**PROGRAMACIÓN ORIENTADA A OBJETOS**

**Excepciones**

**2023-2**

**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. Vivenciar las prácticas ***Designing*** [*Simplicity*](http://www.extremeprogramming.org/rules/simple.html)*.*

[Refactor](http://www.extremeprogramming.org/rules/refactor.html)  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.

**Disfaces**

# EN BLUEJ

**PRACTICANDO MDD y BDD con EXCEPCIONES**

**[En lab04.doc, CostumeShop.asta y BlueJ costumes]**

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

1. En su directorio descarguen los archivos contenidos en costumes.zip revisen el contenido y estudien el diseño estructural de la aplicación (únicamente la zona en azul).

R//: En la parte azul encontramos que hay tres clases, la primera: costume es una clase abstracta que tiene dos metodos abstractos. Y las otras dos clases son herencias de esta, que son complete y basic.

1. Expliquen por qué el proyecto no compila. Realicen las adiciones necesarias para lograrlo.

R//: El Proyecto no compila porque falta la clase de Excepciones y los mensajes por defecto

1. Dado el diseño y las pruebas, documenten y codifiquen el método price().
2. Dada la documentación y el diseño, codifiquen y prueben el método price(type).
3. Documenten, diseñen, codifiquen y prueben el método price(makeUp).

**CostumeShop**

# EN CONSOLA

El objetivo de esta aplicación es mantener un catálogo de los disfraces ofrecidos por una tienda en Halloween en el proyecto CostumeShop. En este proyecto se ofrecen diferentes tipos de disfraces:básicos y completos.

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

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

1. En su directorio descarguen los archivos contenidos en costumeShop.zip, revisen el contenido. ¿Cuántos archivos se tienen? ¿Cómo están organizados? ¿Cómo deberían estar organizados?

R//: Tiene 3 archivos, esta organizado en una sola carpeta llamada costumeShop-2023-2 Deberia estar organizada siguiendo el principio de bin, docs, src.

1. 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?

R//: Deberian haber tres paquetes, domain, presentation y test. En el presentation temenos 3 clases

1. 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?

R//: debe tener la estructura de test, domain y presentation. La clase domain esta en cosutume, las clases de test también están en costume y las clases de presentation están en costumeShop

1. Ejecute el proyecto, ¿qué funcionalidades ofrece? ¿cuáles funcionan?

Ofrece la funcionalidad de listar, adicionar, buscar.

Listar cuando se clikea muestra los disfraces disponibles

Adicionar un disfraz nuevo

Buscar no hace nada aun

1. Revisen el código y la documentación del proyecto. ¿De dónde salen los disfraces iniciales?

¿Qué clase pide que se adicionen? ¿Qué clase los adiciona?

Los disfraces iniciales salen del metodo addSome() que esta en CostumeShop, la clase que los pide es CostumeShopGUI

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

## (NO OLVIDEN BDD - MDD)

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

1. Adicionen un nuevo disfraz básico y un nuevo disfraz completo Básico

Antifaz blanco

500 , 10 *Completo*

Fantasma

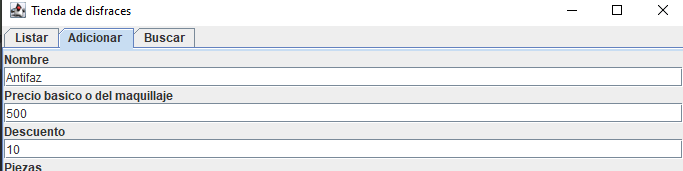
1000, 0

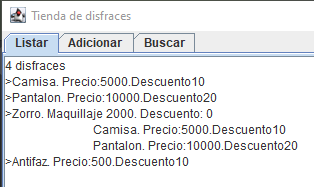
Camisa blanca

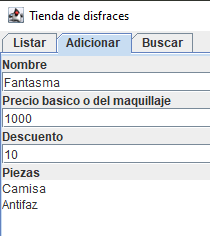
Antifaz blanco

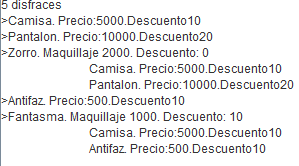
¿Qué ocurre? ¿Cómo lo comprueban? Capturen la pantalla. ¿Es adecuado este comportamiento?

