### PROGRAMACIÓN ORIENTADA A OBJETOS

**Excepciones 2019**

**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*](http://www.extremeprogramming.org/rules/simple.html)*.*

**Coding** Code must be written to agreed [standards](http://www.extremeprogramming.org/rules/standards.html)

### ENTREGA

1. Incluyan en un archivo .zip los archivos correspondientes al laboratorio. El nombre debe ser los dos apellidos de los miembros del equipo ordenados alfabéticamente.
2. 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 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. Dadas las pruebas, diseñen y codifiquen el método valorHora.
3. Dada la especificación, diseñen, codifiquen y prueben el método valorHoraEstimado.
4. Dada la especificación, diseñen, codifiquen y prueben el método valorHoraAsumido.

#### 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")

**Selección Colombia**

**EN CONSOLA**

**Conociendo el proyecto [En lab04.doc]**

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

1. En su directorio descarguen los archivos contenidos en seleccion.zip, revisen el contenido.

¿Cuántos archivos se tienen? ¿Cómo están organizados? ¿Cómo deberían estar organizados?

-Cinco archivos 4 . java y un .asta, los cuales deberían estar organizados en la carpetas bin,docs,src y por paquetes de presentacion y aplicacion.

1. Prepare los directorios necesarios para ejecutar el proyecto. ¿qué estructura debe tener?¿qué instrucciones debe dar para ejecutarlo?

-la estructura que debe tener es en carpetas bin, src, docs y sus instrucciones son:

md bin

md src

md docs

cd src

md Presentacion

md Aplicacion

c../

cd bin

md Presentacion

md Aplicacion

c../

COPY \*.java src

del \*.java

cd src

COPY SelecionGUI.java Presentacion

del SelecionGUI.java

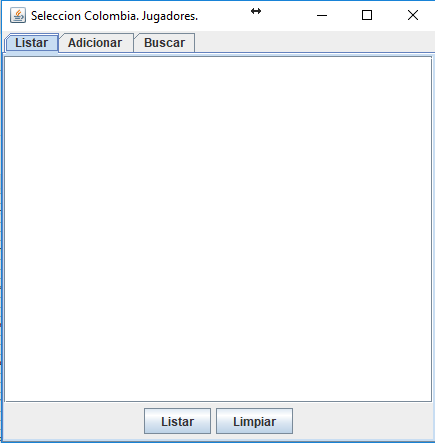
COPY \*.java Aplicacion

del \*.java

cd../

javac -d bin src\Presentacion\\*.java src\aplicacion\\*.java

java -cp bin presentacion.SeleccionGUI

* + Ejecute el proyecto, ¿qué funcionalidades ofrece? ¿cuáles funcionan?  
    
  + Listar a los jugadores
  + Adicionar a los jugadores
  + Buscar a los jugadores
  + Limpiar la lista de los jugadores

1. Revisen el código del proyecto. ¿De dónde salen los jugadores iniciales? ¿Qué clase pide que se adicionen? ¿Qué clase los adiciona?

Provienen de la clase Selección, la clase Registro es la que pide que se adicionen. La clase que los adiciona es Selección.

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

1. Inicie el diseño con un diagrama de paquetes en el que se presente los componentes y las relaciones entre ellos.
2. Estudie el diseño actual de la **capa de aplicación.**
3. Considerando las funcionalidades del sistema. Realicen el diagrama de casos de uso correspondiente. Organice todos los elementos en un modelo llamado useCases0 al

## Adicionar y listar. Todo OK.

[En lab04.doc, selección.asta y \*.java]

(NO OLVIDEN BDD - MDD)

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

1. Adicionen un nuevo jugador

Yerry Mina 195

Defensa

Nominado al FIFA/FIFPro World XI 2018

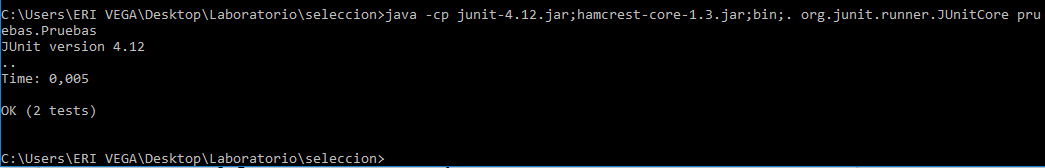
1.¿Qué ocurre? ¿Cómo lo comprueban? Capturen la pantalla. ¿Es adecuado este comportamiento?

