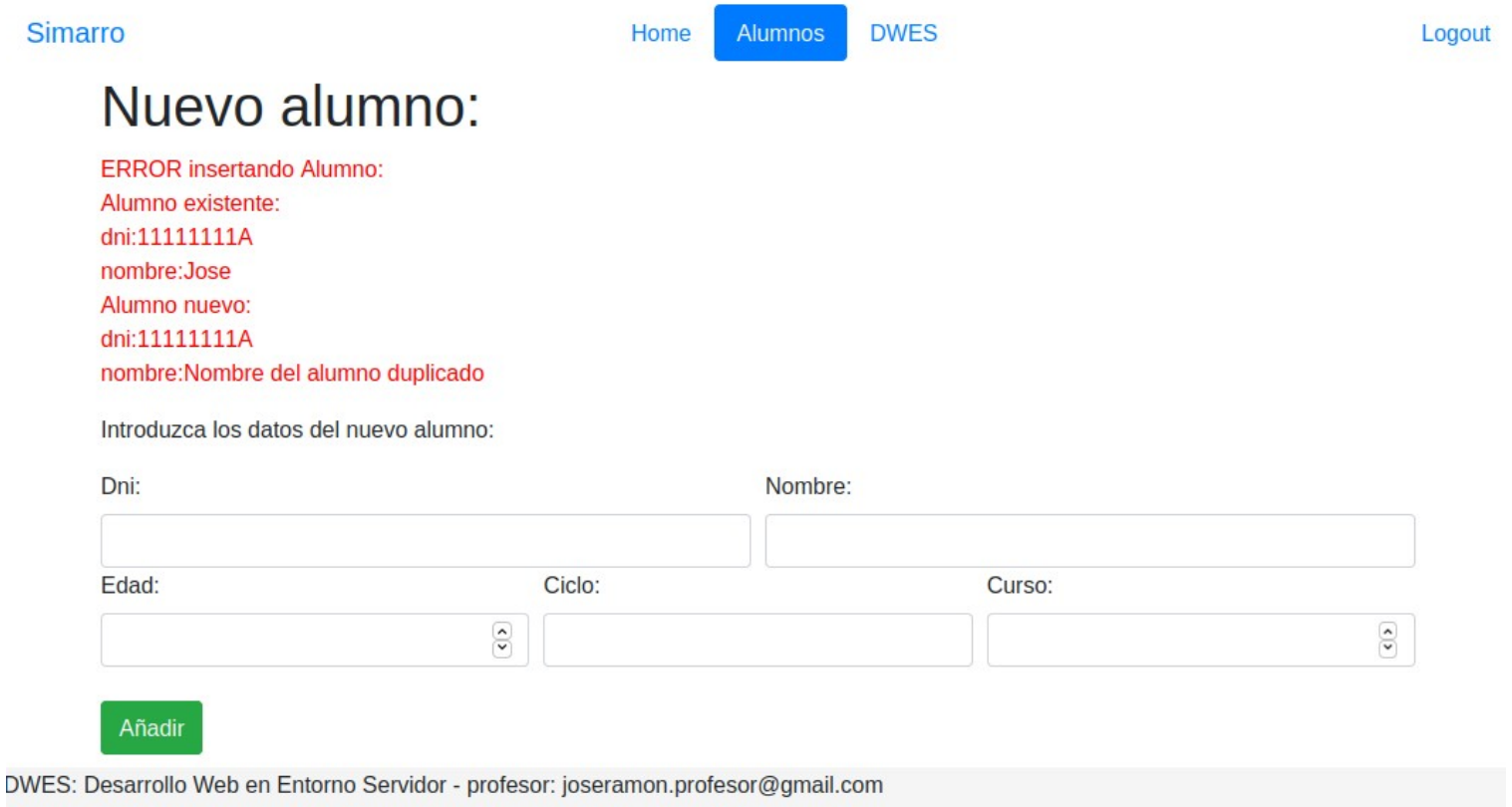
Objetivos de la sesión:

- *Poder comprobar si los **datos están duplicados** para evitar realizar inserciones incorrectas. Para ello se utilizarán **excepciones**.*



¡¡Es importante leer todo el PDF para entender porque hacemos lo que hacemos!!

Una de las mejoras que podemos aplicar a nuestro proyecto es la **comprobación de datos duplicados**. Por ejemplo, Si añadimos a un alumno con el dni “11.111.111A” actualmente nada nos impide volver a introducir otro alumno con el mismo dni.