R//: Ocurre que se agrega el basic cuando no se especifica si tiene piezas, verificamos esto, limpiando y listando otra vez









1. 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?

R//: En presentation el responsable del método es actionAdd(), el responsable del método en domain son los constructores Basic y Complete.

1. Realicen ingeniería reversa para la capa de dominio para **adicionar**.

Capturen los resultados de las pruebas de unidad.







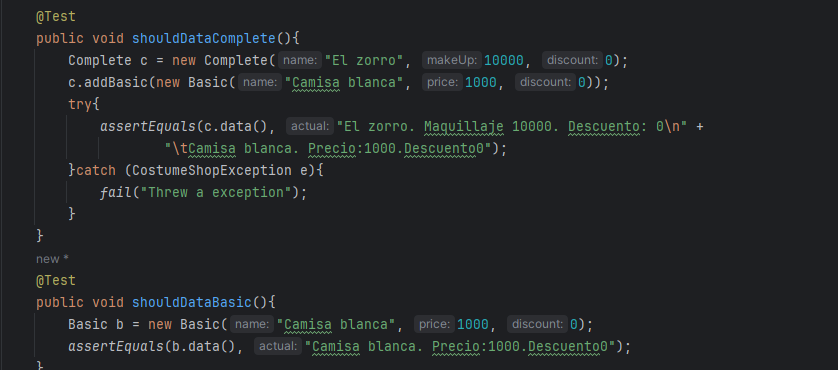


1. Revisen el código asociado a **listar** en la capa de presentación y la capa de dominio. ¿Qué método es responsable en la capa de presentación? ¿Qué método en la capa de dominio?

R//: El método responsable en la capa de presentación es actionList(), en dominio la funcion es Add()

1. Realicen ingeniería reversa para la capa de dominio para **listar**.

Capturen los resultados de las pruebas de unidad.





1. Propongan y ejecuten una prueba de aceptación.

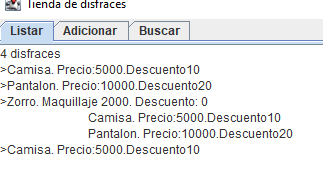
**Adicionar un disfraz. Funcionalidad robusto**  [En lab04.doc, CostumeShop.astay \*.java]

## (NO OLVIDEN BDD – MDD)

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

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

1. **¿Y si el nombre del disfraz ya existe?**
2. **¿Y si en precio no da un número? ¿o no da un número negativo?**
3. **Proponga una nueva condición**
4. Propongan una prueba de aceptación que genere el fallo.
5. 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.
6. Construya la solución propuesta. Capture los resultados de las pruebas de unidad.
7. Ejecuten nuevamente la aplicación con el caso de aceptación propuesto en 1. ¿Qué sucede ahora? Capture la pantalla.
8. ¿Y si el nombre del disfraz ya existe?
9. Proponemos la prueba de agregar un disfraz complete que ya existe desde addSome() como lo es zorro



Actualmente puede ponerse nombres repetidos.

