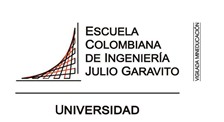
****

**Escuela Colombiana de Ingeniería Julio Garavito**

**Programación Orientada a Objetos 2024-2**

**Laboratorio IV**

**Cristian Santiago Pedraza Rodríguez**

**Andersson David Sánchez Méndez**

**25 de octubre de 2024**

# 

## PROGRAMACIÓN ORIENTADA A OBJETOS

**Excepciones 2024-2**

**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

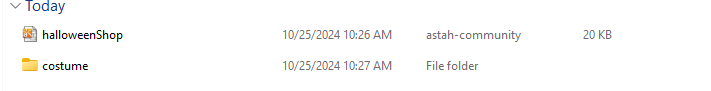
### PRACTICANDO MDD y BDD con EXCEPCIONES

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

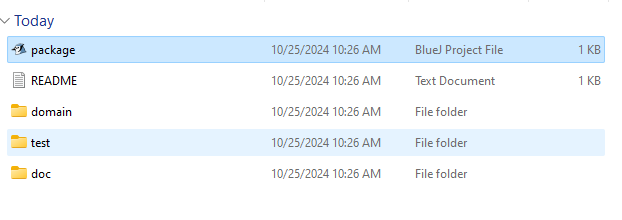
En este punto vamos a aprender a diseñar, codificar y probar usando excepciones. Para esto se van a trabajar algunos métodos de la clase Complete

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).

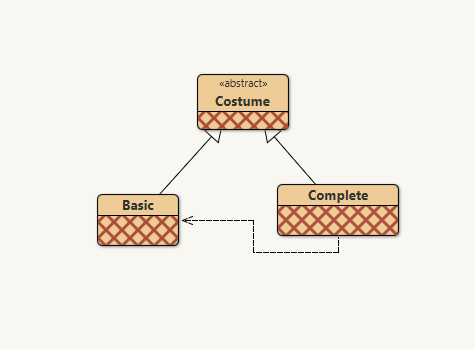




La estructura del proyecto consta de tres capas, domain, test y doc.



Paquete de domain

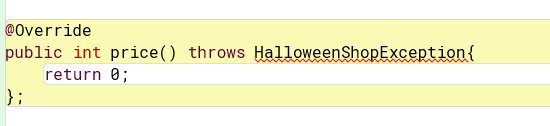


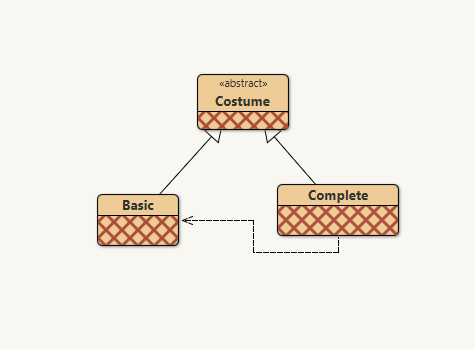
Paquete de test



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

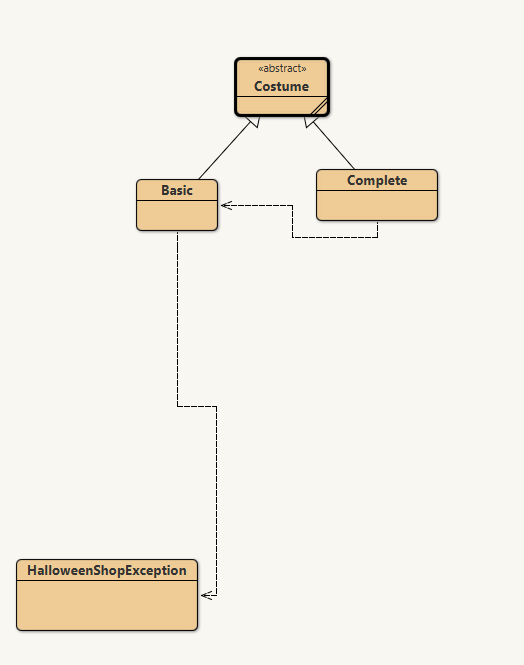
El proyecto no compila dado que falta la implementación de la clase de excepción HalloweenShopException.

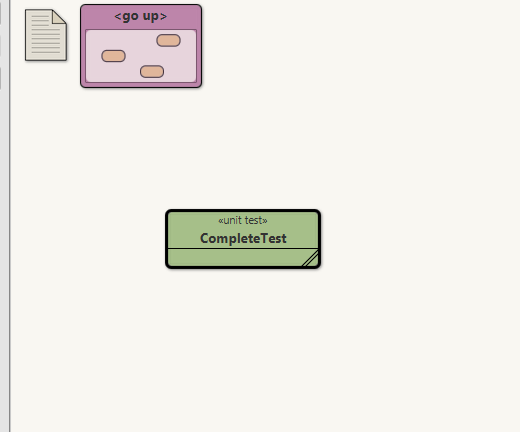




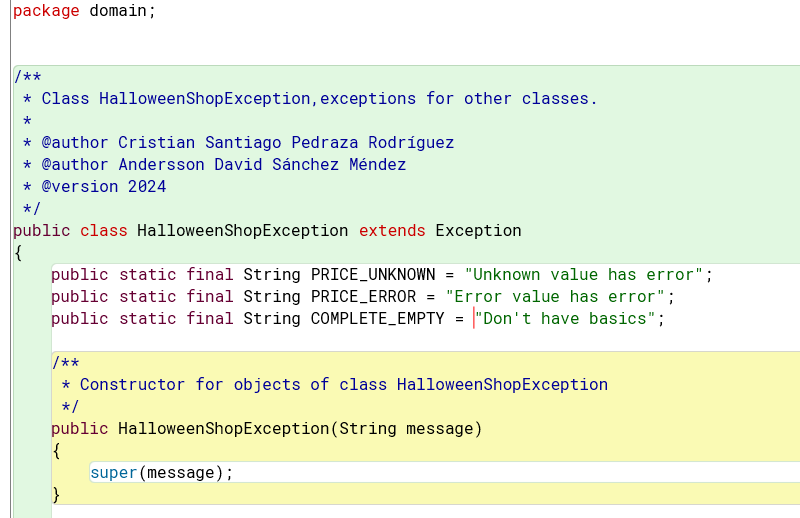


Tras la implementación de la clase exception junto con los respectivos mensajes en la clase exception. Ahora el proyecto compila sin problemas en su totalidad.

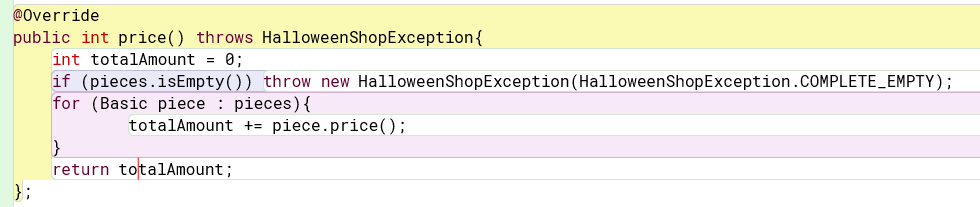




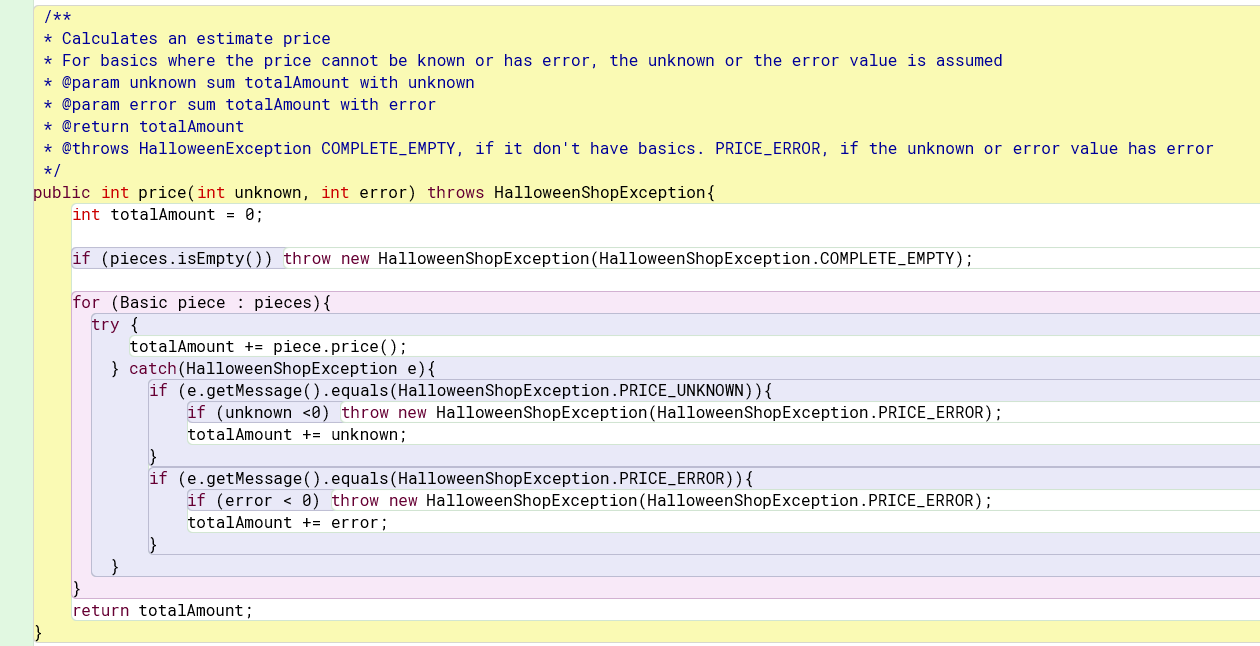
Esta es la implementación de la clase de exceptions



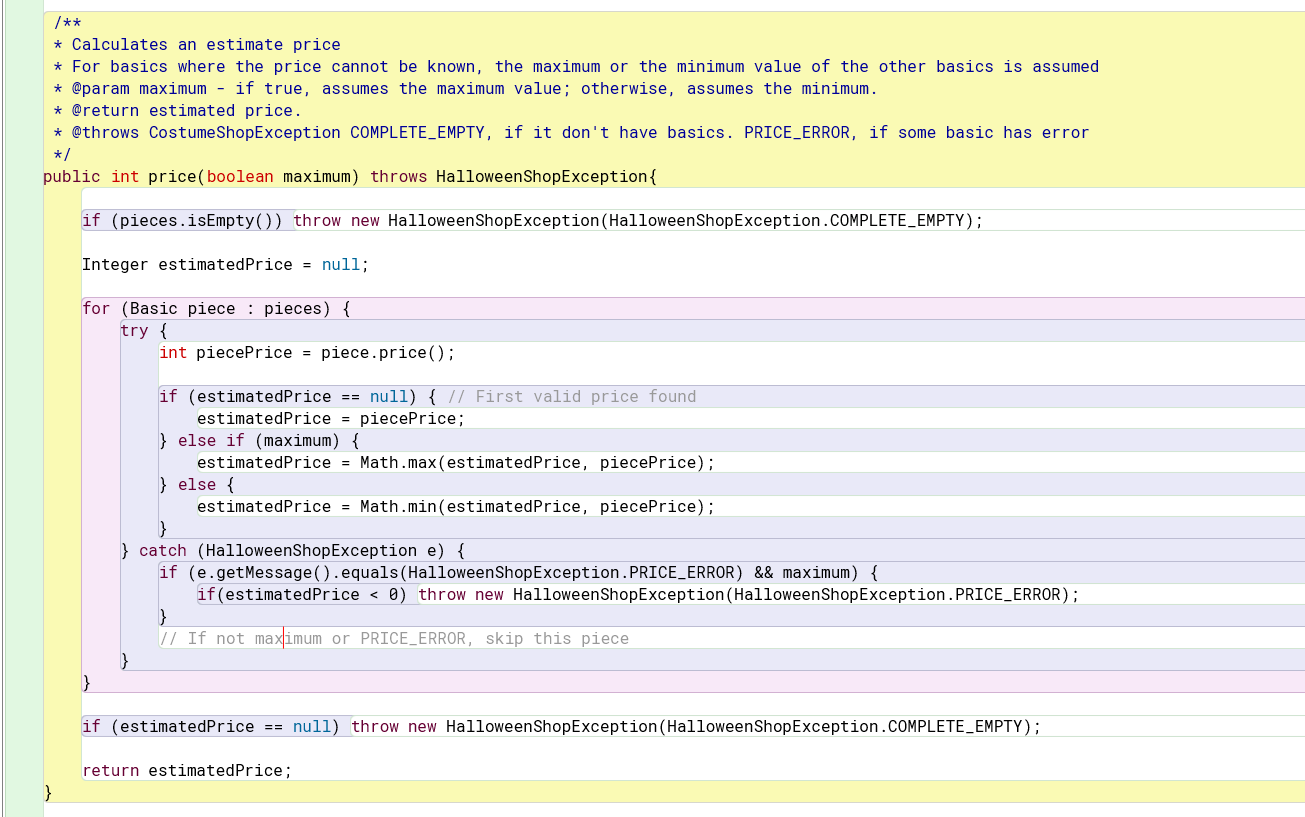
1. Dado el diseño y las pruebas, documenten y codifiquen el método price().