Lo adiciona a la lista, ya que se puede adicionar muchas veces un jugador.

1. 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?

En aplicación es responsable adicione de la clase selección, en presentación es accionAdicionar.

1. Realicen ingeniería reversa para la capa de aplicación para **adicionar**. Capturen los resultados de las pruebas de unidad.  
   ***Compilar proyecto:*** C:\Users\ERI VEGA\Desktop\Laboratorio\seleccion>javac -d bin src\presentacion\\*.java src\aplicacion\\*.java

***Compilar Pruebas :*** C:\Users\ERI VEGA\Desktop\Laboratorio\seleccion\src>

javac -cp junit-4.12.jar;. pruebas/Pruebas.java -d "C:\Users\ERI VEGA\Desktop\Laboratorio\seleccion\bin"

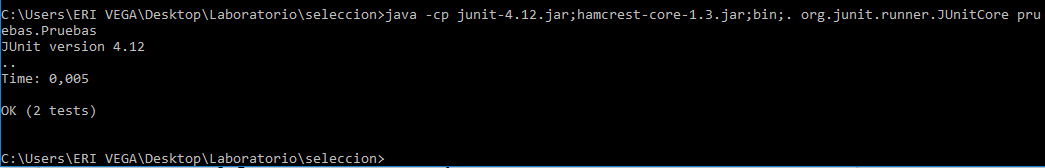
***Ejecutar pruebas:*** java -cp junit-4.12.jar;hamcrest-core-1.3.jar;bin;. org.junit.runner.JUnitCore pruebas.Pruebas

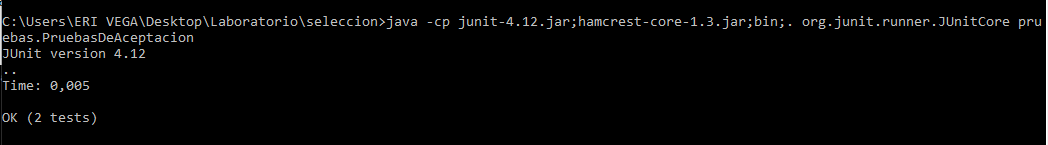
***Ejecutar proyecto:*** C:\Users\ERI VEGA\Desktop\Laboratorio\seleccion>java -cp bin presentacion.SeleccionGUI

1. 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?

En la capa de presentación, el método responsable es accionListar.

En la capa de aplicación, el método responsable es toString

1. Realicen ingeniería reversa de la capa de aplicación para **listar**. Capturen los resultados de las pruebas de unidad.  
     
     
   
2. Propongan y ejecuten una prueba de aceptación.



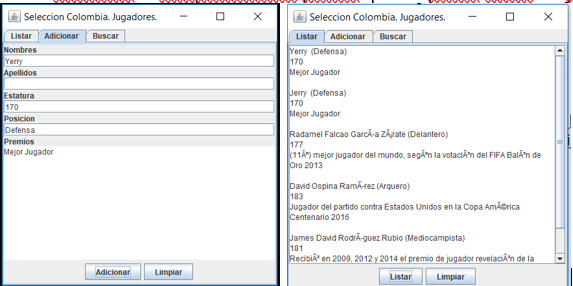
## Adicionar un jugador. ¿Y si no da los apellidos?

[En lab04.doc, selección.asta y \*.java]

(NO OLVIDEN BDD – MDD)

El objetivo es perfeccionar la funcionalidad de adicionar un jugador.

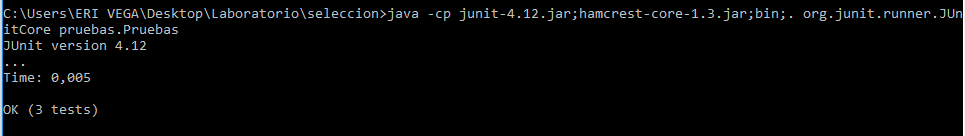
1. Adicionen a Yerry sin apellido ¿Qué ocurre? ¿Cómo lo comprueban? Capturen la pantalla. ¿Es adecuado este comportamiento?

