

# áreaCIÓN ORIENTADA A OBJETOS

## Excepciones

### Octubre 2017

### 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. ¿De qué clase no pueden consultar las fuentes? ¿En qué formato está? ¿Por qué? Estudien su documentación en el diseño o en el código (/doc).
3. Dadas las pruebas, diseñen y codifiquen el método `valorHora`.
4. Dada la especificación, diseñen, codifiquen y prueben el método `valorHoraEstimado`.

### PARA LAS PRUEBAS

Las siguientes personas que tienen valor hora conocido:	Las siguientes personas son conocidas pero no tienen valor hora:
<code>("Pedro",10000);</code>	<code>("Garcia");</code>
<code>("Santiago",20000);</code>	<code>("Ospina");</code>
<code>("Marcos",30000);</code>	<code>("Guarin")</code>
<code>("Juan",40000);</code>	
<code>("Judas",50000);</code>	

# SINAP. Sistema Nacional de Áreas Protegidas

## EN CONSOLA

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

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

1. En su directorio descarguen los archivos contenidos en [SINAP.zip](#), revisen el contenido. ¿Cuántos archivos se tienen? ¿Cómo están organizados? ¿Cómo deberían estar organizados?
2. Estudien el diseño del programa: diagramas de paquetes y de clases. ¿cuántos paquetes tenemos? ¿cuántas clases tiene el sistema? ¿cómo están organizadas? ¿cuál es la clase ejecutiva?
3. Prepare los directorios necesarios para ejecutar el proyecto. ¿qué estructura debe tener? ¿qué instrucciones debe dar para ejecutarlo?
4. Ejecute el proyecto, ¿qué funcionalidades ofrece? ¿cuáles funcionan? Realicen el diagrama de casos de uso correspondiente.
5. ¿De dónde salen las áreas 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, SINAP.asta y \*.java]

#### (NO OLVIDEN BDD - MDD)

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

1. Adicionen una nueva área

#### Tuparro

Tuparro National Park  
Vichada  
548.000

Es una extensa sabana verde surcada por grandes ríos con potentes raudales y playas doradas, pequeños caños de aguas cristalinas, bosques de galería, morichales y saladillales, además de enormes rocas cristalinas en forma de cerros redondeados.

¿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 para la capa de aplicación para **listar**. Capturen los resultados de las pruebas de unidad.
6. Propongan y ejecuten una prueba de aceptación.

### Adicionar un área. ¿Y si no da un nombre? [En lab04.doc, SINAP.asta y \*.java]

#### (NO OLVIDEN BDD - MDD)

El objetivo es perfeccionar la funcionalidad de adicionar un área.

1. Adicionen el área Tuparro sin nombre internacional. ¿Qué ocurre? ¿Cómo lo comprueban? Capturen la pantalla. ¿Es adecuado este comportamiento?
2. Vamos a evitar la creación de áreas con un nombre internacional vacío manejando una excepción [SINAPExcepcion](#). Si el área no tiene nombre internacional, no la creamos y se

lo comunicamos al usuario<sup>1</sup>. Para esto lo primero que debemos hacer es crear la nueva clase `SINAPExcepcion` 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 área. ¿Y si ya se encuentra? [En `SINAPasta`, `lab04.java` y `*.java`]

(NO OLVIDEN BDD - MDD)

El objetivo es perfeccionar la funcionalidad de adicionar un área.

1. Adicionen dos veces la nueva área Tuparro ¿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 área. ¿Otras condiciones? [En `lab04.doc`, `SINAPasta` y `*.java`]

(NO OLVIDEN BDD - MDD)

El objetivo es perfeccionar la funcionalidad de adicionar un área.

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

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

[En `SINAPasta`, `SINAP.log`, `lab04.java` y `*.java`]

(NO OLVIDEN BDD - MDD)

1. Consulten un área especial que inicie con T. ¿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?

---

<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

<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

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!**

(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 para la capa de aplicación para **buscar**. Capturen los resultados de 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 equipo? ¿Qué se comprometen a hacer para mejorar los resultados?