1. Dada la documentación y el diseño, codifiquen y prueben el método price(unknown,error).



1. Documenten, diseñen, codifiquen y prueben el método price(maximum).



**CostumeShop**

# EN CONSOLA

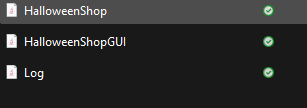
El objetivo de esta aplicación es mantener un catálogo de los disfraces ofrecidos por una tienda en Halloween en el proyecto CostumeShop. En este proyecto se ofrecen diferentes tipos de disfraces: básicos y completos.

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

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

1. En su directorio descarguen los archivos contenidos en halloweenShop.zip, revisen el contenido.
2. **¿Cuántos archivos se tienen?**

Se tienen 3 archivos .java



1. ¿Cómo están organizados?

La única “estructura” que hay es que esos archivos están en una carpeta llamada HalloweenShop, no respeta directorios, los archivos están sueltos.

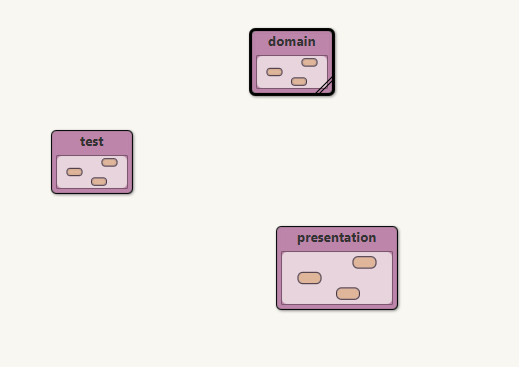


1. ¿Cómo deberían estar organizados?

Debería estar organizado de acuerdo a directorios, bin(.class), src(.java) y docs(.html)

1. Estudien el diseño del programa: diagramas de paquetes y de clases.
2. ¿Cuántos paquetes tenemos?

Existen tres paquetes, domain, presentation y test



1. ¿Cuántas clases tiene el sistema?

Ofrece 3 clases, HalloweenShop, HalloweenShopGUI,Log de halloweenShop.zip y de costumes.zip ofrece 3 también, Costume,Complete,Basic, y la clase de Prueba CompleteTest.



1. ¿cómo están organizadas?

halloweenShop.zip :HalloweenShop y Log están en el paquete de domain, mientras que HalloweenShopGUI está en el paquete de presentation.

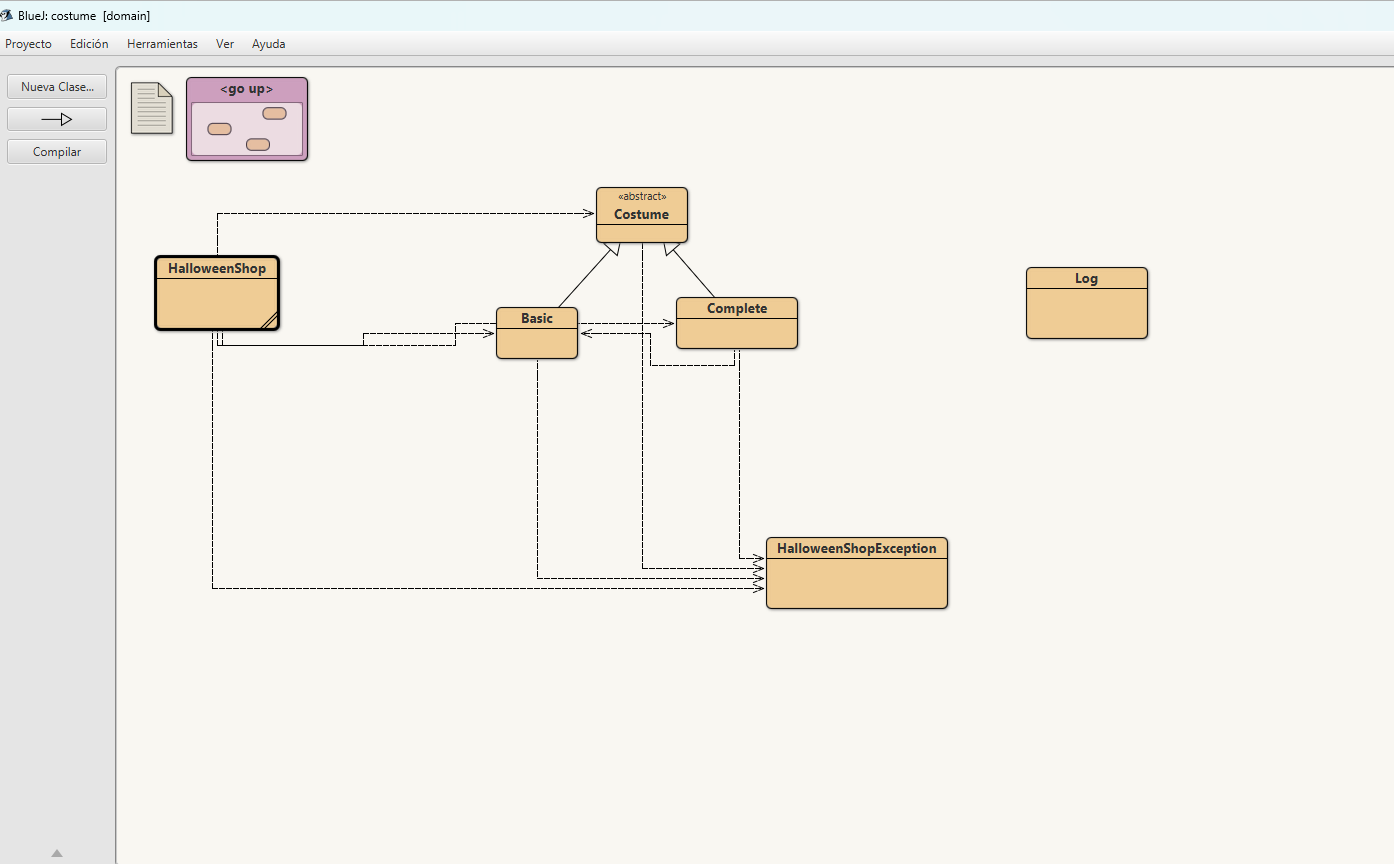
costumes.zip: Costume,Complete,Basic están el paquete domain y en el paquete test CompleteTest.

1. ¿Cuál es la clase ejecutiva?

La clase ejecutiva que se encarga de mostrar la visualización de todo lo que se implemente es HalloweenShopGUI, porque este es el que da la interfaz gráfica a los disfraces.

1. Prepare los directorios necesarios para ejecutar el proyecto.

Después de organizar los archivos .java de halloweenShop.zip, se obtiene en domain esta estructura en BlueJ

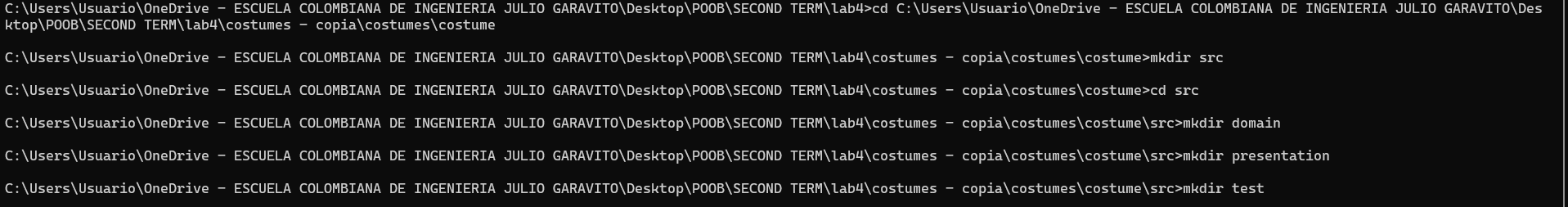


1. ¿Qué estructura debe tener?

La estructura definitiva es convertir el archivo blueJ como archivo.jar para que solo se compile un solo archivo y no todos los .class o .java.

Así que, usando cmd se procede a organizar esto con la estructura de bin,docs y src

Crear directorios y extraer los archivos .java para pasarlos de acuerdo al paquete al directorio src y pasar archivos .class a bin y a docs toda la documentación, se obtiene:

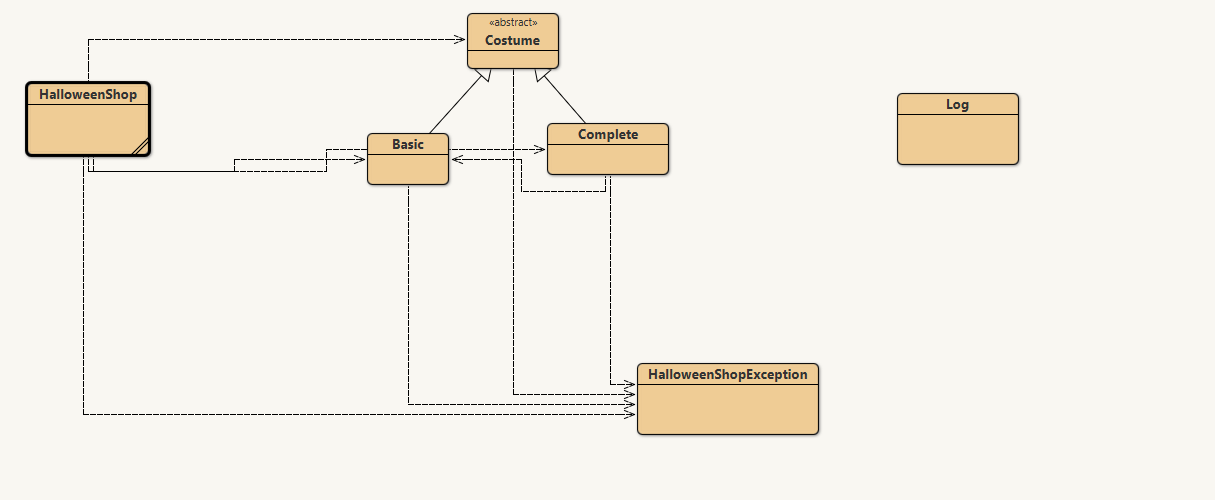






1. ¿Qué clases deben tener?

Debe tener la recopilación de los dos archivos .zip





1. ¿Dónde están esas clases?



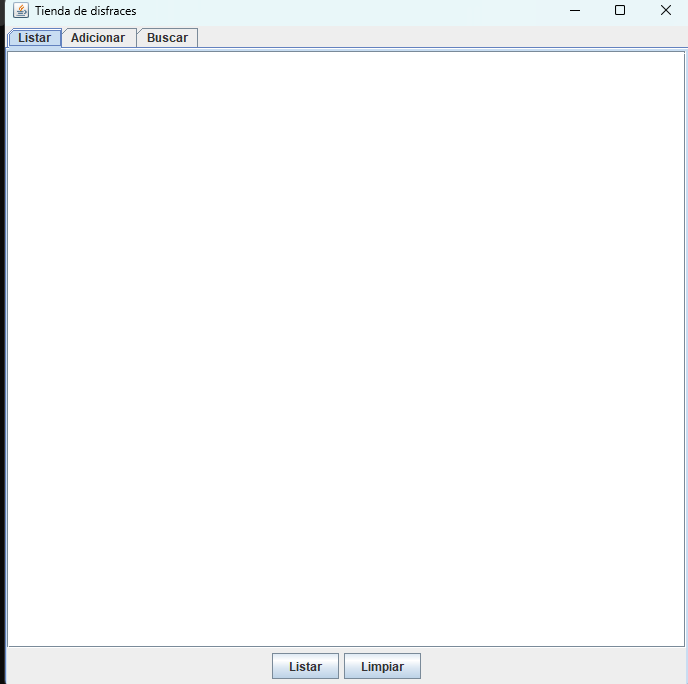
Para ejecutar el programa es necesario ir a HalloweenShopGUI que es donde se va a ver toda la interfaz del programa, esta clase está en el paquete presentation. Y esas clases van a estar en el directorio bin(.class).

1. ¿Qué instrucciones debe dar para ejecutarlo?