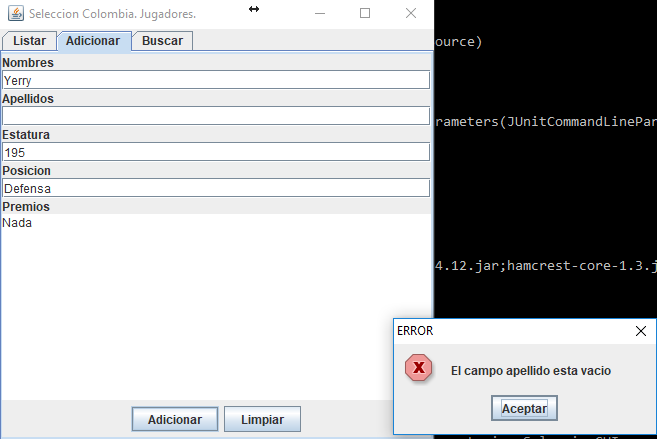


El programa muestra una ventana la cual indica que el campo de “apellido”, y no adiciona a este jugador.

1. Vamos a evitar la creación de áreas con un distribuidor vacío manejando una excepción SeleccionExcepcion. Si el jugador no tiene distribuidor, no lo creamos y se lo comunicamo

al usuario. Para esto lo primero que debemos hacer es crear la nueva clase SeleccionExcepcion considerando este primer mensaje.

1. Analicen el diseño realizado. ¿Qué método debería lanzar la excepción? ¿Qué métodos deberían propagarla? ¿Qué método debería atenderla? Explique claramente.  
     
   La clase Selección e sla que debería lanzar la excepción, el método adicione es la que debería propagarla y el método SeleccionGUI es la que debería atenderlo.
2. Construya la solución propuesta. Capturen los resultados de las pruebas.  
     
   
3. Ejecute nuevamente la aplicación con el caso de prueba propuesto en 1., ¿Qué sucede ahora? Capture la pantalla.



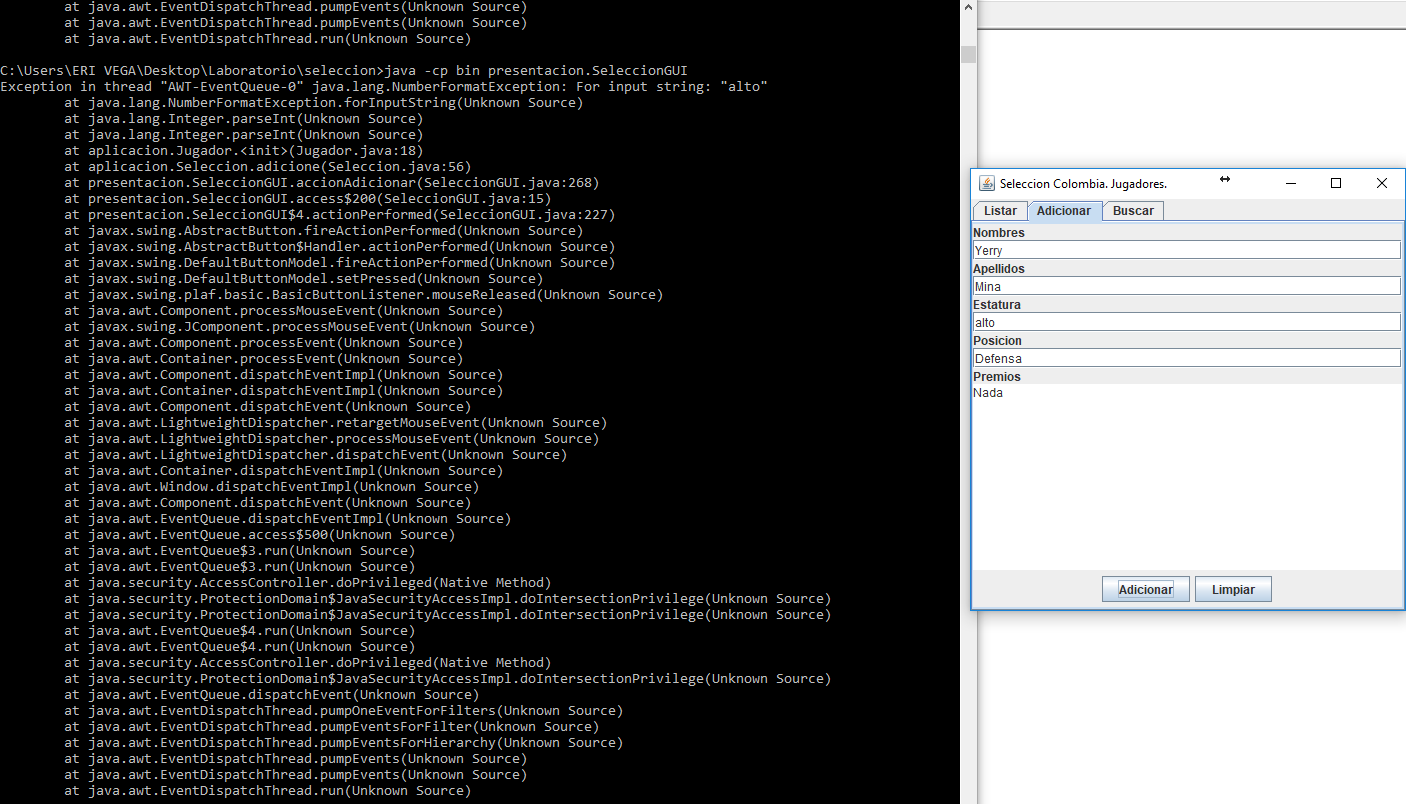
## Adicionar un jugador. ¿Y si da como altura un texto?

