

# áreaCIÓN ORIENTADA A OBJETOS

## Excepciones

### Octubre 2018

### Laboratorio 4/6

## OBJETIVOS

1. Perfeccionar el diseño y código de un proyecto considerando casos especiales y errores.
2. Construir clases de excepción encapsulando mensajes.
3. Manejar excepciones considerando los diferentes tipos.
4. Registrar la información de errores que debe conocer el equipo de desarrollo de una aplicación en producción.
5. Vivenciar la prácticas **Designing** - [Simplicity](#).

**Coding** Code must be written to agreed [standards](#)

## ENTREGA

- ➔ Incluyan en un archivo **.zip** los archivos correspondientes al laboratorio. El nombre debe ser los dos apellidos de los miembros del equipo ordenados alfabéticamente.
- ➔ Deben publicar el avance al final de la sesión y la versión definitiva en la fecha indicada, en los espacios preparados para tal fin.

## EQUIPOS

### EN BLUEJ

### PRACTICANDO MDD y BDD con EXCEPCIONES [En lab04.doc, equipos.asta y BlueJ equipos

En este punto vamos a aprender a diseñar, codificar y probar usando excepciones. Para esto se van a trabajar dos métodos de la clase `Equipo` y la excepción `EquipoExcepcion`

1. En su directorio descarguen los archivos contenidos en `equipo.zip`, revisen el contenido y estudien el diseño estructural de la aplicación.
2. Dadas las pruebas, diseñen y codifiquen el método `valorHora`.
3. Dada la especificación, diseñen, codifiquen y prueben el método `valorHoraEstimado`.

## PARA LAS PRUEBAS

Las siguientes personas que tienen valor hora conocido:

```
("Pedro",10000);  
("Santiago",20000);  
("Marcos",30000);  
("Juan",40000);  
("Judas",50000);
```

Las siguientes personas son conocidas pero no tienen valor hora:

```
("Garcia");  
("Ospina");  
("Guarin")
```

## lemois. Innovación Educativa con MOOC en el programa de Ingeniería de Sistemas EN CONSOLA

El objetivo de **lemois** es definir, aplicar y evaluar estrategias para integrar los MOOC al proyecto de formación del programa de ingeniería de sistemas con el propósito de desarrollar en los estudiantes las competencias que les permita aprovecharlos en su **formación permanente** y en su **vinculación a comunidades académicas a escala internacional**.

### Conociendo el proyecto [En lab04.doc]

No olviden respetar los directorios `bin` `docs` `src`

1. En su directorio descarguen los archivos contenidos en **lemois.zip**, revisen el contenido. ¿Cuántos archivos se tienen? ¿Cómo están organizados? ¿Cómo deberían estar organizados?
2. Prepare los directorios necesarios para ejecutar el proyecto. ¿qué estructura debe tener? ¿qué instrucciones debe dar para ejecutarlo?
3. Ejecute el proyecto, ¿qué funcionalidades ofrece? ¿cuáles funcionan?
4. ¿De dónde salen los cursos iniciales? Revisen el código del proyecto. ¿Qué clase pide que se adicionen? ¿Qué clase los adiciona?

### Arquitectura [En lab04.doc, lemois.asta y \*.java]

1. Inicie el diseño con un diagrama de paquetes en el que se presente los componentes y las relaciones entre ellos.
2. Usando ingeniería reversa. Presente el diseño actual de la **capa de aplicación**.
3. Considerando las funcionalidades del sistema. Realicen el diagrama de casos de uso correspondiente.

### Adicionar y listar. Todo OK. [En lab04.doc, lemois.asta y \*.java]

(NO OLVIDEN BDD - MDD)

El objetivo es realizar ingeniería reversa a las funciones de adicionar y listar.

1. Adicionen un nuevo curso

### Machine Learning

Inteligencia Artificial

Coursera

8

Este curso proporciona una amplia introducción al aprendizaje automático, el análisis de datos y el reconocimiento de patrones estadísticos. Los temas incluyen: aprendizaje supervisado, aprendizaje no supervisado y mejores prácticas en aprendizaje automático.