Para ejecutar es necesario escribir el comando: java presentation.HalloweenShopGUI

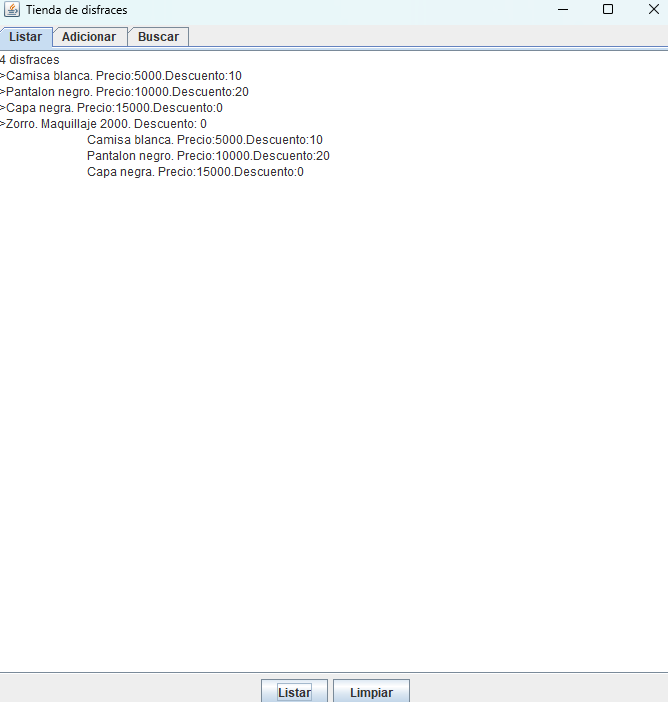


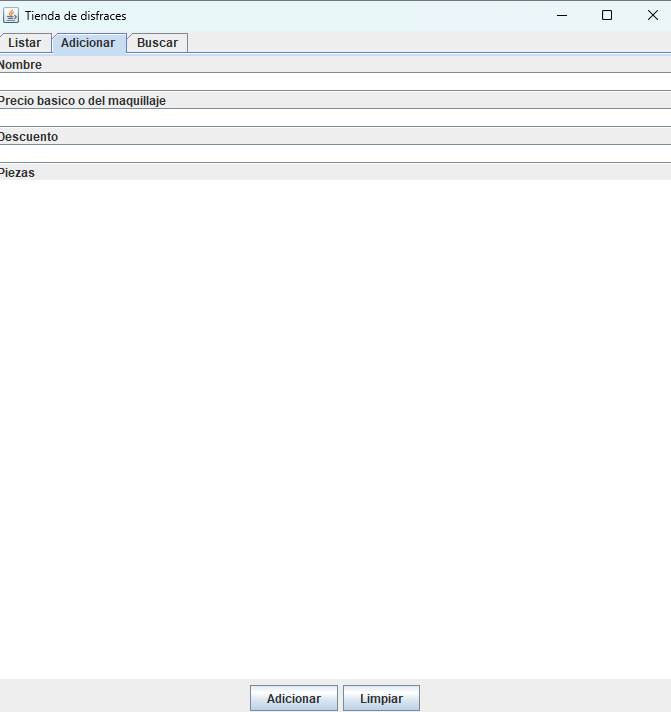
Para que se genere la interfaz gráfica:

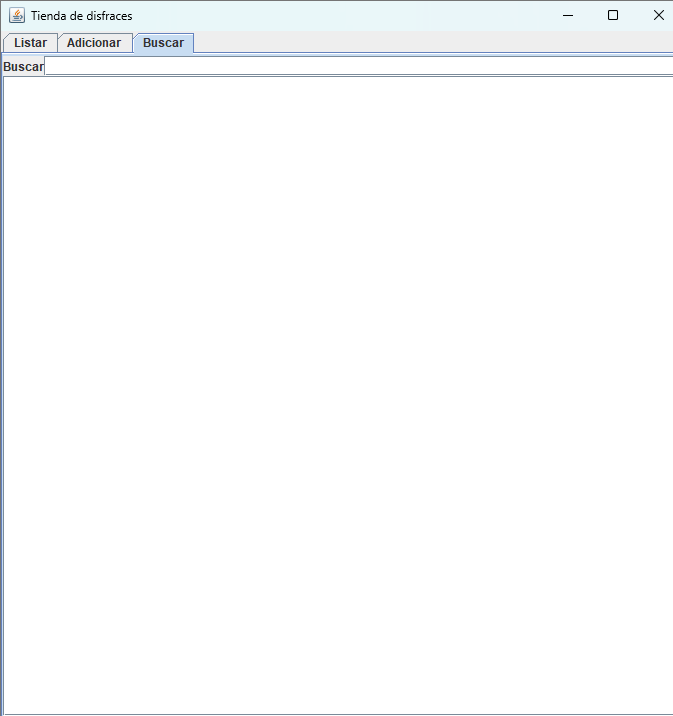


1. Ejecute el proyecto,
2. ¿Qué funcionalidades ofrece?

La tienda de disfraces ofrece opciones para listar disfraces, para limpiar esa lista, para adicionar, y para buscar algún disfraz.



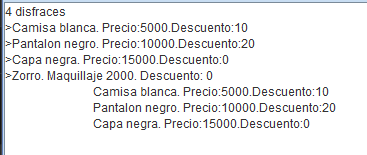




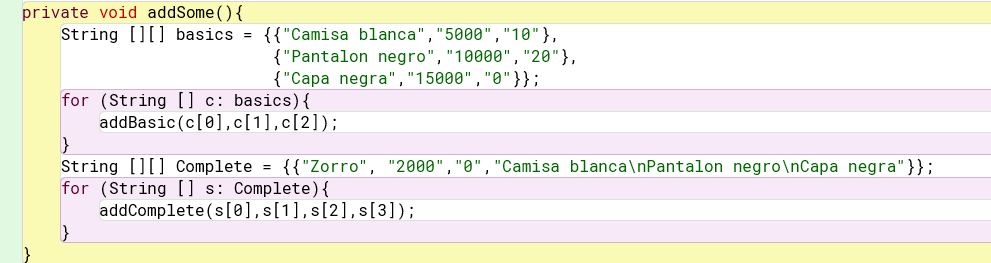
1. ¿Cuáles funcionan?

Ninguna funcionalidad todavía funciona, es decir, están los recuadros para cada función, pero se intenta agregar un nuevo disfraz y no deja, o se busca y no aparecen búsquedas acerca de la lista de disfraces que ya están propuestos.

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



Estos disfraces iniciales salen de darle en el botón de listar, revisando código y documentación, nos damos cuenta de que en el método addSome() se añaden los disfraces de la clase HalloweenShop



1. ¿Qué clase pide que se adicionen?

La clase que pide la adición de disfraces es HalloweenShop. En particular, dentro de addSome(), se utiliza el método addBasic para agregar disfraces básicos y addComplete para disfraces completos.

1. ¿Qué clase los adiciona?

Los disfraces son realmente añadidos por HalloweenShop. Los métodos addBasic y addComplete se implementan en esta clase, permitiendo la creación y almacenamiento de nuevos disfraces básicos y completos en las colecciones costumes y basics.

**Adicionar y listar. Todo OK.** [En lab04.doc, CostumeShop.asta y \*.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

Pantalon rojo 30000 , 25

*Completo*

Diablo 25000, 50

Pantalon rojo Capa negra

1. ¿Qué ocurre?

El requerimiento es añadir dos nuevos disfraces

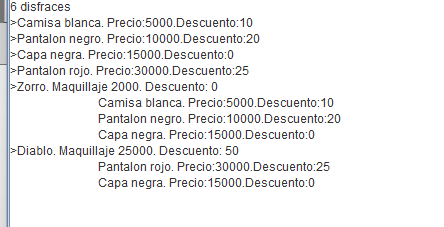
* Disfraz básico: Añadir un básico de Pantalón rojo con precio, 30000 y descuento de 25%.
* Disfraz completo: se va a llamar Diablo y va a tener un precio de 25000 con un descuento del 50%, donde tiene el pantalón rojo, y una capa negra

Esto es en código:



1. ¿Cómo lo comprueban? Capturen la pantalla.

Se comprueba ejucutando HalloweenShopGUI de presentation, y cuando se lista, se ve los nuevos disfraces.



1. ¿Es adecuado este comportamiento?

Sí, este comportamiento es adecuado si ambos disfraces se muestran correctamente con su nombre, precio, descuento, y piezas; esto se cumple a la perfección.

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

En la capa de presentation, es decir, en la clase HalloweenShopGUI se evidencia que el método responsable de añadir es actionAdd(). Se activa cuando se presiona el botón "Adicionar" en la interfaz gráfica, y decide si agregar un disfraz básico o completo en función del contenido del campo basics.

1. ¿Qué método en la capa de dominio?

En la capa de dominio, los métodos responsables de agregar nuevos disfraces son addBasic() para disfraces básicos y addComplete() para disfraces completos. addSome() solo se utiliza para inicializar datos predeterminados y está relacionado indirectamente con la adición de disfraces.

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.
2. ¿Qué método es responsable en la capa de presentación?

En la capa de presentation, es decir, en la clase HalloweenShopGUI se evidencia que el método responsable de listar es actionList(). Este método convierte en formato en texto lo que recibe de toString() de la otra clase para cuando se oprima listar, se vean todas las opciones que hay de disfraces.

1. ¿Qué método en la capa de dominio?

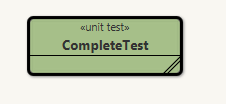
En la capa de dominio, el método responsable de listar es el método toString() en la clase HalloweenShop. Este método llama a data(ArrayList<Costume> selected), que construye el texto que se muestra en el área de resultados de la interfaz gráfica. toString() usa todos los disfraces en la colección costumes y pasa esa lista a data(), que retorna un formato de texto listo para ser mostrado.

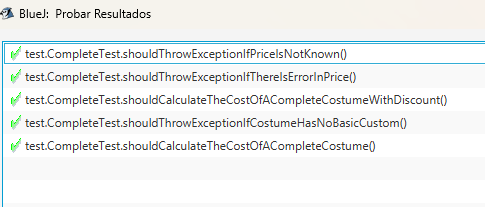
1. Realicen ingeniería reversa para la capa de dominio para **listar**. Capturen los resultados de las pruebas de unidad.





1. Propongan y ejecuten una prueba de aceptación.









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

(NO OLVIDEN BDD – MDD)

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

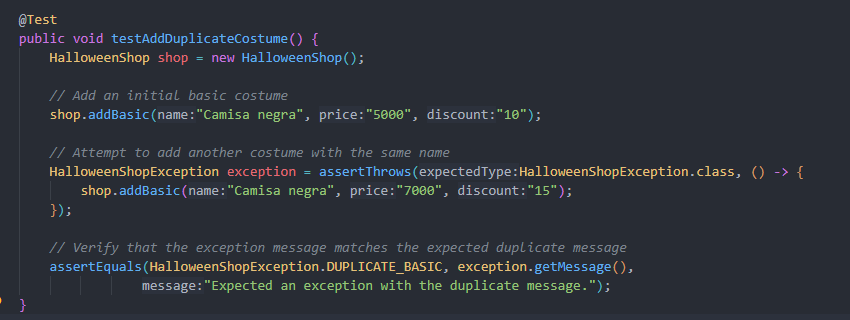
### Para cada uno de los siguientes casos realice los pasos del 1 al 4.

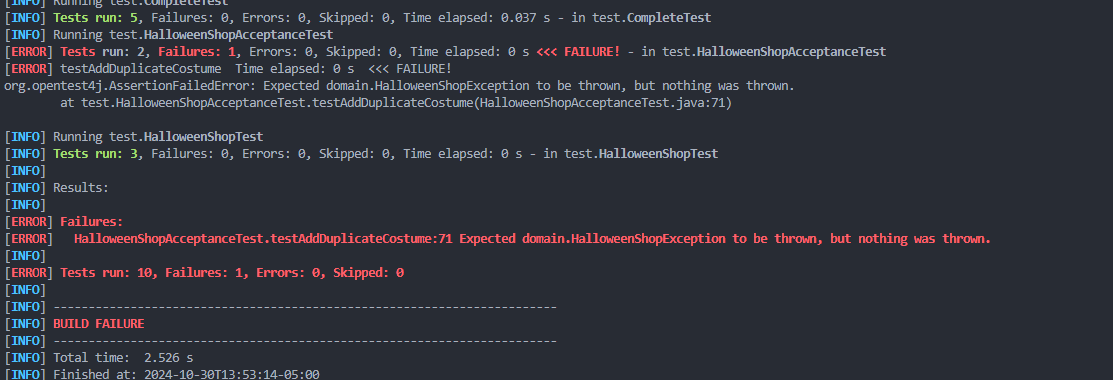
* 1. **¿Y si el nombre del disfraz ya existe?**

1. Propongan una prueba de aceptación que genere el fallo.

De hecho, no sucede absolutamente nada a nivel de excepciones no manejadas en el código, dado que no se valida que dos disfraces no puedan tener el mismo nombre. Esto quiere decir que no se genera fallo como tal.