[En lab04.doc, seleccion.asta y \*.java]

(NO OLVIDEN BDD - MDD)

El objetivo es perfeccionar la funcionalidad de adicionar un jugador.

1. Adicionen el nuevo jugador *Yerry Mina* pero en lugar de 195 coloquen alto. ¿Qué ocurre?

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

1. Analicen el diseño realizado. ¿Qué método lanzan la excepción? ¿Qué métodos la propagan? Explique claramente. ¿Qué problema tenemos ahí?  
   Debería lanzar la excepcion el método adicione, debería propagarla adicione y accionAdicionar.

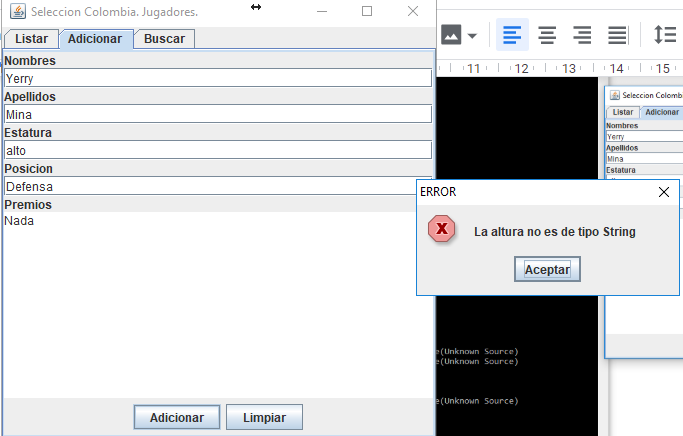
El problema que tenemos, es que es una excepcion muy general, toca verificar si lo que está en cadena se puede convertir a numero y así poder mandar o no la excepcion.

Deberia lanzar expecion el metodo Adicione

1. Analicen el diseño realizado. ¿Qué método debería lanzar la excepción propia? ¿Qué métodos deberían propagarla? ¿Qué método debería atenderla? Explique claramente.

El método que debería lanzar la excepción propia es Seleccion, el que la propaga es Adicione y Seleccione. En SeleccionGUI, el metodo accionAdicionar.

