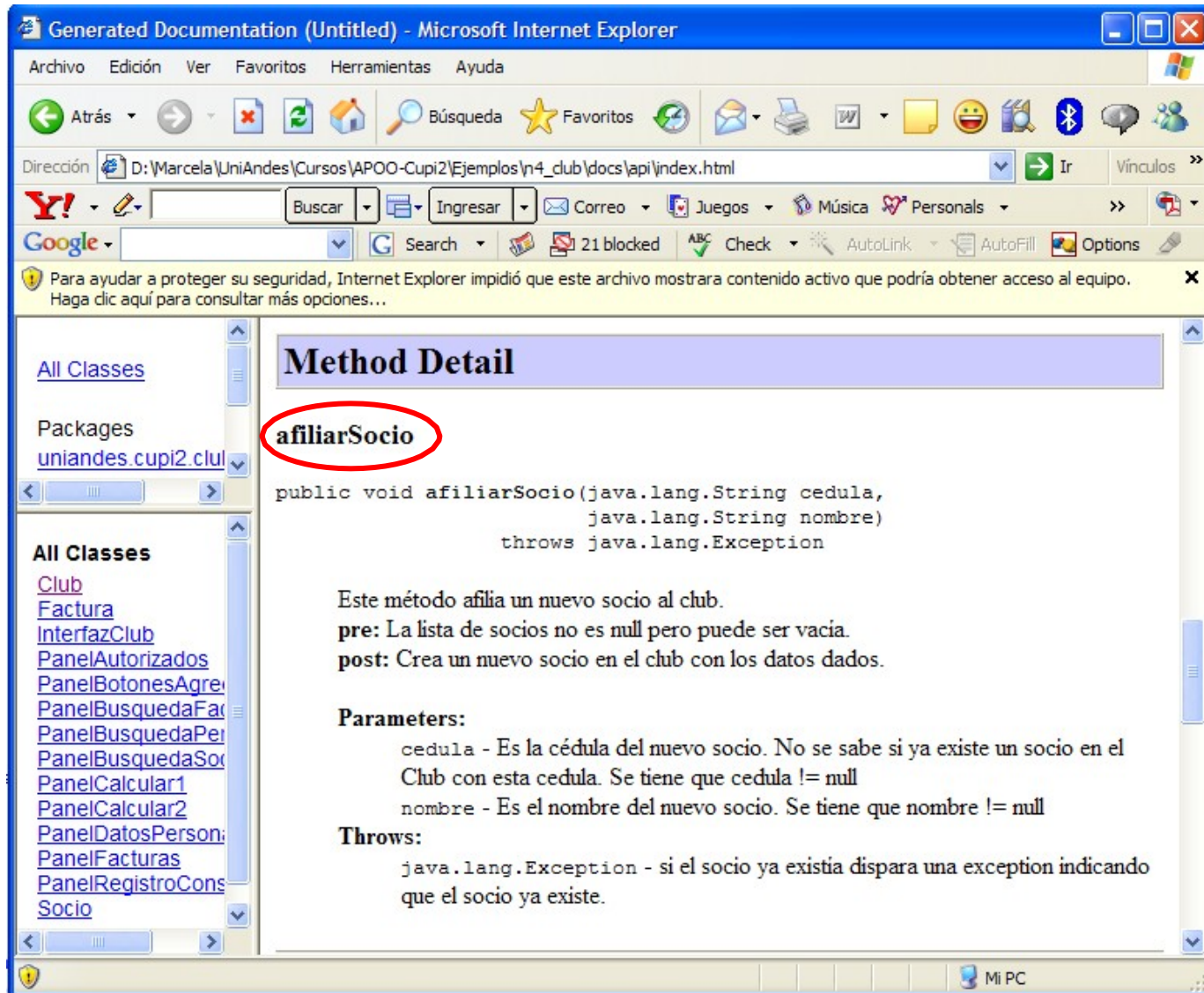
➤ **n4_club/bin/doc.bat**

Este archivo **doc.bat** corre el programa “javadoc” que es el que genera la documentación html

- ❷ Abrir el archivo **index.html** que se encuentra en el directorio **docs/api** del proyecto (haciendo doble click)

➤ **n4_club/docs/api/index.html**

index.html



- Tarea No. 6: Generar la documentación del ejemplo del club y conteste las siguientes preguntas

Qué pasa si el método buscarSocio de la clase Club no encuentra el socio cuya cédula se recibió como parámetro?	
Qué precondition exige el método buscarSocio de la clase Club al atributo que representa la cédula?	
Qué retorna el método darConcepto de la clase Factura? Qué condiciones cumple dicho valor? Qué nombre uso el contrato para representar el valor de retorno?	
Qué sucede si en el método agregarAutorizado de la clase Socio, el parámetro de entrada corresponde al nombre del socio?	
Cuál es la poscondición del método pagarFactura de la clase Socio?	
Cuál es la precondition sobre el parámetro valor en el método registrarConsumo de la clase Socio?	
En cuántos casos lanza una excepción el método agregarAutorizado de la clase Socio?	

Diseño de las firmas de los
métodos

Qué es diseñar un método

❶ Decidir:

- Cuáles serán los **parámetros** del método
- Cuál será su valor de **retorno**
- Qué **excepciones** puede disparar

❷ Precisar su contrato:

- Definir **precondición** y **poscondición**

De dónde viene la información para diseñar la signatura de un método ?



Identificación de
entradas y
salidas de los
requerimientos
funcionales



Tipos de atributos
utilizados en el modelaje
del mundo del problema

De dónde viene la información para diseñar la signatura de un método ?



Identificación de
entradas y
salidas de los
requerimientos
funcionales

Ejemplo

RF: Afiliar un socio

Entradas: cédula del
socio, nombre del socio



- cedula
- nombre

**PARAMETROS
DE
ENTRADA**

Dos preguntas útiles ...



1. Qué información externa al objeto se necesita para resolver el problema que se plantea en el método?

Dos preguntas útiles ...



1. Qué información externa al objeto se necesita para resolver el problema que se plantea en el método?



**PARAMETROS
DE
ENTRADA**

Dos preguntas útiles ...



1. Qué información externa al objeto se necesita para resolver el problema que se plantea en el método?

2. Cómo se modeló esa información dentro del objeto?



**PARAMETROS
DE
ENTRADA**

Dos preguntas útiles ...



1. Qué información externa al objeto se necesita para resolver el problema que se plantea en el método?



**PARAMETROS
DE
ENTRADA**

2. Cómo se modeló esa información dentro del objeto?



**TIPOS DE LOS
PARAMETROS**

- Tarea No. 7: Para la clase Socio, diseñar la signature de los siguientes métodos y justificar las decisiones

existeAutorizado	
eliminarAutorizado	
agregarAutorizado	
pagarFactura	
registrarConsumo	