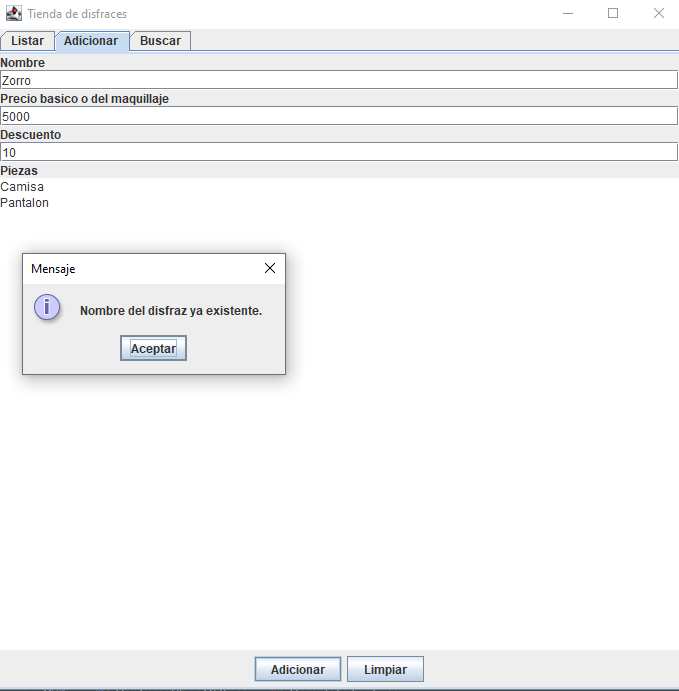
1. El método addComplete debería lanzarla y propagarla, y desde CostumeShopGUI debería atenderse, en este caso mostrando un mensaje al usuario.
2. Construcción de la solución:

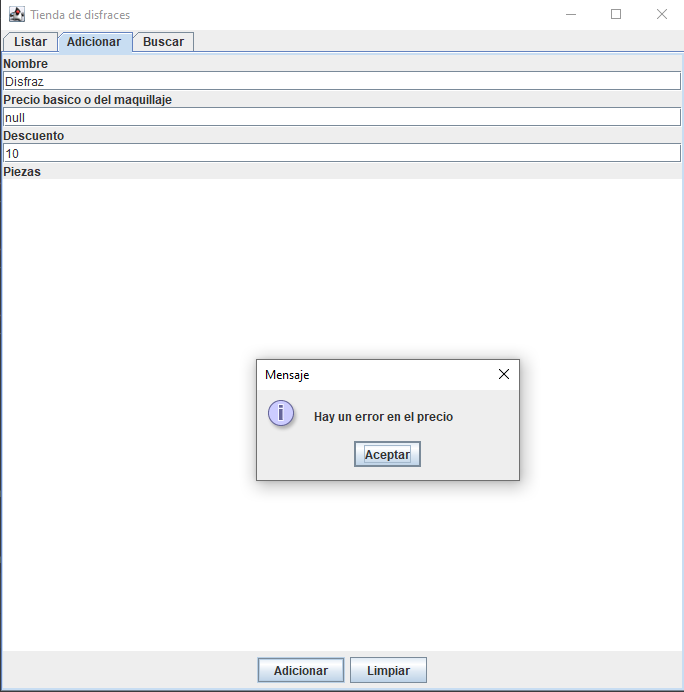
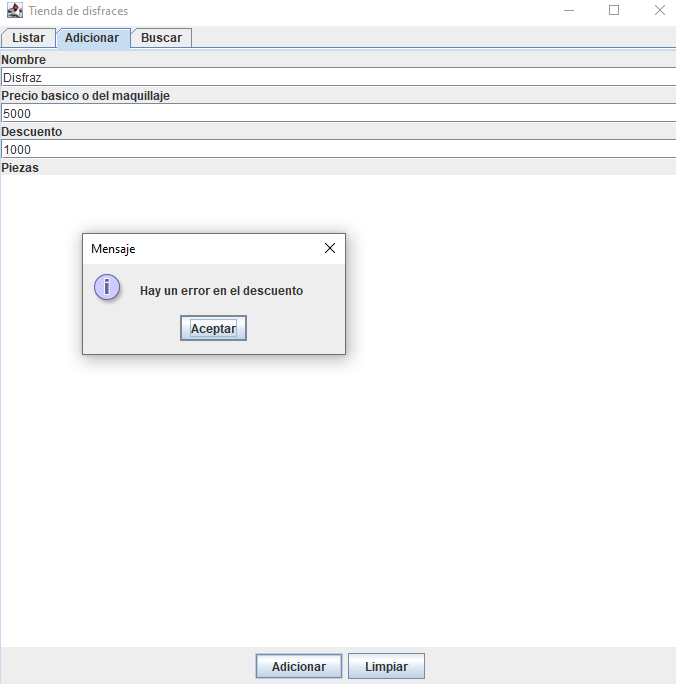
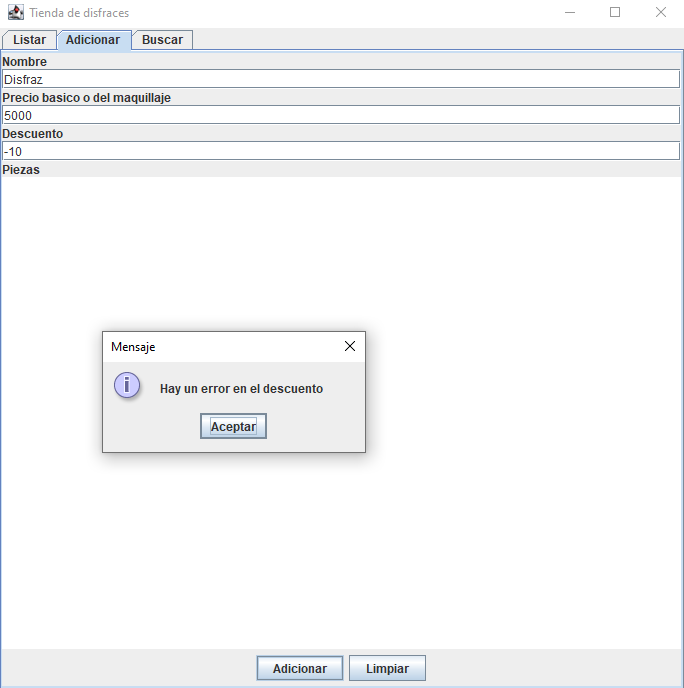