1. Ejecuten nuevamente la aplicación con el caso de prueba propuesto en 1., ¿Qué sucede ahora? Capture la pantalla.

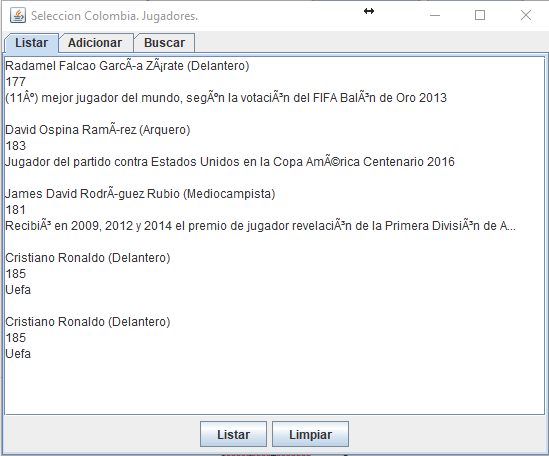


## Adicionar un jugador. ¿Y si ya se encuentra?

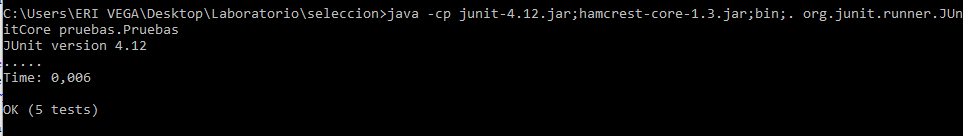
[En lab04.doc, seleccion.asta y \*.java]

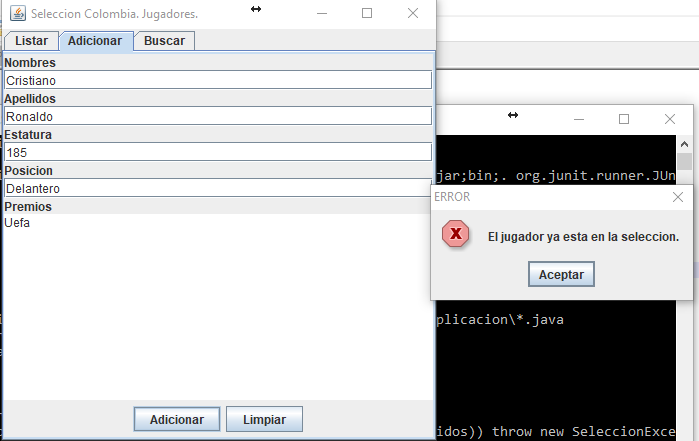
(NO OLVIDEN BDD - MDD)

El objetivo es perfeccionar la funcionalidad de adicionar un jugador.

1. Adicionen dos veces el nuevo jugador ¿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.

Debería lanzar la excepcion el método adicione, debería propagarla adicione y accionAdicionar. El problema que tenemos es que toca verificar si el jugador ya existe en nuestro sistema.

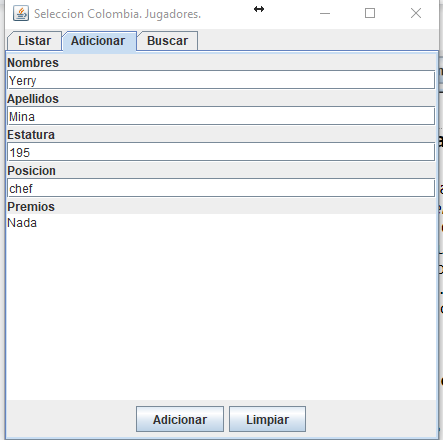
1. Construya la solución propuesta. Capturen los resultados de las pruebas.  
   
2. Ejecuten nuevamente la aplicación con el caso de prueba propuesto en 1., ¿Qué sucede ahora? Capture la pantalla.

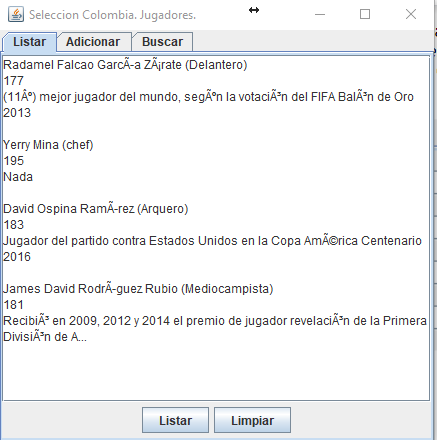


**Adicionar un jugador. ¿Y si dan mal la posición?** [En lab04.doc, seleccion.asta y \*.java]

(NO OLVIDEN BDD - MDD)

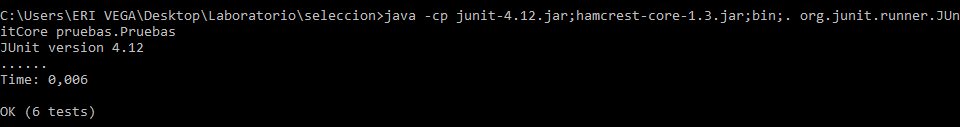
El objetivo es perfeccionar la funcionalidad de adicionar un jugador.

