PROGRAMACIÓN ORIENTADA A OBJETOS
Excepciones
2020-2
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.

COMBOS
EN BLUEJ
PRACTICANDO MDD y BDD con EXCEPCIONES
[En lab04.doc, combos.asta y BlueJ combos]

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

1. En su directorio descarguen los archivos contenidos en `combos.zip`, revisen el contenido y estudien el diseño estructural de la aplicación. ¿Cuántas clases tenemos? ¿Cuál sería la clase fachada?
2. Expliquen por qué el proyecto no compila. Realicen las adiciones necesarias para lograrlo.
3. Dadas la documentación, el diseño y las pruebas; codifiquen el método `precio()`
4. Dada la documentación, diseñen, codifiquen y prueben el método `precioOmision()`
5. Dada la documentación, diseñen, codifiquen y prueben el método `precioAsumido()`

IEMOIS

EN CONSOLA

El objetivo de esta aplicación es mantener un catálogo de los MOOC ofrecidos por la decanatura en el período intermedio a sus estudiantes en el programa IEMOIS.

Conociendo el proyecto [En lab04.doc]

No olviden respetar los directorios bin docs src

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

Arquitectura [En lab04.doc, IEMOIS.asta y *.java]

1. Realicen el el diseño arquitectónico con un diagrama de paquetes en el que se presente los componentes y las relaciones entre ellos. ¿Cuántos paquetes tenemos? ¿Cuáles son?
2. Completen el diseño actual de la capa de dominio**.** ¿Qué faltaría considerar?
3. Considerando las funcionalidades del sistema, realicen el diagrama de casos de uso correspondiente. Organice todos los elementos en un modelo llamado useCases0

Adicionar y listar. Todo OK.

[En lab04.doc, IEMOIS.asta y *.java]

(NO OLVIDEN BDD - MDD)

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

1. Adicionen un nuevo MOOC

```
Prendiendo a aprender
aprendizaje
coursera
4 semanas
Este curso te brinda acceso a invaluables técnicas de aprendizaje utilizadas por expertos. Aprenderemos
cómo el cerebro utiliza dos modos de aprendizaje muy distintos y cómo encapsula la información.
```

   ¿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 dominio. ¿Qué método es responsable en la capa de presentación? ¿Qué método en la capa de dominio?
3. Realicen ingeniería reversa para la capa de dominio 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 dominio. ¿Qué método es responsable en la capa de presentación? ¿Qué método en la capa de dominio?
5. Realicen ingeniería reversa de la capa de dominio para **listar**. Capturen los resultados de las pruebas de unidad.
6. Propongan y ejecuten una prueba de aceptación.

Adicionar un MOOC. ¿Y si no da el nombre?

[En lab04.doc, IEMOIS.asta y *.java]
(NO OLVIDEN BDD – MDD)

1. El objetivo es perfeccionar la funcionalidad de adicionar un MOOC.
2. Adicionen a *Aprendiendo a Aprender* sin nombre ¿Qué ocurre? ¿Cómo lo comprueban? Capturen la pantalla. ¿Es adecuado este comportamiento?
3. Vamos a evitar la creación de MOOC sin nombre manejando la excepción `IEMOISExcepcion`. Si el MOOC no tiene nombre no la creamos y se lo comunicamos al usuario[1]. Para esto lo primero que debemos hacer es crear el mensaje en la clase `IEMOISExcepcion`.
4. Analicen el diseño realizado. ¿Qué método debería lanzar la excepción? ¿Qué métodos deberían propagarla? ¿Qué método debería atenderla? Explique claramente.
5. Construyan la solución propuesta. Capturen los resultados de las pruebas.
6. Ejecuten nuevamente la aplicación con el caso de prueba propuesto en 1., ¿Qué sucede ahora? Capture la pantalla.

Adicionar un MOOC. ¿Y si da semanas como un texto?

[En lab04.doc, IEMOIS.asta y *.java]
(NO OLVIDEN BDD - MDD)

El objetivo es perfeccionar la funcionalidad de adicionar un nuevo MOOC.

1. Adicionen de nuevo MOOC *Aprendiendo a Aprender* y en semanas indiquen *varias*. ¿Qué ocurre? 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. Construyan 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 MOOC. ¿Y si ya se encuentra?

[En lab04.doc, IEMOIS.asta y *.java]
(NO OLVIDEN BDD - MDD)

El objetivo es perfeccionar la funcionalidad de adicionar un nuevo MOOC.

1. Adicionen dos veces el nuevo MOOC ¿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. Construyan 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.

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

[En lab04.doc, IEMOIS.asta y *.java]
(NO OLVIDEN BDD - MDD)

1. Consulten un MOOC especial que contenga *Aprender*. ¿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?

---

1 Para presentar los mensajes de error al usuario use el méodo 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

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 Construyann 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 Construyann 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, IEMOIS.asta y *.java]

(NO OLVIDEN BDD - MDD)

1. Revisen el código asociado a **buscar** en la capa de presentación y la capa de dominio. ¿Qué método es responsable en la capa de presentación? ¿Qué método es responsable en la capa de dominio?
2. Realicen ingeniería reversa de la capa de dominio para **buscar**. Capturen los resultados de las pruebas. En este momento las pruebas deben fallar.
3. ¿Cuál es el error? Soluciónenlo. 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 combos? ¿Qué se comprometen a hacer para mejorar los resultados?