1. ¿Y si en precio no da un número? ¿o no da un número negativo?
2. Proponemos que cuando el ingreso del precio sea algún string desconocido se lance un error al usuario, cuando es precio sea negativo se le asigne por defecto 0
3. El método que lo lanza es el addBasic de CostumeShop, lo propaga addBasic en CostumeShop y lo atiende CostumeShopGUI mostrando el mensaje al usuario
4. 
5. 
6. Proponemos la condición de que cuando se agregue un descuento, este no sea mayor a 100 o menor a 0.
7. La prueba va a ser tratar de ingresar un basic con un descuento del 200%
8. El método que lo lanza es Price de Basic, lo propaga addBasic y lo atiende CostumeShopGUI
9. 
10.  

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

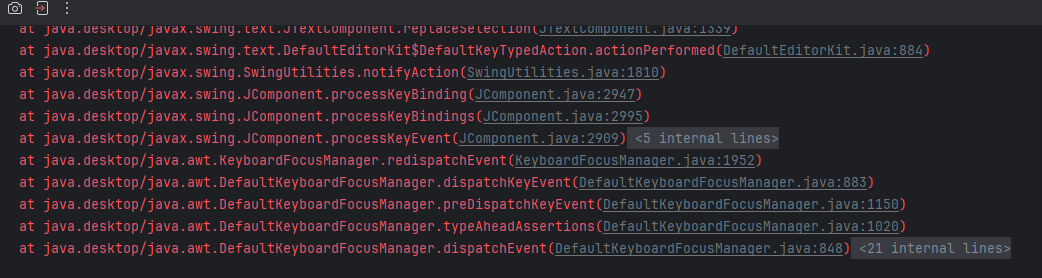
[En CostumeShop.asta, CostumeShop.log, lab04.java y \*.java]

## (NO OLVIDEN BDD - MDD)

1. Consulten un disfraz completo que inicie con I. ¿Qué sucede? ¿Qué creen que pasó? Capturen el resultado. ¿Quién debe conocer y quien NO debe conocer esta información?

R//:

* ¿Qué sucede?

No hace nada, solo muestra errores 

* ¿Qué creen que pasó?

Es como si estuviera mal definido answer, da error de nullPointer



* ¿Quién debe conocer y quien NO debe conocer esta información?

Los errores deberían conocerlos los programadores, este error no debería salirle al usuario.