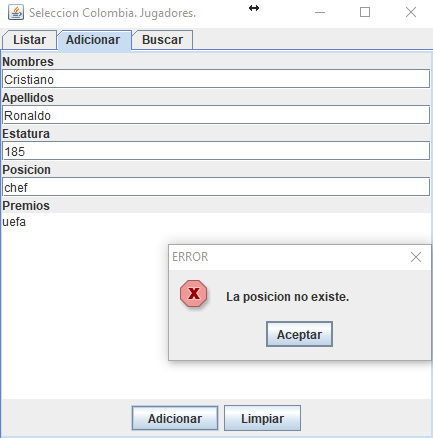
1. Adicionen el nuevo jugador *Yerry Mina* pero en lugar de Delantero coloquen chef. ¿Qué ocurre? ¿Cómo lo comprueban? Capturen la pantalla. ¿Es adecuado este comportamiento?  
   



1. Analicen el diseño realizado. ¿Qué método debería lanzar la excepción? ¿Qué métodos deberían propagarla? ¿Qué método debería atenderla? Explique claramente.

Debería lanzar la excepcion el método adicione, debería propagarla adicione y accionAdicionar. El problema que tenemos que definir los tipos de posiciones que existen para asi comparar que sean las correctas.

1. Construya la solución propuesta. Capturen los resultados de las pruebas.  
   
2. Ejecuten nuevamente la aplicación con el caso de prueba propuesto en 1., ¿Qué sucede ahora? Capture la pantalla.



**Adicionar un jugador. ¿Otras condiciones?** [En lab04.doc, seleccion.asta y \*.java]

1 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

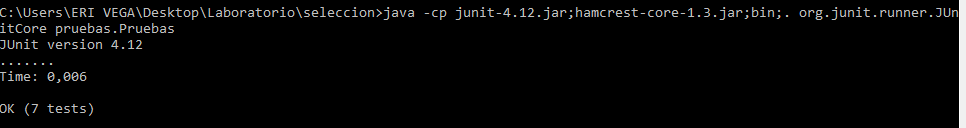
(NO OLVIDEN BDD – MDD)

El objetivo es perfeccionar la funcionalidad de adicionar un jugador.

1. Propongan nuevas condiciones para que la adición de un jugador sea más robusta.

Que el ususario no llegue a digitar el nombre del jugador es nuestra propuesta.

1. 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 lab04.doc, seleccion.asta y \*.java]

(NO OLVIDEN BDD - MDD)

1. Consulten un jugador especial que inicie con Y. ¿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?
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!** [En lab04.doc, seleccion.asta y \*.java]

(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?

1. Realicen ingeniería reversa de la capa de aplicación para **buscar**. Capturen los resultados de las pruebas. Deben fallar.
2. ¿Cuál es el error? Soluciónenlo. Capturen los resultados de las pruebas.
3. 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)

18 horas fue el total de tiempo invertido

1. ¿Cuál es el estado actual del laboratorio? ¿Por qué?

Su estado actual es incompleto, porque falta la capa buscar.

1. Considerando las prácticas XP del laboratorio. ¿cuál fue la más útil? ¿por qué?

La práctica XP más útil fue la de trabajo a pares, ya que es un gran método de trabajo.

1. ¿Cuál consideran fue el mayor logro? ¿Por qué?

Lograr ejecutar todo desde consola con un mínimo conocimiento sobre cmd.

1. ¿Cuál consideran que fue el mayor problema técnico? ¿Qué hicieron para resolverlo?

Fue no conocer la sintaxis y la manera de usar cmd.

1. ¿Qué hicieron bien como equipo? ¿Qué se comprometen a hacer para mejorar los resultados?

Nos comprometemos a conocer un poco mas y ser un poco mas efectivos al momento de trabajar.