Por esto se propone una prueba que espere una excepción a ser lanzada. Pero no funciona dado que actualmente no se lanza ninguna excepción





1. 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.

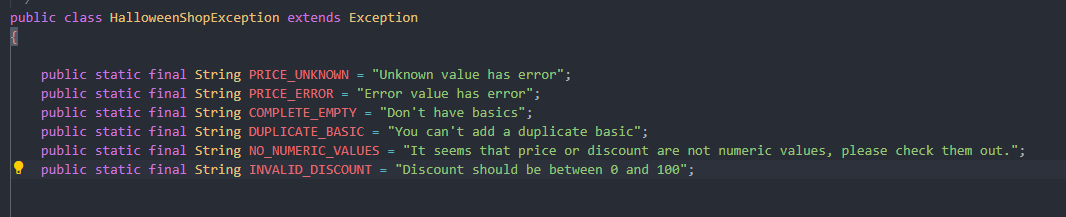
El método que debería lanzar la excepción es addBasic. Dado que ella es la encargada de añadir piezas basicas y a ella recurre addComplete posteriormente.

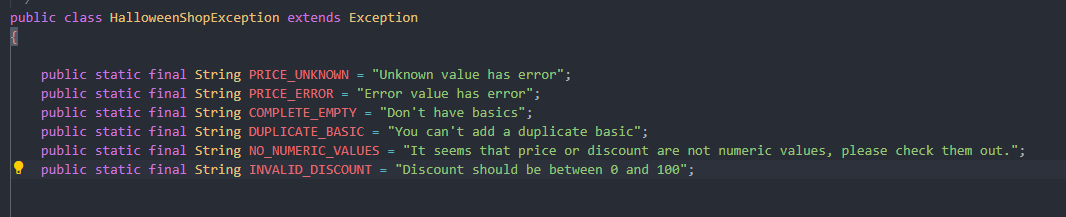
Ahora, para propagarla, debe ser addComplete, dado que si al hacer un addComplete, ella recibe una excepción de este tipo, debe propagarla para que el que está invocando este método sea el encargado de atenderla.

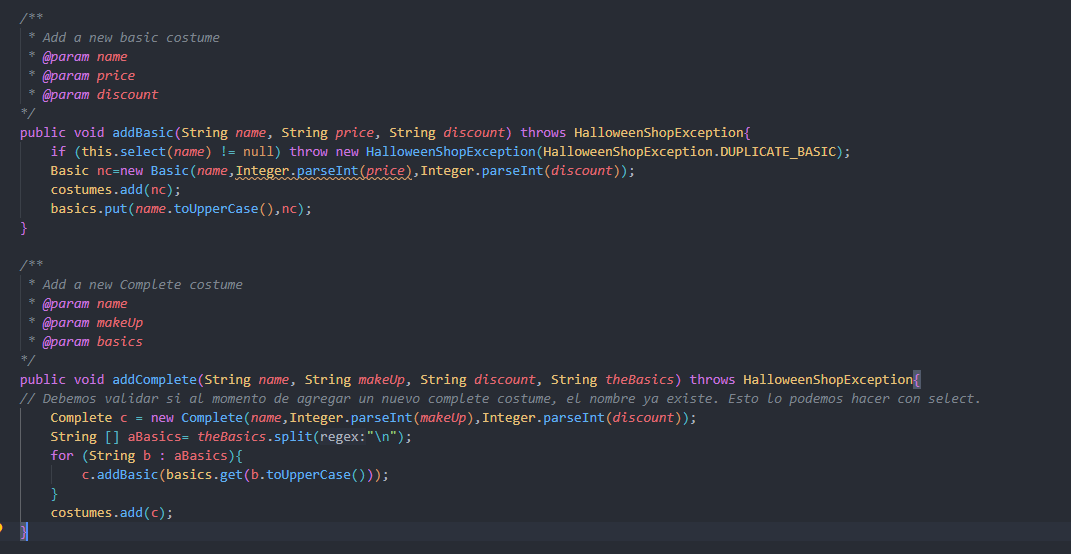
El encargado de atenderla será HalloweenShopGUI dado que es desde allí en donde podemos hacer una validación de los casos exitosos o no satisfactorios de los métodos. Así como addSome, también deberá atenderla

1. Construya la solución propuesta. Capture los resultados de las pruebas de unidad.

Lanzamiento y propagaciones de la excepción



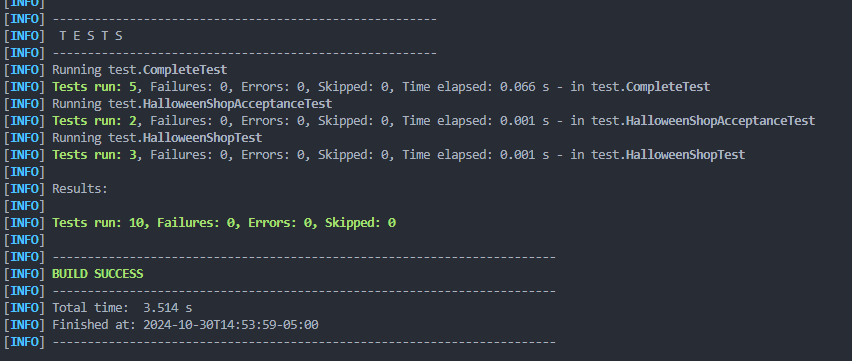




Atención de la excepción



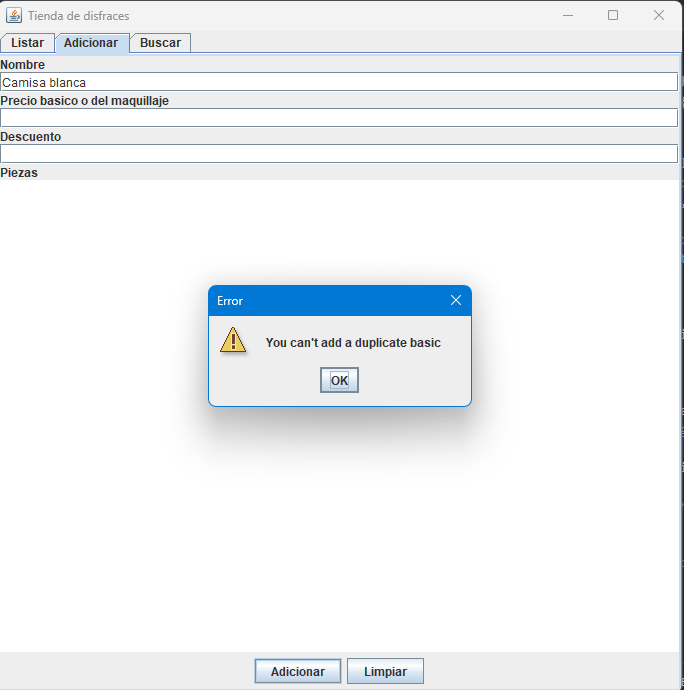




Ahora, el total de las pruebas son exitosas, dado que ahora se valida que no se puedan agregar disfraces repetidos. Lanzando una excepción si se llegara a dar el caso.

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

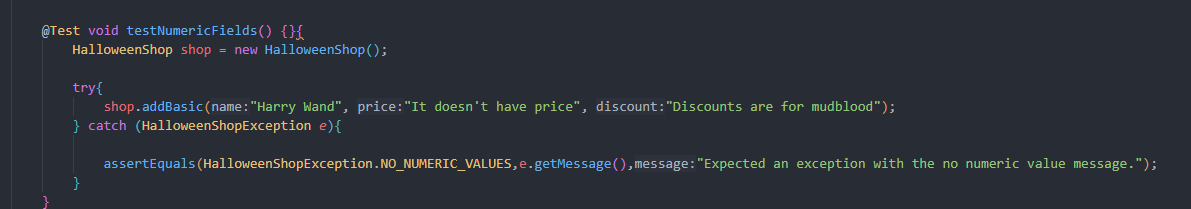
A la hora de ejecutar podemos ver que se lanza el JOPtionpane de forma correcta.

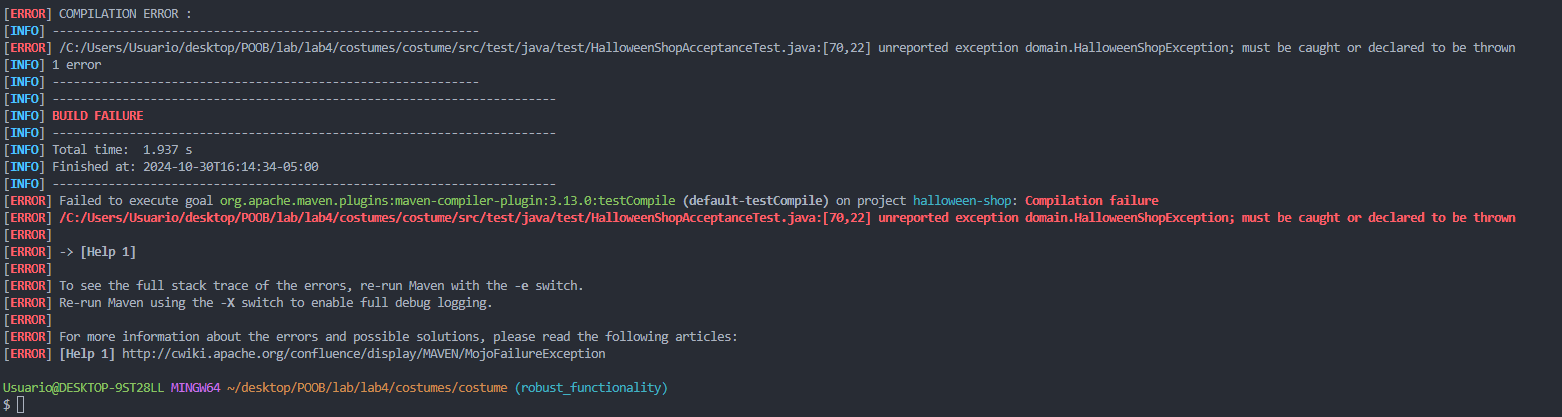


* 1. **¿Y si los valores numéricos no son numéricos?**

1. Propongan una prueba de aceptación que genere el fallo.

Esperamos una excepción de tipo no valores numéricos.

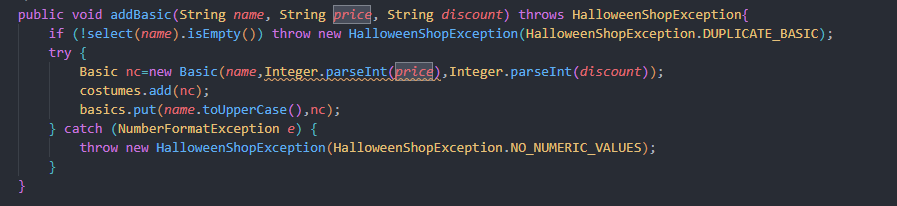


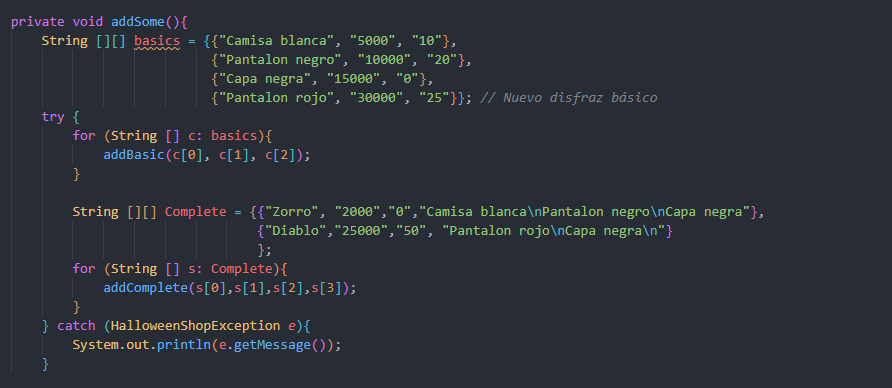


1. Analicen el diseño realizado. Para hacer el software robusto: ¿Qué método debería lanzar la excepción? ¿Qué métodos deberían propagarla? ¿Qué método debería atenderla? Explique claramente.