1. Exploren el método record de la clase Log ¿Qué servicio presta?

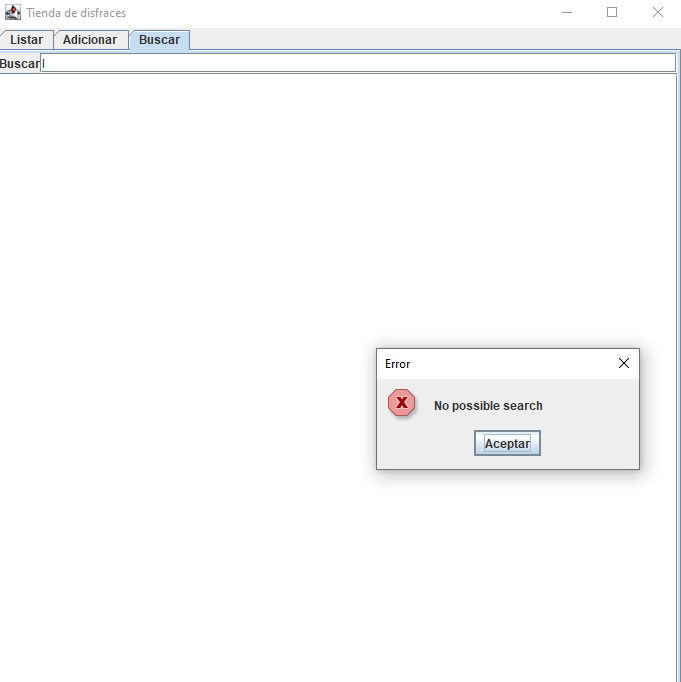
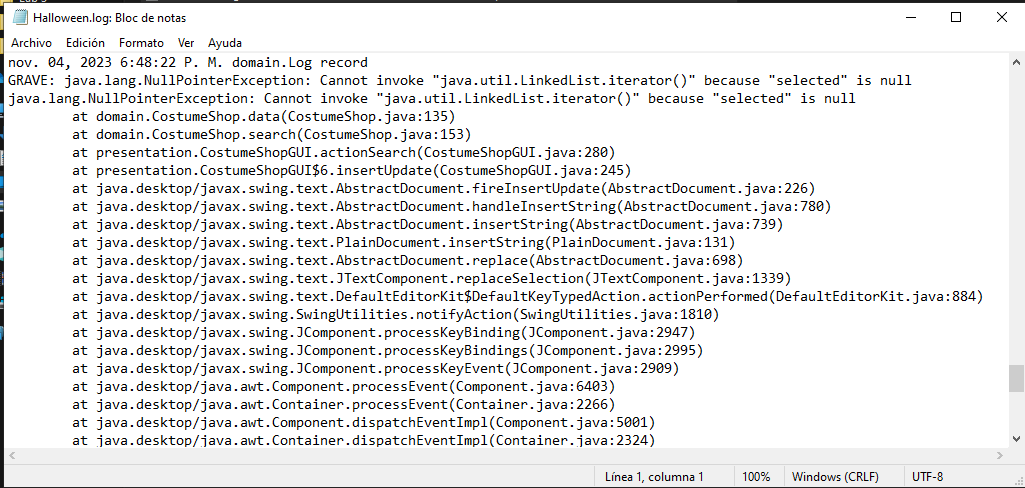
La clase Log se utiliza para registrar excepciones en un archivo de registro. Utiliza un Logger con un nombre específico, deshabilita controladores heredados, configura un FileHandler para escribir registros en un archivo con un nombre derivado de la variable nombre, agrega un formato de registro y registra excepciones con un nivel de gravedad SEVERE, cerrando el FileHandler al final. En caso de errores durante la configuración o escritura del registro, se imprime la traza de la excepción en la salida estándar y el programa se cierra.

1. 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.

R//: Lo que hicimos es que siempre que haga catch a una exception antes de lanzarla, la guarde en record.

1. 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?

R//:

* El mensaje que salió en la pantalla es 
* La aplicación no termina
* La información que tiene son los errores generados: 

1. ¿Es adecuado que la aplicación continúe su ejecución después de sufrir un incidente como este? ¿de qué dependería continuar o parar?