¿Como podemos incorporar en nuestra aplicación un sistema de comprobación de duplicados? Y sobretodo ¿Donde lo implementamos y como?

Excepciones:

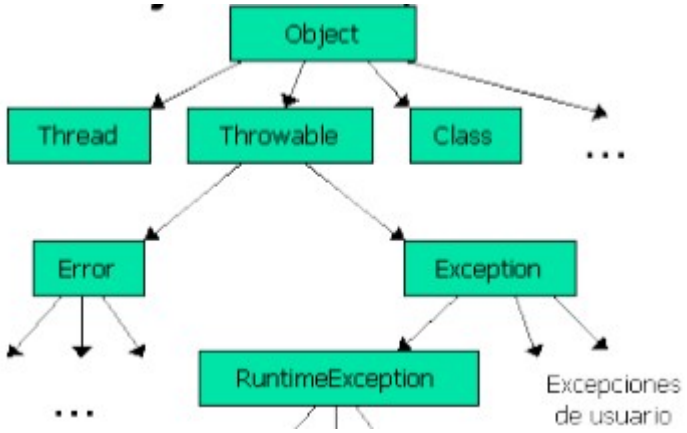
· El sistema que pretendemos implementar consiste en ***generar una excepción cuando intentamos volver a añadir el mismo alumno. Dicha excepción será tratada por el Servlet que solicita el servicio y dicho mensaje de error se deberá mostrar en la página jsp correspondiente***.

Si recuerdas como utilizar las excepciones esta práctica será muy rápida!!

¿Que recuerdas de las excepciones que se vieron en el módulo de “Programación”? ¿Quieres que repasemos los conceptos?

... continuación Excepciones:

- Las **excepciones** son eventos que ocurren durante la ejecución de un programa y hacen que éste salga de su flujo normal de Instrucciones.
- Este mecanismo permite **tratar los errores de una forma elegante**, ya que separa el código para el tratamiento de errores del código normal del programa.
- Se dice que una **excepción es lanzada** cuando se produce un error, y esta **excepción puede ser capturada** para tratar dicho error.
- **Cuando se lanza una excepción, esa excepción es en realidad un objeto.**



... continuación **Excepciones:**

Para manejar un error mediante excepciones se utilizan las sentencias try catch y finally.

La estructura **try-catch-finally**, consta de tres bloques de código:

- **try:** Contiene el código regular de nuestro programa que puede producir una excepción en caso de error.
- **catch:** Contiene el código con el que trataremos el error en caso de producirse.
- **finally:** Contiene el código que se ejecutará al final tanto si se ha producido una excepción como si no lo ha hecho. El bloque finally no es obligatorio ponerlo.

```java
import java.util.InputMismatchException;
import java.util.Scanner;

public class EjemploTryCatchApp {
    Run | Debug
    public static void main(String[] args) {
        Scanner leer = new Scanner(System.in);
        try {
            System.out.print("Introduce un divisor: ");
            int valor = leer.nextInt();
            int auxiliar = 8 / valor;
            System.out.println(auxiliar);
        } catch (ArithmeticException e1) {
            System.out.println("División por cero");
        } catch (InputMismatchException e2) {
            System.out.println("No se ha leído un entero....");
        } catch (Exception e9) {
            System.out.println("Error general");
        } finally {
            leer.nextLine();
        }
        leer.close();
    }
}
```

... continuación **Excepciones:**

```java
import java.util.InputMismatchException;
import java.util.Scanner;

public class EjemploTryCatchApp {
    Run | Debug
    public static void main(String[] args) {
        Scanner leer = new Scanner(System.in);
        try {
            System.out.print("Introduce un divisor: ");
            int valor = leer.nextInt();
            int auxiliar = 8 / valor;
            System.out.println(auxiliar);
        } catch (ArithmeticException e1) {
            System.out.println("División por cero");
        } catch (InputMismatchException e2) {
            System.out.println("No se ha leído un entero....");
        } catch (Exception e9) {
            System.out.println("Error general");
        } finally {
            leer.nextLine();
        }
        leer.close();
    }
}
```

- Cuando se produce una excepción se dejan de ejecutar las siguientes lineas dentro del try y **se entra en un único catch**. Para saber cual, se busca por orden de los catch definidos el primer catch que utiliza el mismo tipo de excepción que se ha producido.
- **Cada clase captura las excepciones de todas sus subclases**.

→ Si especificamos **Exception capturaremos cualquier excepción**, ya que está es la superclase común de todas las excepciones. Por tanto, en caso de capturar una excepción de tipo **Exception debe estar definida en el último catch** o habrá un error de compilación porque el resto de catch ya no son accesibles una vez lo capture el catch de Exception.