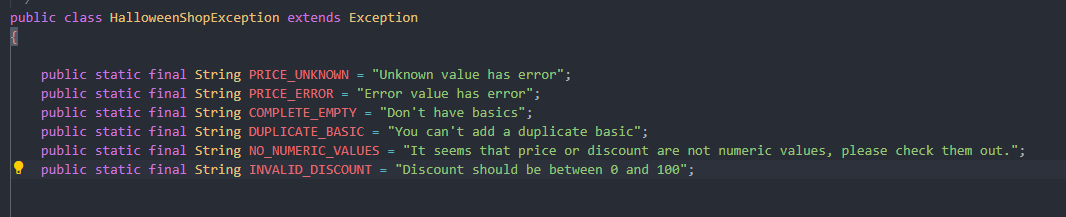
En addBasics, se debería lanzar la excepción no valores numéricos que compruebe que deben ser instancia de entero.

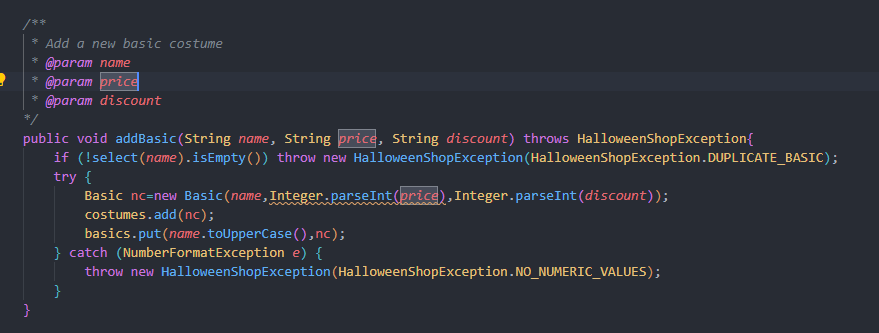
Ahora, la debería propagar addComplete y la deben manejar HalloowenShopGUI y addSome

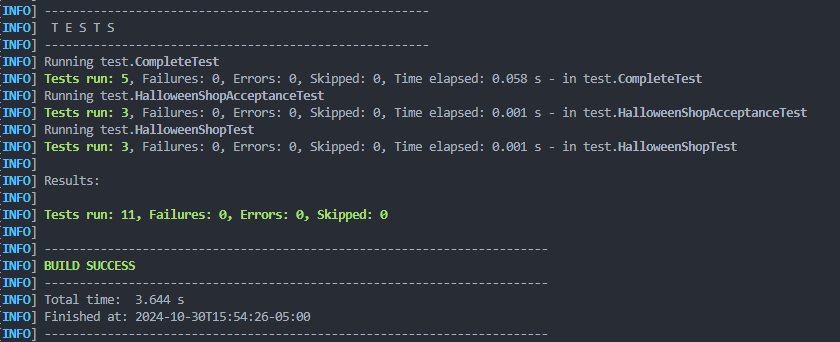




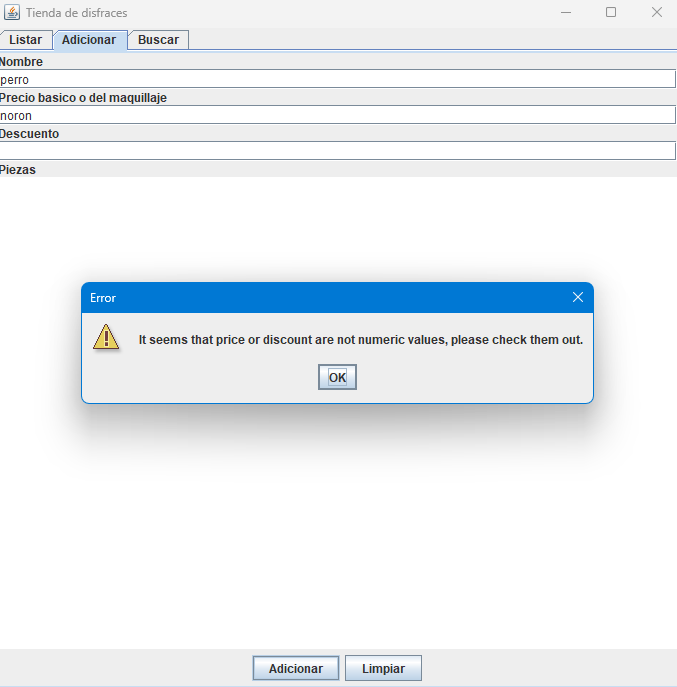
1. Construya la solución propuesta. Capture los resultados de las pruebas de unidad.





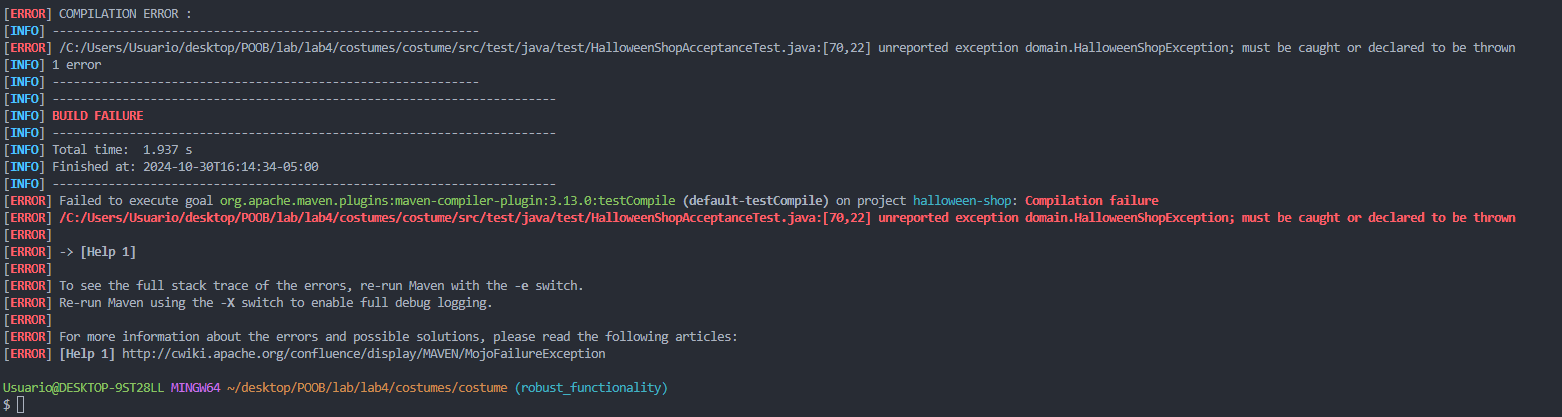


1. Ejecuten nuevamente la aplicación con el caso de aceptación propuesto en 1. ¿Qué sucede ahora? Capture la pantalla.



* 1. **¿Y si el descuento no está entre 0 y 100?**

1. Propongan una prueba de aceptación que genere el fallo.





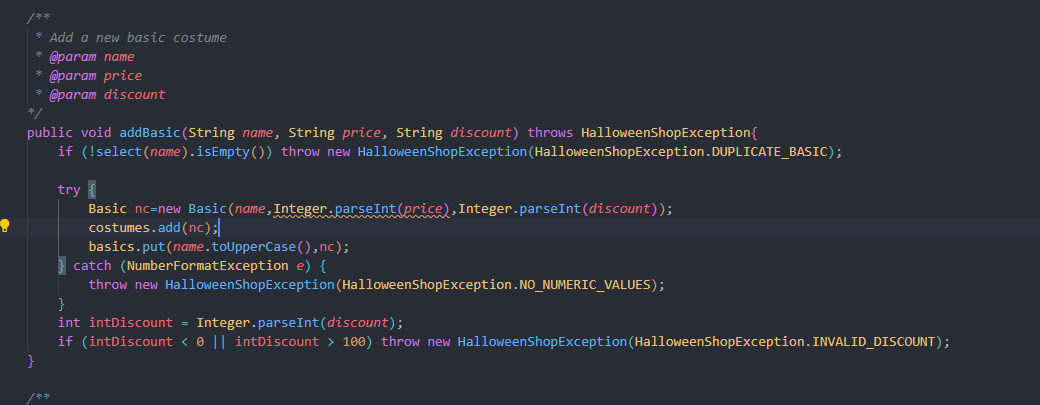
1. Analicen el diseño realizado. Para hacer el software robusto: ¿Qué método debería lanzar la excepción? ¿Qué métodos deberían propagarla? ¿Qué método debería atenderla? Explique claramente.

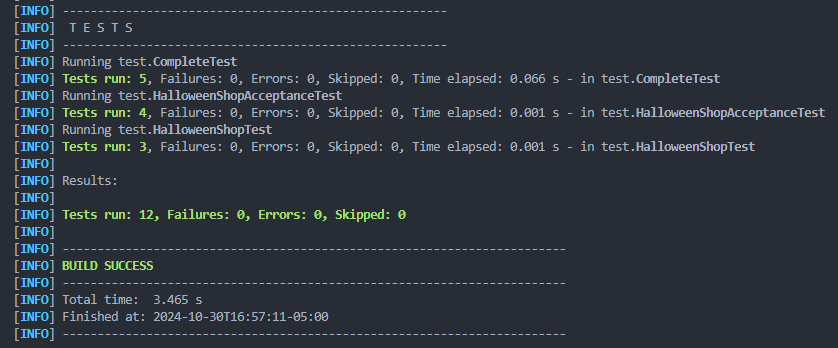
En addBasics, se debería lanzar la excepción al validar si no está en el rango de 0 a 100.

Ahora, la debería propagar addComplete y la deben manejar HalloowenShopGUI y addSome

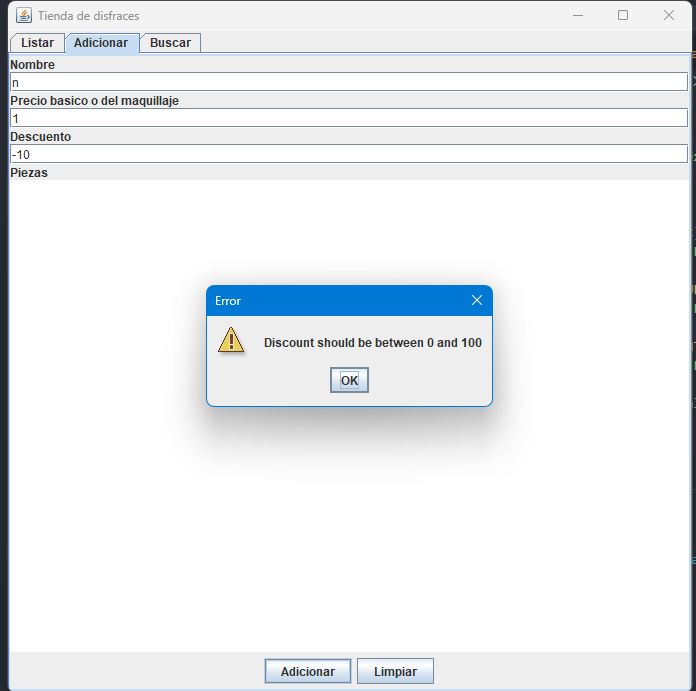
1. Construya la solución propuesta. Capture los resultados de las pruebas de unidad.







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

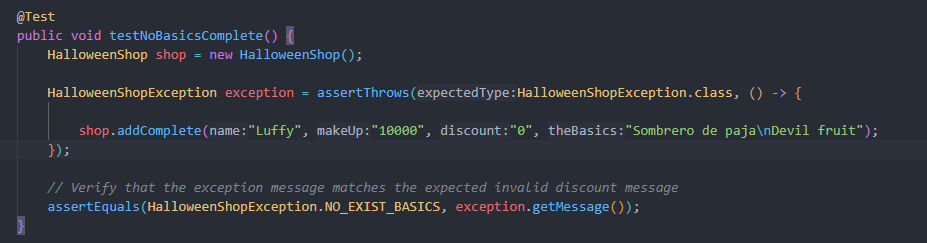


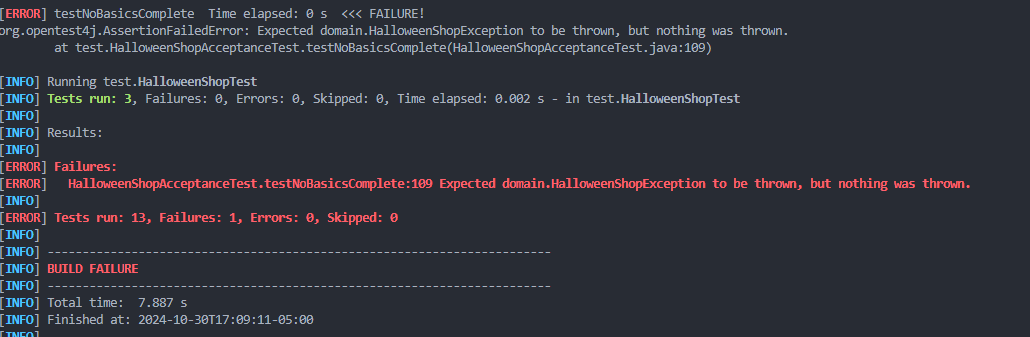
Ahora se valida que el valor debe estar entre 0 y 100, si nó, al usuario se le da un mensaje con JOptionpane

* 1. **Proponga una nueva condición (¿Y si al adicionar un disfraz completo, no existen sus piezas básicas?)**

1. Propongan una prueba de aceptación que genere el fallo.

Esperamos una excepción del tipo no valid basics pero recibiremos null pointer.