R//: En este caso si porque no es conveniente que se termine la ejecución por una búsqueda invalida, para que se decida para debe ser un error mucho mas importante.

1. Modifiquen la aplicación para garantizar que **SIEMPRE** que haya un error se maneje de forma adecuada. ¿Cuál fue la solución implementada?

R//: Para garantizar que siempre que haya un error se maneje de forma adecuada hicimos que en el catch se espere cualquier tipo de excepción, usando la clase Exception, y se muestre el mensaje correspondiente al usuario.

**Consultando por patrones. ¡Ahora si funciona!**

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

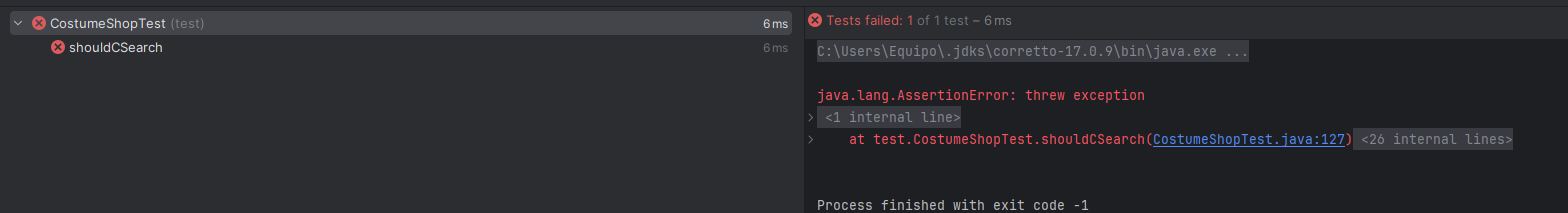
## (NO OLVIDEN BDD - MDD)

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?

R//: El método responsable en presentación es actionSearch(), y el método responsable en la capa de dominio es search.

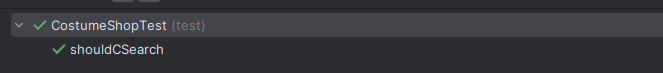
1. Realicen ingeniería reversa para la capa de dominio para **buscar**. Capturen los resultados de las pruebas. Deben fallar.



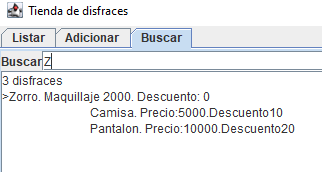


1. ¿Cuál es el error? Soluciónenlo. Capturen los resultados de las pruebas.

R//: El error es que no se ha inicializado en arraylist answers en el método select, que usa search.



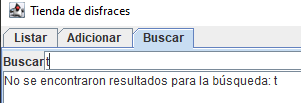
1. Ejecuten la aplicación nuevamente con el caso propuesto. ¿Qué tenemos en pantalla? ¿Qué información tiene el archivo de errores?

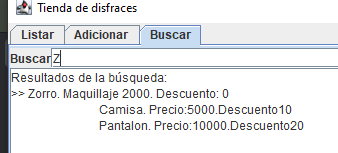
R//:  En pantalla tenemos que ya se buscan los disfraces, y el archivo de errores continua con los errores antiguos, no se generaron nuevos.



1. Refactorice la funcionalidad para que sea más amable con el usuario. ¿Cuál es la propuesta? ¿Cómo la implementa?

Nuestra propuesta es que si hay un error de búsqueda, por ejemplo que no se encuentre ningún resultado, muestre mensajes mas amigables y claros. La implementamos cambiando el código de actionSearch.





## **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?

1. Considerando las prácticas XP del laboratorio. ¿cuál fue la más útil? ¿por qué?
2. ¿Cuál consideran fue el mayor logro? ¿Por qué?
3. ¿Cuál consideran que fue el mayor problema técnico? ¿Qué hicieron para resolverlo?
4. ¿Qué hicieron bien como actividades? ¿Qué se comprometen a hacer para mejorar los resultados?