→ Si vamos a **tratar todas las excepciones** (sean del tipo que sean) , hemos de pensar si queremos hacer cosas distintas dependiendo del error o con un solo catch de tipo Exception tenemos suficiente para tratarlas todas.

... continuación **Excepciones:**

Un error se puede Capturar o Propagar:

- **Situación 1:Capturar el error: Controlar el error dentro del método :**

  Dentro del método capturamos y tratamos el error con un ***try-catch***.

```java
public static int leerNumero(){
    Scanner leer=new Scanner(System.in);
    int num=0;
    try {
        System.out.print("Inserta un número: ");
        num=leer.nextInt();
        System.out.println("Número correcto");
    } catch (Exception e){
        System.out.println("Solo se pueden introducir números enteros.");
    }
    leer.close();
    return num;
}
```

- **Situación 2. Propagar el error: Controlar el error fuera del método:**

  Si ***la llamada al método está dentro de un bloque try-catch*** podemos controlar el error desde fuera del método.

```java
import java.util.InputMismatchException;
import java.util.Scanner;
public class PropagaError3App {
    public static int leerNumero() throws InputMismatchException{
        Scanner leer=new Scanner(System.in);
        int num=0;
        System.out.print("Inserta un número: ");
        num=leer.nextInt();
        System.out.println("Número correcto");
        leer.close();
        return num;
    }
    Run | Debug
    public static void main(String[] args) {
        try {
            leerNumero();
        } catch (InputMismatchException e){
            System.out.println("Captura de la excepción desde el main.");
        }
    }
}
```

AYUDA: Nuestro servicio generará el error y nuestro servlet lo capturará.

... continuación **Excepciones:**

- En una **aplicación en local** donde el **usuario puede interactuar** con la aplicación **para resolver los errores** puede ser interesante **capturar los errores** en el mismo sitio donde se produzcan **para intentar resolverlos**.

- Sin embargo, los **programadores de BackEnd** (parte servidor) generalmente tienen la costumbre de solo **implementar el funcionamiento correcto en sus aplicaciones y propagar los errores en caso de problemas porque no pueden resolverlos solos.** Pensemos por ejemplo en microservicios con tecnología webservice o REST donde el usuario desde el FrontEnd (pagina web o similar) solicita una operación y realiza la llamada al servidor mediante un petición REST. Si alguno de los datos no es correcto o falla la aplicación simplemente se propaga el error hasta que le llega al usuario, pero no se intenta resolver porque desde el backend no podemos preguntarle al usuario hasta que introduzca un datos correcto. Podemos decirselo al Frontend y si quiere crear la lógica para revisar los errores puede con la información que le llega del backend.

... continuación Excepciones:

Las excepciones pueden utilizarse de muchas formas.

Aunque lancemos una excepción con la instrucción **throw** , eso no significa que no podamos hacerlo dentro de un bloque try...catch y solucionar el error sin salir del método y/o de la clausula try..catch como podemos ver en el ejemplo.

Esta forma de programar solo la veras cuando la app es de escritorio y no esta dividida en Frontend y Backend.

```java
import java.util.InputMismatchException;
import java.util.Scanner;

public class MinutosLanzarYControlarExcepcionApp {
    public static int leerMinutos() throws InputMismatchException{
        Scanner leer = new Scanner(System.in);
        int minutos=0;
        boolean salir=false;
        do {
            try {
                minutos=leer.nextInt();
                if (minutos<0 || minutos>=60){
                    throw new InputMismatchException("Minutos incorrectos.");
                } else {
                    leer.close();
                    salir=true;
                }
            } catch (Exception e) {
                System.out.println(e.getMessage());
                System.out.println("Vuelve a introducir los minutos:");
            }
        } while (!salir);
        return minutos;
    }
    Run | Debug
    public static void main(String[] args) {
        System.out.println("Introduce los minutos:");
        System.out.println("Minutos="+leerMinutos());
    }
}
```

... continuación **Excepciones:**

E incluso podemos tratarlo dentro del método que genera la excepción, pero también fuera a la vez:

En este ejemplo pedimos los minutos, si los introducimos mal dentro del catch nos avisa de que debemos volver a introducirlos, pero si nos equivocamos 3 veces no continuaremos dando oportunidades al usuario y saldremos del método.

```java
import java.util.InputMismatchException;
import java.util.Scanner;

public class MinutosLanzarYControlarExcepcionNvecesApp {
    public static int leerMinutos() throws InputMismatchException{
        Scanner leer = new Scanner(System.in);
        int minutos=0,intentos=0;
        boolean salir=false;
        do {
            try {
                minutos=leer.nextInt();
                if (minutos<0 || minutos>=60){
                    throw new InputMismatchException(++intentos
                    +"º ERROR: Minutos incorrectos.");
                } else {
                    leer.close();
                    salir=true;
                }
            } catch (Exception e) {
                System.out.println(e.getMessage());
                if (intentos<3)
                    System.out.println("Vuelve a introducir los minutos:");
                else
                    throw new InputMismatchException("Minutos incorrectos. "+
                    "Superado el límite de intentos ("+intentos+" intentos)");
            }
        } while (!salir);
        return minutos;
    }
    Run | Debug
    public static void main(String[] args) {
        try {
            System.out.println("Introduce los minutos:");
            System.out.println("Minutos="+leerMinutos());
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

... continuación Excepciones:

Si ninguna de los errores o excepciones predefinidas satisface nuestras necesidades podemos crear ***nuevos errores o excepciones que hereden de Throwable o cualquiera de sus subclases*** de Error o Exception.

```
1   /**
2    * ExcepcionOpcionFueraDeRango
3    */
4   public class ExcepcionOpcionFueraDeRango
5       extends Exception{
6        private static final long serialVersionUID = 1L;
7
8        public ExcepcionOpcionFueraDeRango(String msg) {
9            super(msg);
10       }
11  }
```

Nota : serialVersionUID no es necesario, pero saltará un warning si no lo ponemos porque las excepciones son clases serializables. La serialización nos permite enviar a otro proceso el objeto o guardarlo para su posterior uso, pero requiere del atributo serialVersionUID para hacerlo.

... continuación **Excepciones:**

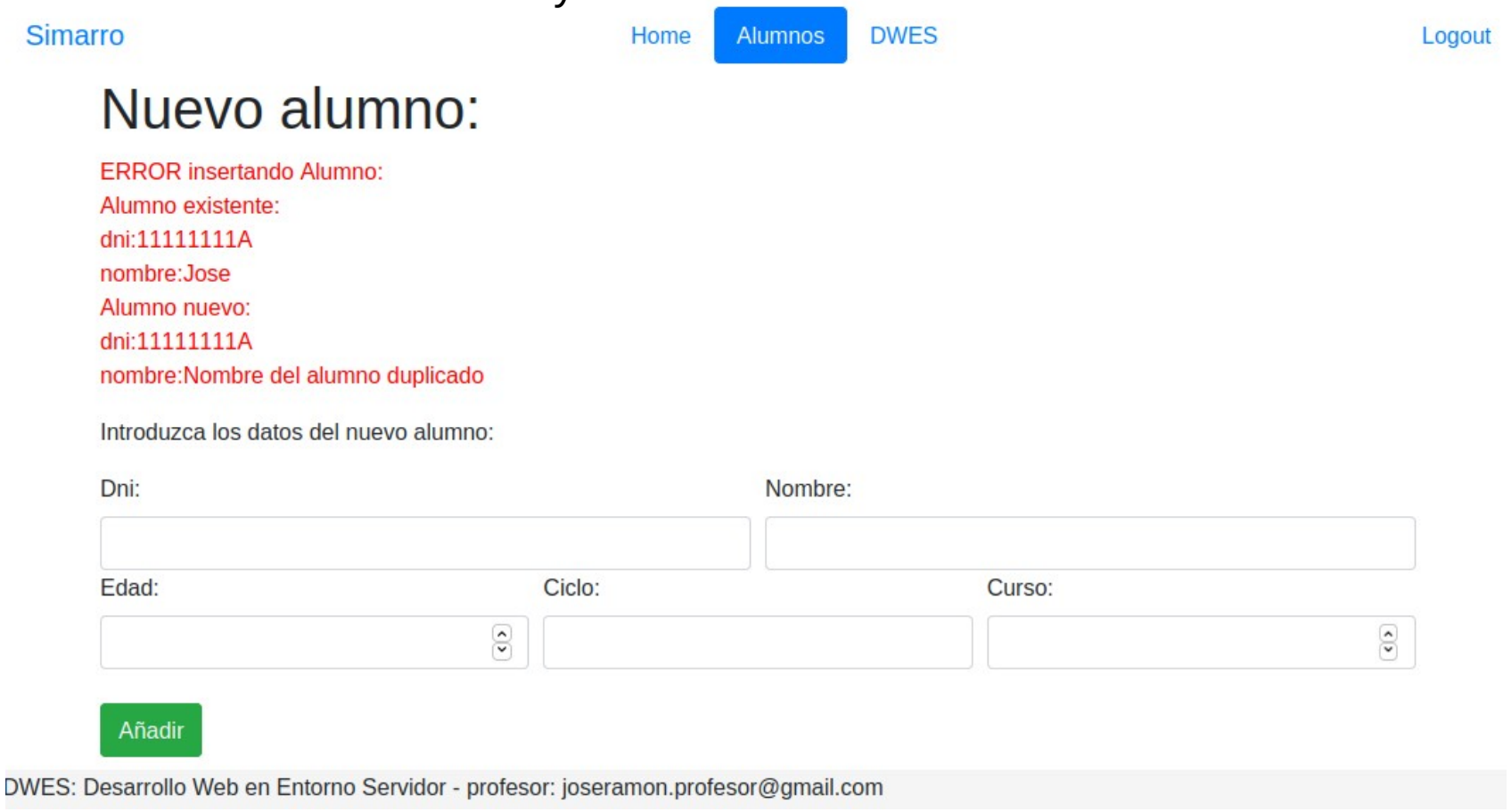
También podemos incluso crear un constructor de la excepción al que le pasemos un objeto procedente de una de nuestras clases en vez de pasarle un String:

```
class AnimalExisteException extends Exception {
    private static final long serialVersionUID = 1L;
    private Animal animal;

    public AnimalExisteException(Animal animal) {
        this.animal = animal;
    }
    @Override
    public String toString(){
        return "El animal identificado con el id '"+
        animal.getId_animal() + "'' ya existe.";
    }
}
```

```
public static Animal nuevoTigre(int id_animal, double peso, double altura) throws AnimalExisteException {
    Tigre t = Tigre.getTigreById(id_animal);// Comprobamos si ya existe
    if (t != null) {// existe
        throw new AnimalExisteException(t);
    } else {// no existe
        return new Tigre(id_animal, peso, altura);
    }
}

Run | Debug
public static void main(String[] args) {
    Animal t;
    try {
        t = nuevoTigre(1, 80, 90);
    } catch (AnimalExisteException e) {
        e.printStackTrace();
    }
```

... continuación Excepciones:

Una vez repasado estos conceptos lo que pretendemos hacer es que la aplicación nos muestre algunos datos del alumno existente y del alumno que se ha intentado insertar erróneamente si el DNI a insertar ya existe :



Para ello tenemos algunas pistas en la siguiente diapositiva!!

... continuación Excepciones:

Hará falta crear 2 métodos en el servicio para poder saber si existe un alumno con ese dni y poder encontrar el alumno ya existente en la lista :

· boolean existeAlumno(Alumno alumno)

· Alumno encontrarAlumnoPorDni(String dni)

Crear una clase AlumnoDuplicadoException que herede de Exception que en su constructor contenga 2 parámetros (el alumno existente en nuestra aplicación y el nuevo alumno que pretendemos insertar).

*El método toString() de la excepción mostrará el mensaje de error (con un mensaje como el de la diapositiva anterior). En el mensaje se tendrá que mostrar el dni y el nombre del alumno existente y del alumno que se pretendía insertar en formato html (con
' s para tener varias lineas).*

ERROR insertando Alumno:
Alumno existente:
dni:11111111A
nombre:Jose
Alumno nuevo:
dni:11111111A
nombre:Nombre del alumno duplicado

En el servicio cuando se intente insertar un nuevo alumno (addAlumno) se deberá de comprobar si ese alumno ya existe y en caso de que exista se deberá de propagar una excepción AlumnoDuplicadoException que capturará el servlet de inserción. El servlet que añade el alumno deberá de almacenar el mensaje de error en un atributo "errores" si el alumno esta duplicado para que la página add-alumno.jsp pueda informar del error.

EJERCICIO:

Sigue todos los pasos de los PDF y sube la aplicación final al moodle.

Para ello:

1º Haz un "Run As \Maven Clean" para dejar solo los fichero fuentes y quitar momentaneamente los necesarios para ejecutar la aplicación (dependencias).

2º Comprime la carpeta de tu aplicación y ponle como nombre al fichero comprimido UD1_practica8_nombreAlumno.tar.gz donde nombreAlumno es el nombre del alumno que entrega la práctica.

3º Súbela al moodle.

IMPORTANTE: No comprimir en RAR, porque Ubuntu no lo lee bien y en clase tenemos Ubuntu. Si tuviesemos Windows, podemos comprimir en ZIP.