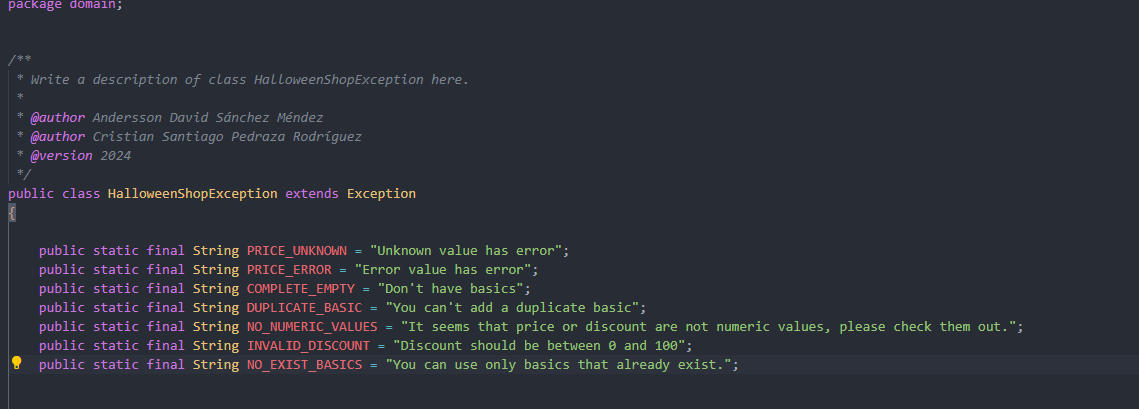


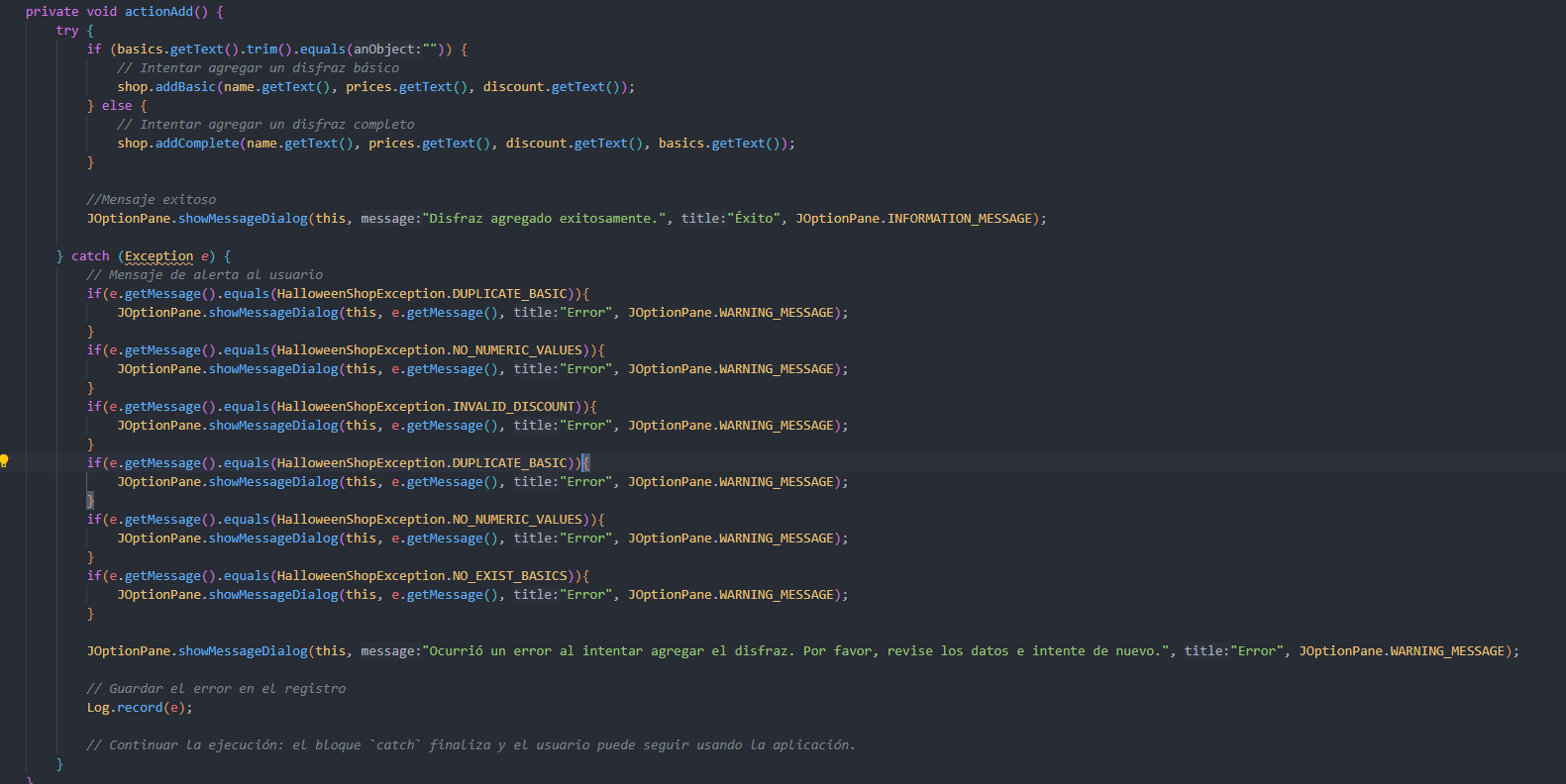


1. Analicen el diseño realizado. Para hacer el software robusto: ¿Qué método debería lanzar la excepción? ¿Qué métodos deberían propagarla? ¿Qué método debería atenderla? Explique claramente.

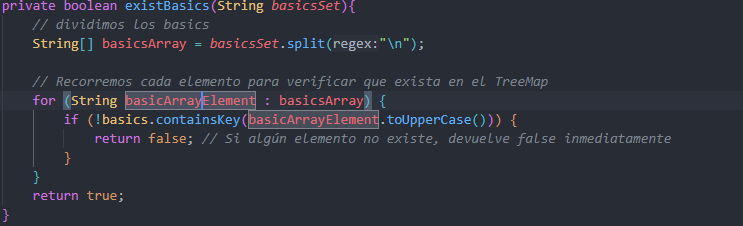
addComplete debe encargarse de lanzar esta excepción, dado que el tiene la responsabilidad de validar si ya existen dichos básicos.

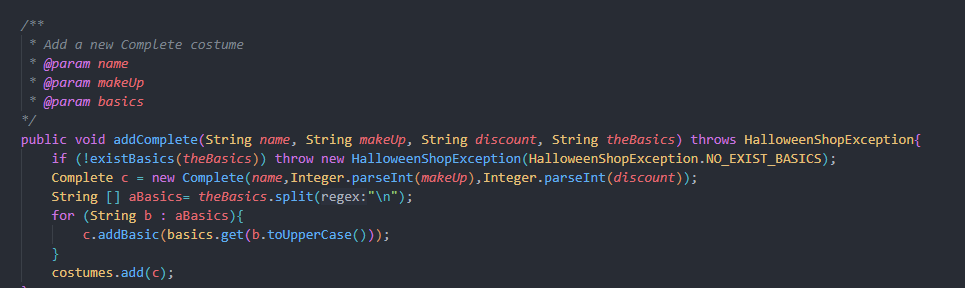
addSome y HalloowenShopGUI deben atenderla.

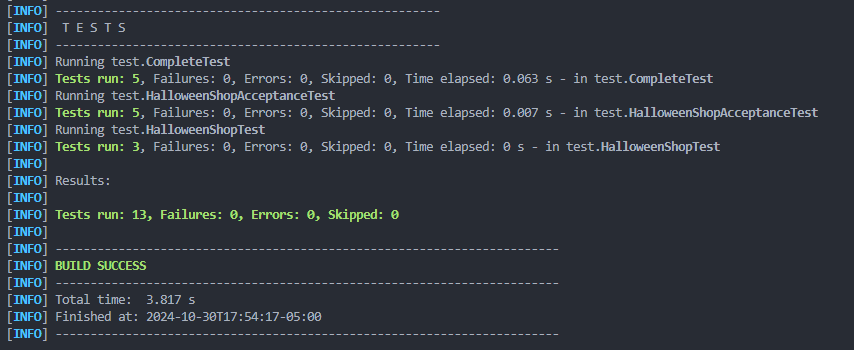
1. Construya la solución propuesta. Capture los resultados de las pruebas de unidad.
2. 



Se crea una función privada auxiliar que valida que todos los elementos existan, si al menos uno no existe. Devuelve false de inmediato.

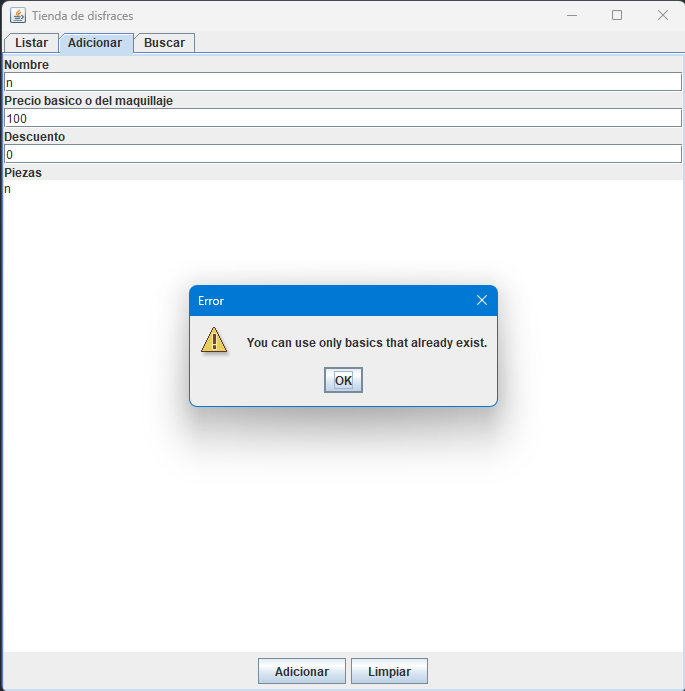






1. Ejecuten nuevamente la aplicación con el caso de aceptación propuesto en 1. ¿Qué sucede ahora? Capture la pantalla.

Antes, si añadimos un complete sin basics existentes, mandaba null pointer en consola, ahora manga un elegante y muy amigable mensaje al usuario.



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

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

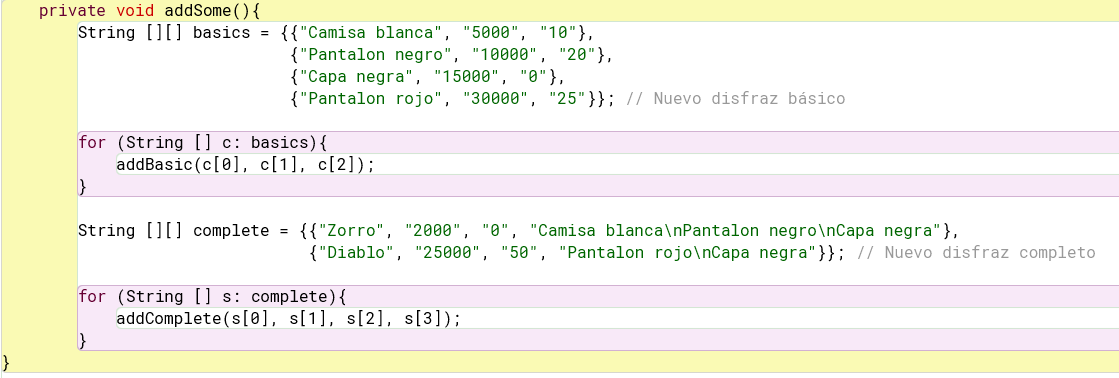
(NO OLVIDEN BDD - MDD)

1. Consulten un disfraz completo que inicie con I.
2. ¿Qué sucede?



1. ¿Qué creen que pasó? Capturen el resultado.

Lo que pasó y que debió haber pasado es que haya 6 disfraces, lo cual es verdad porque fue tal cual la cantidad de disfraces anteriores (disfraces predeterminados), 4, más el nuevo disfraz de Diablo complete y el nuevo conjunto básico de Pantalón rojo.