¿Qué ocurre? ¿Cómo lo comprueban? Capturen la pantalla. ¿Es adecuado este comportamiento?

2. Revisen el código asociado a **adicionar** en la capa de presentación y la capa de aplicación. ¿Qué método es responsable en la capa de presentación? ¿Qué método en la capa de aplicación?
3. Realicen ingeniería reversa para la capa de aplicación para **adicionar**. Capturen los resultados de las pruebas de unidad.
4. Revisen el código asociado a **listar** en la capa de presentación y la capa de aplicación. ¿Qué método es responsable en la capa de presentación? ¿Qué método en la capa de aplicación?
5. Realicen ingeniería reversa de la capa de aplicación para **listar**. Capturen los resultados de las pruebas de unidad.
6. Propongan y ejecuten una prueba de aceptación.

## Adicionar un curso. ¿Y si no da un distribuidor? [En lab04.doc, lemois.asta y \*.java]

(NO OLVIDEN BDD - MDD)

El objetivo es perfeccionar la funcionalidad de adicionar un curso.

1. Adicionen el curso *Machine Learning* sin distribuidor. ¿Qué ocurre? ¿Cómo lo comprueban? Capturen la pantalla. ¿Es adecuado este comportamiento?
2. Vamos a evitar la creación de áreas con un distribuidor vacío manejando una excepción `IemoisExcepcion`. Si el curso no tiene distribuidor, no lo creamos y se lo comunicamos al usuario<sup>1</sup>. Para esto lo primero que debemos hacer es crear la nueva clase `IemoisExcepcion` considerando este primer mensaje.
3. Analicen el diseño realizado. ¿Qué método debería lanzar la excepción? ¿Qué métodos deberían propagarla? ¿Qué método debería atenderla? Explique claramente.
4. Construya la solución propuesta. Capturen los resultados de las pruebas.
5. Ejecuten nuevamente la aplicación con el caso de prueba propuesto en 1., ¿Qué sucede ahora? Capture la pantalla.

## Adicionar un curso. ¿Y si ya se encuentra? [En lab04.doc, lemois.asta y \*.java]

(NO OLVIDEN BDD - MDD)

El objetivo es perfeccionar la funcionalidad de adicionar un curso.

1. Adicionen dos veces el nuevo curso *Machine Learning* ¿Qué ocurre? ¿Cómo lo comprueban? Capturen la pantalla. ¿Es adecuado este comportamiento?
2. Analicen el diseño realizado. ¿Qué método debería lanzar la excepción? ¿Qué métodos deberían propagarla? ¿Qué método debería atenderla? Explique claramente.
3. Construya la solución propuesta. Capturen los resultados de las pruebas.
4. Ejecuten nuevamente la aplicación con el caso de prueba propuesto en 1., ¿Qué sucede ahora? Capture la pantalla.

## Adicionar un curso. ¿Y si dan mal las semanas? [En lab04.doc, lemois.asta y \*.java]

(NO OLVIDEN BDD - MDD)

El objetivo es perfeccionar la funcionalidad de adicionar un curso.

1. Adicionen el nuevo curso *Machine Learning* pero en lugar de 8 semanas colquen B semanas. ¿Qué ocurre? ¿Cómo lo comprueban? Capturen la pantalla. ¿Es adecuado este comportamiento?
2. Analicen el diseño realizado. ¿Qué método debería lanzar la excepción? ¿Qué métodos deberían propagarla? ¿Qué método debería atenderla? Explique claramente.
3. Construya la solución propuesta. Capturen los resultados de las pruebas.
4. Ejecuten nuevamente la aplicación con el caso de prueba propuesto en 1., ¿Qué sucede ahora? Capture la pantalla.

