PROGRAMACIÓN ORIENTADA A OBJETOS
Excepciones
Octubre 2015
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 XP : *Simplicity, 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.
- ➔ En el foro de entrega deben indicar el estado de avance de su laboratorio y los problemas pendientes por resolver.
- ➔ Deben publicar el avance al final de la sesión y la versión definitiva en la fecha indicada, en los espacios preparados para tal fin.

RECURSOS

PRACTICANDO MDD y TDD con EXCEPCIONES [En lab04.doc, equipos.asta y BlueJ recursos]

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 presenten el diseño estructural de la aplicación. (No incluyan las excepciones)
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")
```

RUTAS DEL CONFLICTO

Conociendo el proyecto Rutas del Conflicto [En lab04.doc]

TRABAJO DESDE CONSOLA

No olviden respetar los directorios bin docs src

1. En su directorio descarguen los archivos contenidos en rutasConflicto.zip, revisen el contenido. ¿cuántas clases tiene el sistema? ¿cómo están organizadas? ¿cuál es la clase ejecutiva? Realicen los diagramas de paquetes y de clases correspondientes. El diagrama de clases de presentación NO debe incluir los elementos privados.
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? Realicen el diagrama de casos de uso correspondiente.
4. ¿De dónde salen las masacres iniciales? Revisen el código y la documentación del proyecto. ¿Qué clase pide que se adicionen? ¿Qué clase los adiciona?

Adicionar y listar. Todo OK. [En lab04.doc, rutasConflicto.asta y *.java]

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

1. Adicionen una nueva masacre

" Masacre de Gualanday 2001

Vereda o Corregimiento: Río Negro
Grupo Armado: Paramilitares del Bloque Calima
Fecha: Noviembre de 2001
A las 3:45 p.m. del 18 de noviembre, paramilitares encapuchados que se movilizaban en motocicletas por una carretera de la vereda Río Negro del municipio de Corinto, Cauca, interceptaron una chiva en la que viajaban varios campesinos e indígenas del resguardo del pueblo. Con lista en mano empezaron a llamar por nombre propio a un grupo de pasajeros y los obligaron a bajar del bus para luego dispararles.

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

2. Revisen el código asociado a **adicionar** en la capa de aplicación y la capa de interfaz. ¿Qué métodos se invocan en la capa de aplicación? ¿Desde qué métodos en la capa de interfaz?
3. Realicen ingeniería reversa para la capa de aplicación para **adicionar**.Capturen los resultados de las pruebas de unidad. **(BDD y MDD)**
4. Revisen el código asociado a **listar** en la capa de aplicación y la capa de interfaz. ¿Qué métodos se invocan en la capa de aplicación? ¿Desde qué métodos en la capa de interfaz?
5. Realicen ingeniería reversa para la capa de aplicación para **listar**. (Capturen los resultados de las pruebas de unidad**. (BDD y MDD)**
6. Propongan y ejecuten una prueba de aceptación.

Adicionar una masacre. ¿Y si no da un nombre? [En lab04.doc, rutasConflicto.asta y *.java]

El objetivo es perfeccionar la funcionalidad de adicionar una masacre.

1. Adicionen la masacre Gualanday sin nombre. ¿Qué ocurre? ¿Cómo lo comprueban? Capturen la pantalla. ¿Es adecuado este comportamiento?
2. Vamos a evitar la creación de masacres con un valor de nombre vacío manejando una excepción `rutasConflictoExcepcion`. Si la masacre no tiene nombre, no la creamos y

se lo comunicamos al usuario[1]. Para esto lo primero que debemos hacer es crear la nueva clase `rutasConflictoExcepcion` 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. (diseño, pruebas de unidad, código) Capturen los resultados de las pruebas de unidad. **(BDD y MDD)**
5. Ejecuten nuevamente la aplicación con el caso de prueba propuesto en 1., ¿Qué sucede ahora? Capture la pantalla.

Adicionar una masacre. ¿Y si ya se encuentra? [En rutasConflicto.asta, lab04.java y *.java]

El objetivo es perfeccionar la funcionalidad de adicionar una masacre.

1. Adicionen dos veces la nueva masacre Gualianday ¿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. (diseño, prueba de unidad, código) Capturen los resultados de las pruebas de unidad. **(BDD y MDD)**
4. Ejecuten nuevamente la aplicación con el caso de prueba propuesto en 1., ¿Qué sucede ahora? Capture la pantalla.

Adicionar una masacre. ¿Otras condiciones? [En lab04.doc, rutasConflicto.asta y *.java]

El objetivo es perfeccionar la funcionalidad de adicionar una masacre.

1. Propongan nuevas condiciones para que la adición de una masacre sea más robusta.[2]
2. Construya la solución propuesta. (diseño, prueba de unidad, código) Capturen los resultados de las pruebas de unidad. **(BDD y MDD)**

Consultando por patrones. ¡ No funciona y queda sin funcionar!
[En rutasConflicto.asta, rutasConflicto.log, lab04.java y *.java]

1. Consulten una masacre especial que inicie con La. ¿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?

---

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

2 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

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!

1. Revisen el código asociado a **adicionar** en la capa de aplicación y la capa de interfaz. ¿Qué métodos se invocan en la capa de aplicación? ¿Desde qué métodos en la capa de interfaz?
2. Realicen ingeniería reversa para la capa de aplicación para **adicionar**. Capturen los resultados de las pruebas de unidad. **(BDD y MDD)**
3. ¿Cuál es el error? Solucionenlo. Capturen los resultados de las pruebas de unidad. **(BDD y MDD)**
4. Ejecuten la aplicación nuevamente con el caso propuesto. ¿Qué tenemos en pantalla? ¿Qué información tiene el archivo de errores?

RETROSPECTIVA

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