1. ¿Quién debe conocer y quién NO debe conocer esta información?

* **El que debe conocer:** La misma tienda de disfraces HalloweenShop debería saber qué servicios o productos está ofreciendo; como su precio base, descuento aplicado, piezas que lo componen de los disfraces para poder vender al cliente el precio final.
* **El que no debe conocer:** El cliente no debería conocer el descuento que se aplica internamente a cada producto, el cliente solo sabe el precio definitivo del disfraz, ya sea básico o completo, es decir, el cliente no debería poder relación directa con el método addSome() porque esta es de consulta internamente en la tienda.

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

Este método permite a la aplicación registrar información sobre errores críticos en un archivo(logs), lo que brinda varios beneficios:

* **Depuración:** Los desarrolladores revisan el archivo de registro para identificar y analizar los errores que ocurrieron durante la ejecución.
* **Monitoreo de la Aplicación:** Permite llevar un registro histórico de los errores, para detectar patrones de fallos o problemas recurrentes.
* **Persistencia de Errores:** Al almacenar los errores en un archivo, los datos sobre fallos persisten incluso si la aplicación se cierra, así es útil para diagnóstico y auditoría.

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.

**Explicación de la Solución**

**Try-catch:** Poner bloque try-catch alrededor del código que intenta agregar un disfraz, para manejar cualquier excepción que pueda surgir durante este proceso.

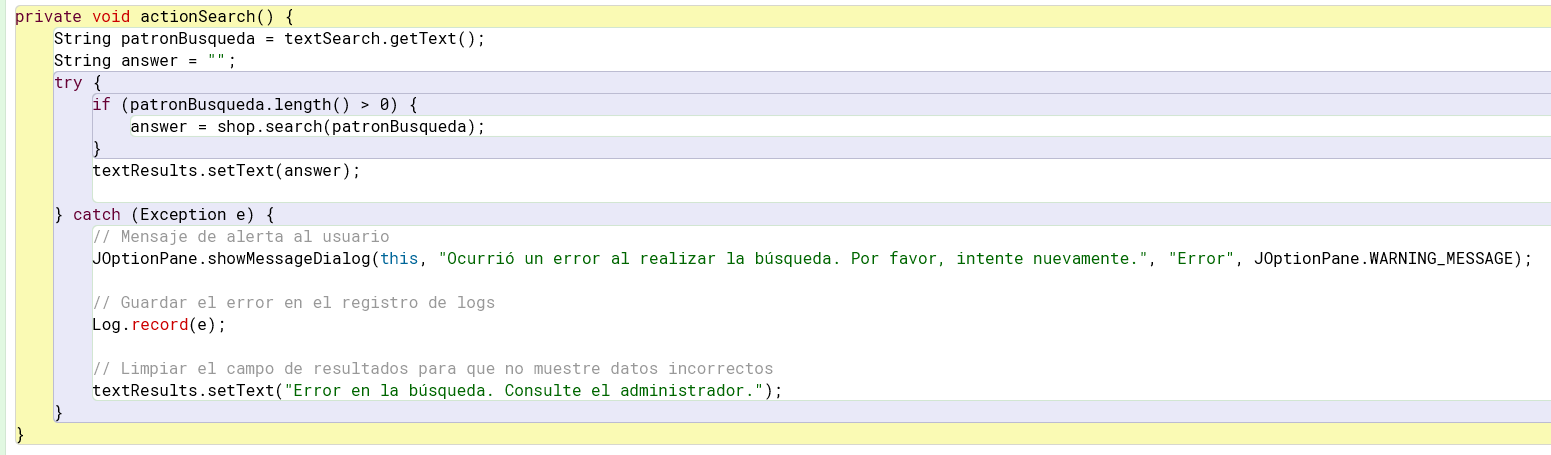
**Mostrar mensaje de alerta al usuario**: En catch, se usa JOptionPane.showMessageDialog para mostrar un mensaje de advertencia al usuario, informándole que ocurrió un problema al agregar el disfraz. Esto con el incentivo de dar un feedback al usuario sugiriendo revisar los datos, o en el otro caso, se asegura que cualquier excepción lanzada por shop.search(patronBusqueda) sea capturada.

**Registro del error:** Se llama a Log.record(e); para registrar la excepción en el archivo de log, lo cual permitirá a los desarrolladores o administradores de la aplicación revisar el error a largo plazo.

**Continuar la ejecución:** La aplicación no se detiene, ya que después del mensaje y el registro, la ejecución continúa y el usuario puede seguir interactuando con la GUI.

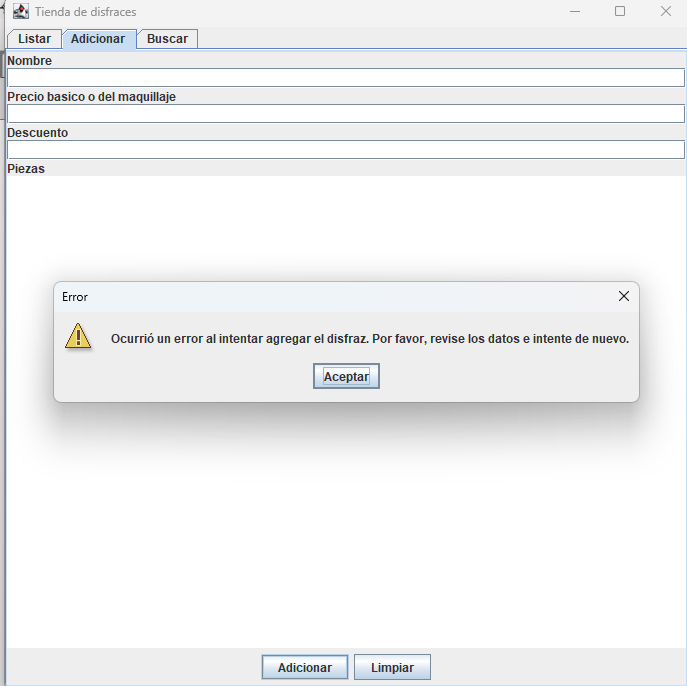
La solución se implementó directamente en la GUI, es decir, en la capa de presentation, en el método de actionAdd() y actionSearch() de HalloweenShopGUI.

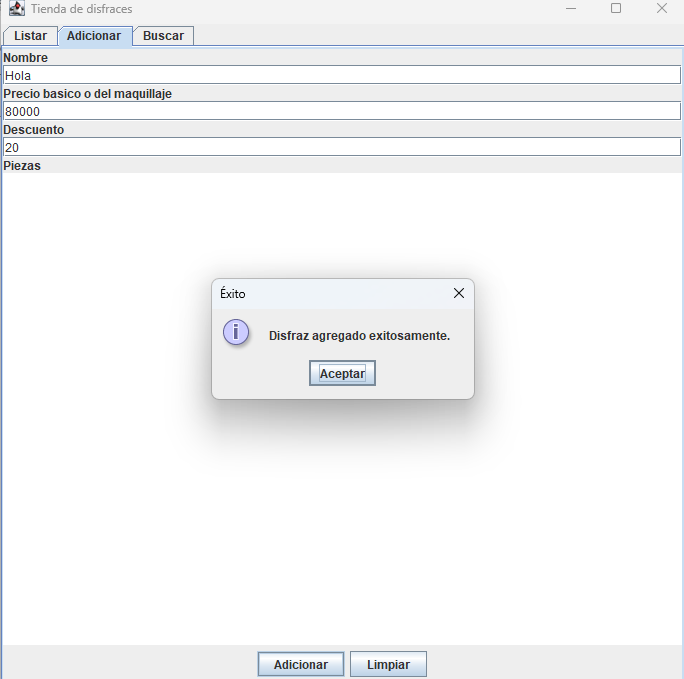




1. Ejecuten nuevamente la aplicación con el caso propuesto en 1. **Agregando actionAdd()**
2. ¿Qué mensaje salió en pantalla?

**actionAdd()**





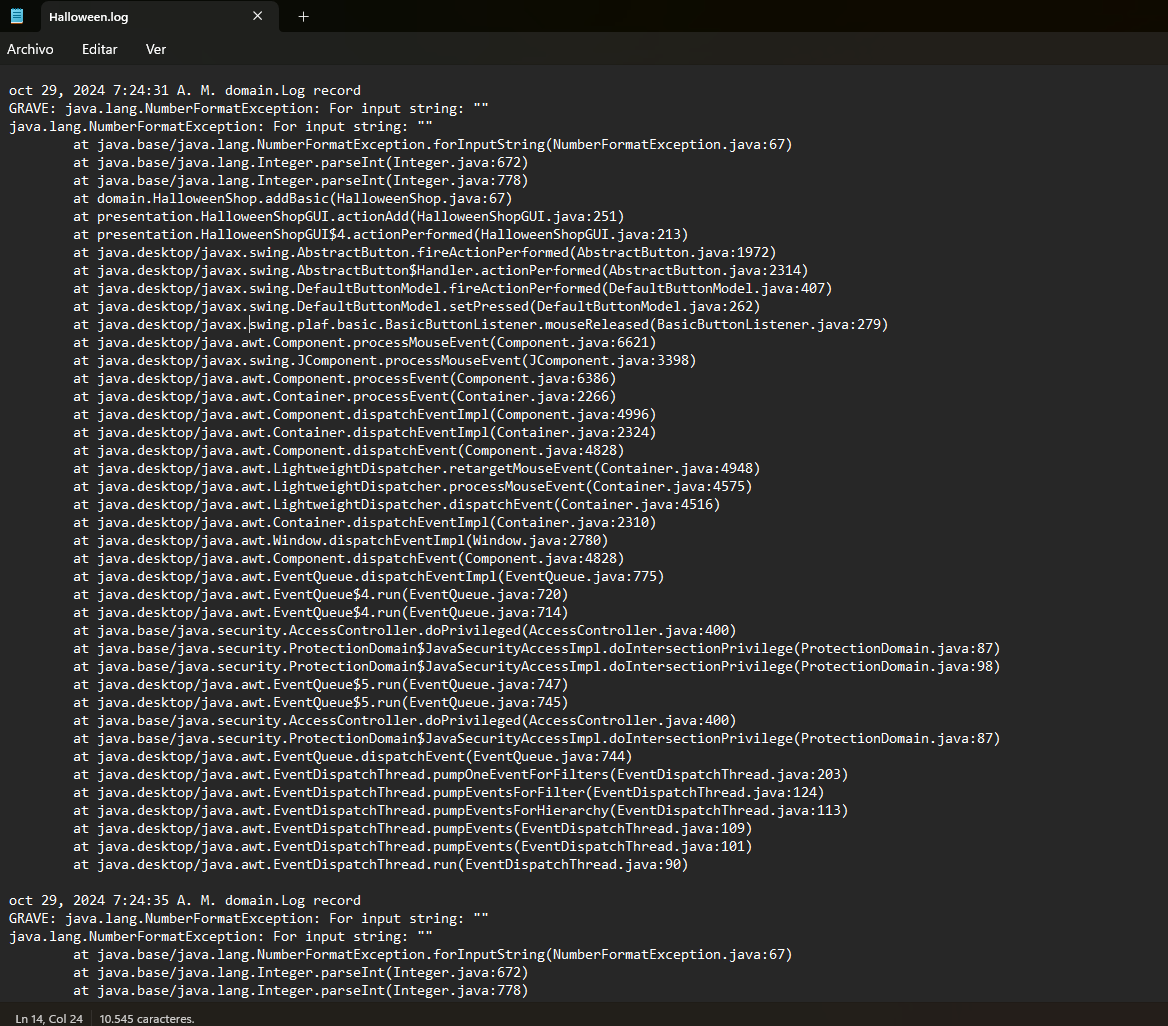
**actionSearch()**

****

1. ¿La aplicación termina?

No, la aplicación continúa ejecutándose. El bloque catch maneja la excepción, muestra el mensaje al usuario y registra el error sin interrumpir el flujo general de la aplicación.

1. ¿Qué información tiene el archivo de errores?



Este archivo Log que se crea cuando ocurre una excepción, en ese caso cuando aparece:  
  
 "Ocurrió un error al intentar agregar el disfraz. Por favor, revise los datos e intente de nuevo." o "Ocurrió un error al realizar la búsqueda. Por favor, intente nuevamente."

Estos mensajes son la base de lo que muestra el archivo .txt de Logs especificando cada información de la excepción para dar un análisis profundo de porqué ocurrió eso.

1. a) **¿Es adecuado que la aplicación continúe su ejecución después de sufrir un incidente como este?**

¿Desde la perspectiva de nosotros como usuarios, estaríamos de acuerdo que por x,y razón se nos olvide poner un dato, y nos genere el error? Sí, porque, pues nos da la visión de que algo falta, y nos dé chance de solo agregar lo que falta, pero no escribir toda la información que sí está bien, así que, consideramos que la aplicación debe seguir funcionando, pero generando los mensajes de usuario, si hay problemas inesperados.

