**DESARROLLO ORIENTADO POR OBJETOS [DOPO-POOB]**
**Excepciones**
**2025-2**
**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. Practicar 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.

## CONTEXTO: Clash Royale

Es un videojuego de estrategia en línea basado en los personajes de Clash of Clans.

- Cada jugador tiene un mazo de 8 cartas que representan tropas, hechizos o estructuras.
- Las cartas se juegan usando elixir, un recurso que se regenera con el tiempo.
- El objetivo es destruir las torres del oponente, especialmente la torre central (la del rey).

# EN BLUEJ

## PRACTICANDO MDD y BDD con EXCEPCIONES

**[En lab04.doc, clashRoyale.asta y BlueJ units]**

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

1. En su directorio descarguen los archivos contenidos en `battleDeck.zip` revisen el contenido y estudien el diseño estructural de la aplicación (únicamente la zona en azul).
2. Expliquen por qué el proyecto no compila. Realicen las adiciones necesarias para lograrlo.
3. Dado el diseño y las pruebas, documenten (si es necesario) y codifiquen el método `elixir()`.
4. Dada la documentación y el diseño, codifiquen y prueben el método `knownElixir()`.
5. Documenten, diseñen, codifiquen y prueben el método
   `estimateElixir(unknown:String, error: String)`
   `[unknown y error pueden tomar los siguientes valores MIN, MAX o AVG.`
   `Indican el valor que se usa para reemplazar un valor con error o desconocido:`
   `el mínimo, el máximo o el promedio de los valores válidos del mazo]`
6. Propongan el cálculo, documenten, diseñen, codifiquen y prueben el método `resistance()`

# Clash Royale
## EN CONSOLA

El objetivo de esta aplicación es mantener un catálogo de mazos y cartas de *Clash Royale*.

**Conociendo el proyecto ClashRoyale** **[En lab04.doc]**
**No olviden respetar los directorios bin docs src**

1. En su directorio descarguen los archivos contenidos en clashRoyale.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 clases deben tener? ¿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, clashRoyale.asta y *.java]

(NO OLVIDEN BDD - MDD)

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

1. Adicionen una nueva carta y un nuevo mazo

   Card

   ```
   Knight
   Type: TROOP
   Elixir: 3
   Hitpoints;  1579
   ```

   *Mazo*

   ```
   NoMercy
   Type:  Defensive
   Cards: Knight
          Baby Dragon
   ```

   ¿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 carta o un mazo. Funcionalidad robusta [En lab04.doc, clashRoyale.asta y *.java]

(NO OLVIDEN BDD – MDD)

El objetivo es perfeccionar la funcionalidad de adicionar una carta 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 carta o el mazo ya existe?**

**b. ¿Y si los valores numéricos no son numéricos?**

**c. ¿Y si el elixir no tiene los valores esperados?**

**d. 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 clashRoyale.asta, clashRoyale.log, lab04.java y *.java]

(NO OLVIDEN BDD - MDD)

1. Consulten un mazo que inicie con C. ¿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 clashRoyale.asta, clashRoyale.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?
7. ¿Qué referencias usaron? ¿Cuál fue la más útil?  Incluyan citas con estándares adecuados.