

# PROGRAMACIÓN ORIENTADA A OBJETOS

## Excepciones

### 2024-1

#### 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 las prácticas **Designing** - [Simplicity](#).

□ [Refactor](#) whenever and wherever possible.

### 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.

## Proyectos

### EN BLUEJ

#### PRACTICANDO MDD y BDD con EXCEPCIONES

[En [lab04.doc](#), [activities.asta](#) y [Bluej activities](#)]

En este punto vamos a aprender a diseñar, codificar y probar usando excepciones. Para esto se van a trabajar algunos métodos de la clase [Activity](#)

1. En su directorio descarguen los archivos contenidos en [activities.zip](#) revisen el contenido y estudien el diseño parcial que está en el diagrama de clases. ¿Qué estructura de datos es Actividad? Justifique la respuesta.
2. Expliquen porque el proyecto no compila. Realicen las adiciones necesarias para lograrlo.
3. Dadas las pruebas, documenten, diseñen y codifiquen el método `time()`.  
Adicionen pruebas análogas a cada una de las dadas para una actividad más profunda (nivel 3)
4. Diseñen, codifiquen y prueben el método `time(unknown, error, empty)`.
5. Diseñen, codifiquen y prueben el método `time(estimate)`.
6. Diseñen, codifiquen y prueben el método `time(activity)`.

## Project

### EN CONSOLA

El objetivo de esta aplicación es realizar un gestor de proyectos. Los proyectos están compuestos de dos tipos de actividades: actividades simples y actividades compuestas. Las actividades compuestas pueden ser paralelas o secuenciales.

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

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

1. En su directorio descarguen los archivos contenidos en [project.zip](#), revisen el contenido. ¿Cuántos archivos se tienen? ¿Cómo están organizados? ¿Cómo deberían estar organizados?
2. Realicen el diseño del programa: diagramas de paquetes y de clases (presentación y dominio). ¿cuántos paquetes tenemos? ¿cuántas clases tiene el sistema? ¿cuál es la clase ejecutiva?
3. Prepare los directorios necesarios para ejecutar el proyecto. ¿qué estructura debe tener? ¿qué clases debe tener cada directorio? ¿dónde están esas clases? ¿qué instrucciones debe dar para ejecutarlo?
4. Ejecute el proyecto, ¿qué funcionalidades ofrece? ¿cuáles funcionan?
5. Revisen el código y la documentación del proyecto. ¿De dónde salen los disfraces iniciales? ¿Qué clase pide que se adicionen? ¿Qué clase los adiciona?

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

##### (NO OLVIDEN BDD - MDD)

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

1. Adicionen una nueva actividad simple y una nueva actividad compuesta

Simple

Iterar

tiempo 500 , Costo 10000

*Completo*

Iterar 3 veces

100, Pararela

Iterar

Iterar

Iterar

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

## Adicionar una actividad. Funcionalidad robusto [En lab04.doc, Project.asta y \*.java]

(NO OLVIDEN BDD - MDD)

El objetivo es perfeccionar la funcionalidad de adicionar un curso para hacerla más robusta.

**Para cada uno de los siguientes casos realice los pasos del 1 al 4.**

**a. ¿Y si el nombre de la actividad ya existe?**

**b. ¿Y si en precio o costo no da un número? ¿si el tipo no es paralelo o secuencial?**

**c. Proponga una nueva condición**

1. Propongan una prueba de aceptación que genere el fallo.
2. Analicen el diseño realizado. Para hacer el software robusto: ¿Qué método debería lanzar la excepción? ¿Qué métodos deberían propagarla? ¿Qué método debería atenderla? Explique claramente.
3. Construya la solución propuesta. Capture los resultados de las pruebas de unidad.
4. Ejecuten nuevamente la aplicación con el caso de aceptación propuesto en 1. ¿Qué sucede ahora? Capture la pantalla.

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

[En Project.asta, Project.log, lab04.java y \*.java]

(NO OLVIDEN BDD - MDD)

1. Consulten una actividad que inicie con I. ¿Qué sucede? ¿Qué creen que pasó? Capturen el resultado. ¿Quién debe conocer y quien NO debe conocer esta información?
2. Exploren el método `record` de la clase `Log` ¿Qué servicio presta?
3. 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.
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. Modifiquen la aplicación para garantizar que **SIEMPRE** que haya un error se maneje de forma adecuada. ¿Cuál fue la solución implementada?

## Consultando por patrones. ¡Ahora si funciona!

[En Project.asta, Project.log, lab04.java 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 para la capa de dominio 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?
5. Refactorice la funcionalidad para que sea más amable con el usuario. ¿Cuál es la propuesta? ¿Cómo la implementa?

## **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 actividades? ¿Qué se comprometen a hacer para mejorar los resultados?