## Adicionar un curso. ¿Otras condiciones? [En lab04.doc, lemois.asta y \*.java]

(NO OLVIDEN BDD - MDD)

El objetivo es perfeccionar la funcionalidad de adicionar un curso.

1. Propongan nuevas condiciones para que la adición de un curso sea más robusta.<sup>2</sup>
2. Construya la solución propuesta. (diseño, prueba de unidad, código) Capturen los resultados de las pruebas.

---

<sup>1</sup> Para presentar los mensajes de error al usuario use el método de clase de `JOptionPane` `public static void showMessageDialog(Component parentComponent,`

```
Object message,  
String title,  
int messageType)  
throws HeadlessException
```

Con componente padre:este mensaje: la cadena correspondiente al mensaje de error de la excepcion correspondiente, titulo: ERROR y tipo de mensaje: JOptionPane.ERROR\_MESSAGE

## Consultando por patrones. ¡ No funciona y queda sin funcionar!

[En lab04.doc, lemois.asta y \*.java]

(NO OLVIDEN BDD - MDD)

1. Consulten un curso especial que inicie con M. ¿Qué sucede? ¿Qué creen que pasó? Capturen el resultado. ¿Quién debe conocer y quien NO debe conocer esta información?
2. Exploren el método `registre` de la clase `Registro` ¿Qué servicio presta?
3. Analicen el punto adecuado para que **SIEMPRE**, al sufrir en cualquier punto el sistema un incidente como este, se presente un mensaje especial de alerta al usuario, se guarde la información del error en el registro de error y termine la ejecución. Expliquen y construyan la solución.
4. Ejecuten nuevamente la aplicación con el caso propuesto en 1. ¿Qué mensaje salió en pantalla? ¿La aplicación termina? ¿Qué información tiene el archivo de errores?
5. ¿Es adecuado que la aplicación continúe su ejecución después de sufrir un incidente como este? ¿de qué dependería continuar o parar?
6. Analicen el punto adecuado para que **EN ESTE CASO** se presente un mensaje especial de alerta al usuario, se guarde la información del error en el registro y continúe la ejecución. Expliquen y construyan la solución. No eliminen la solución de 3.
7. Ejecuten nuevamente la aplicación con el caso propuesto en 1. ¿Qué mensaje salió en pantalla? ¿La aplicación termina? ¿Qué información tiene el archivo de errores?

## Consultando por patrones. ¡Ahora si funciona! [En lab04.doc, lemois.asta y \*.java]

(NO OLVIDEN BDD - MDD)

1. Revisen el código asociado a **buscar** en la capa de presentación y la capa de aplicación. ¿Qué método es responsable en la capa de presentación? ¿Qué método es responsable en la capa de aplicación?
2. Realicen ingeniería reversa de la capa de aplicación para **buscar**. Capturen los resultados de las pruebas. Deben fallar.
3. ¿Cuál es el error? Solúcionenlo. Capturen los resultados de las pruebas.
4. Ejecuten la aplicación nuevamente con el caso propuesto. ¿Qué tenemos en pantalla? ¿Qué información tiene el archivo de errores?

## RETROSPECTIVA

1. ¿Cuál fue el tiempo total invertido en el laboratorio por cada uno de ustedes? (Horas/Hombre)
2. ¿Cuál es el estado actual del laboratorio? ¿Por qué?
3. Considerando las prácticas XP del laboratorio. ¿cuál fue la más útil? ¿por qué?
4. ¿Cuál consideran fue el mayor logro? ¿Por qué?
5. ¿Cuál consideran que fue el mayor problema técnico? ¿Qué hicieron para resolverlo?
6. ¿Qué hicieron bien como equipo? ¿Qué se comprometen a hacer para mejorar los resultados?

---

<sup>2</sup>Robustez o solidez. Se refiere a la capacidad del software de defenderse de las acciones anormales que llevan al sistema a un estado no deseado o por lo menos no previsto, causando un comportamiento inesperado, indeseado y posiblemente erróneo