Para aplicaciones como esta, se mantiene la interacción directa con el usuario de manera amigable si los errores se pueden corregir sin mayores riesgos, pero si son errores que comprometen la integridad de los datos, es mejor no seguir continuando la ejecución de la aplicación.

b) ¿**de qué dependería continuar o parar?**

Ya depende del software que se necesite y los requerimientos solicitados en la empresa de desarrollo o compañía, cualquiera de las dos opciones vendría bien, de acuerdo con lo acordado en la empresa y de cómo se deben manejar datos si ocurren excepciones. Por lo tanto, se establecen pautas en modo a los errores que ocurran, se definen si son prioritarios o no, y desde ahí se valida que tanta importancia tiene o no detener la aplicación.

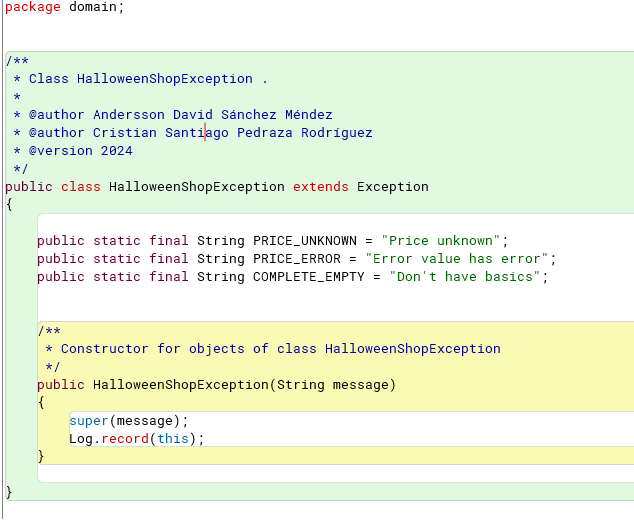
Por ejemplo, en software financiero o médico, un error puede representar un riesgo para la integridad de la información o la seguridad del usuario. En estos casos, puede ser necesario detener la aplicación ante ciertas excepciones.

Si hay errores de formato se manejan con mensajes de advertencia sin detener ejecución, pero por ejemplo fallos en bases de datos, lo conveniente es detener todo tipo de operaciones dentro de la aplicación

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

Con la modificación que se hizo en la capa de presentation, que es donde está toda la parte de interfaz se validan en actionAdd() y en actionSearch() todos los posibles manejos de los errores que pueden ocurrir, además en la clase HalloweenShopException se podría validar que cuando haya alguna excepción en cuanto al precio que sea negativo o desconocido, entonces genere el JOptionPane adecuado, pero además de eso, mande los errores a un archivo log.





### 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.
2. ¿Qué método es responsable en la capa de presentación?

En la capa de presentation, el método responsable de manejar la búsqueda es actionSearch() en la clase HalloweenShopGUI. Se obtiene el texto de búsqueda del campo textSearch.

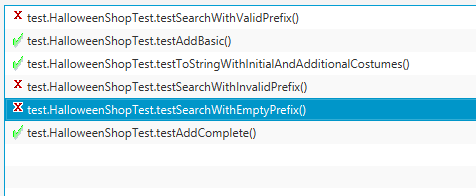
Llama al método search en la clase HalloweenShop (capa de dominio) para realizar la búsqueda.

Muestra los resultados en el área de texto textResults.

1. ¿Qué método es responsable en la capa de dominio?

En la capa de dominio, el método responsable de realizar la búsqueda es search(String prefix) en la clase HalloweenShop. El proceso que usa esté metodo es: llamar a select(String prefix) para obtener una lista de disfraces cuyo nombre comienza con el prefijo dado; luego usa el método data(ArrayList<Costume> selected) para convertir esa lista en una cadena de texto que representa los resultados.

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



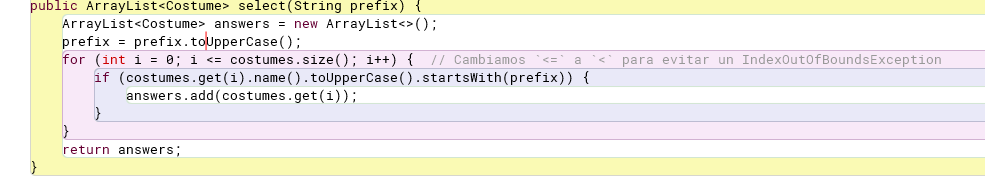
Las pruebas se están haciendo directamente en:



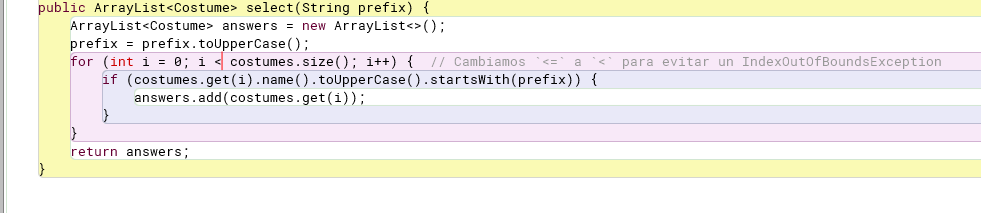
1. ¿Cuál es el error? Soluciónenlo. Capturen los resultados de las pruebas.

El error existe porque se está intentando acceder a algo que está fuera del rango de la lista, en este caso del método select, así que para solucionar es importante revisar el ciclo for, y ver sobre que se está iterando, y así se corrije el error.

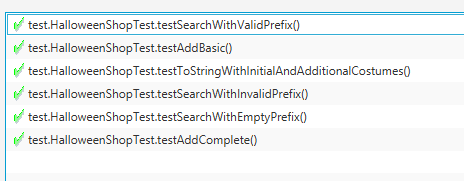
Antes



Después

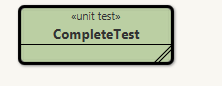


Resultado de las pruebas también validando los disfraces que están predeterminados, por eso también ocurren errores en la contatoria.

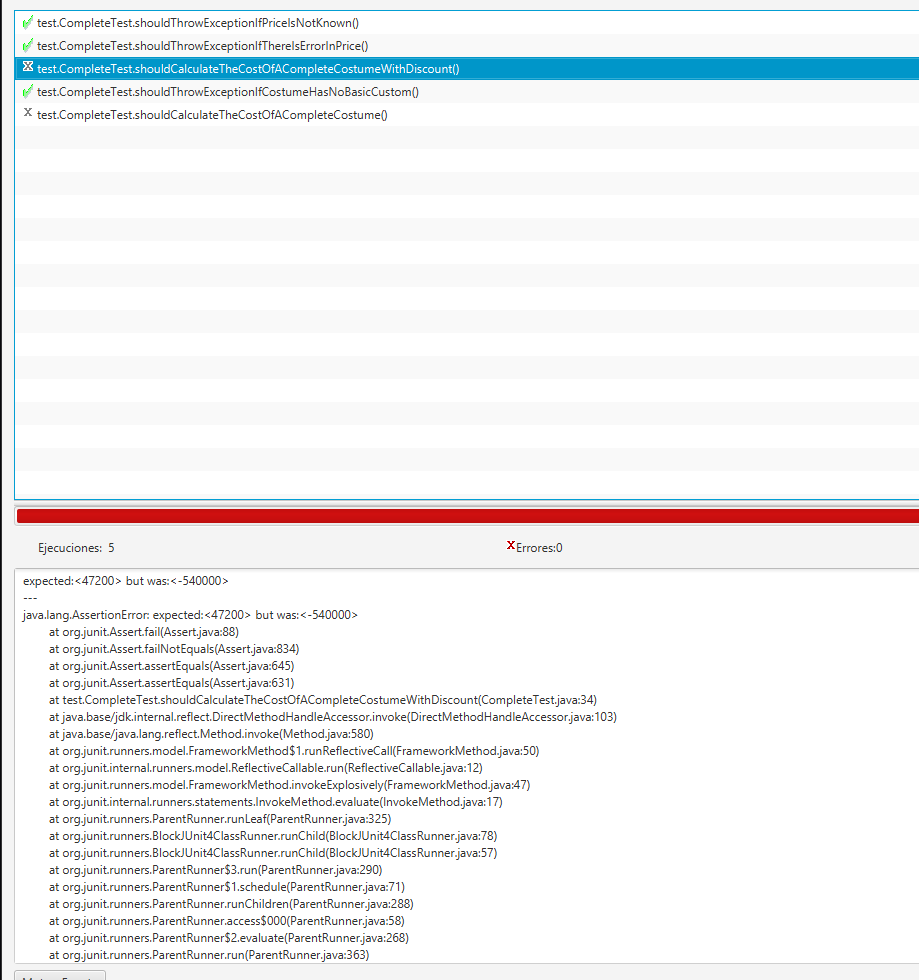


1. Ejecuten la aplicación nuevamente con el caso propuesto.
2. ¿Qué tenemos en pantalla?

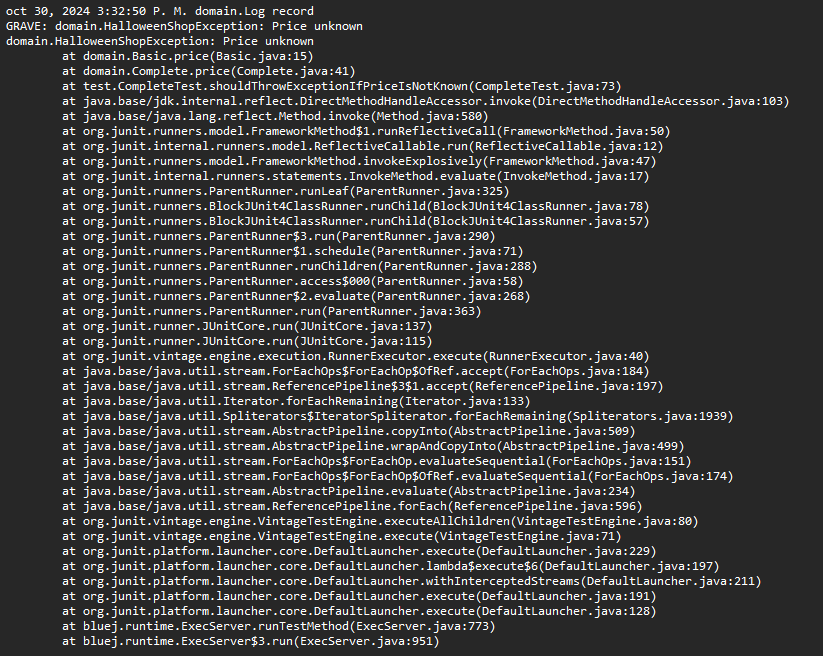
Teniendo en cuenta que el caso propuesto está en Complete Test, sin modificar se obtiene:

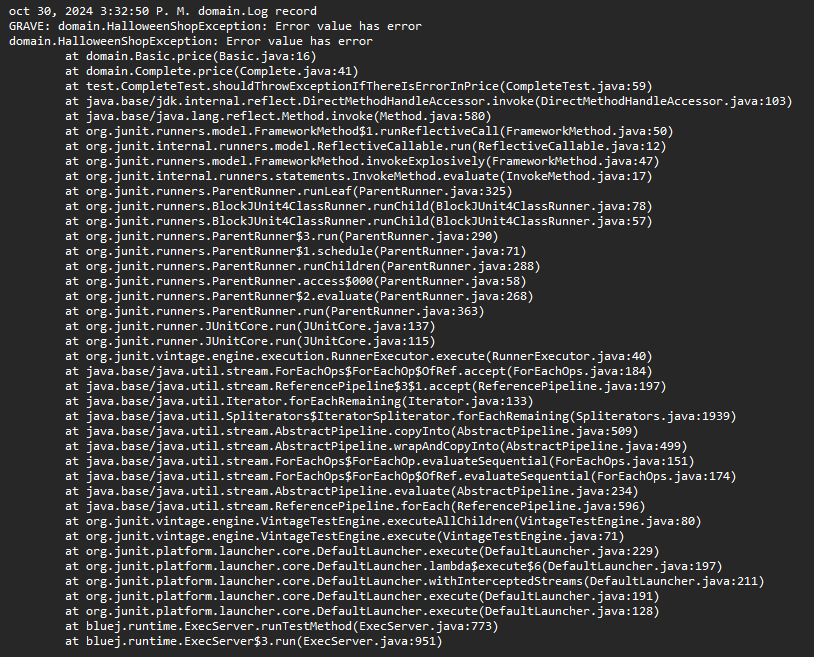


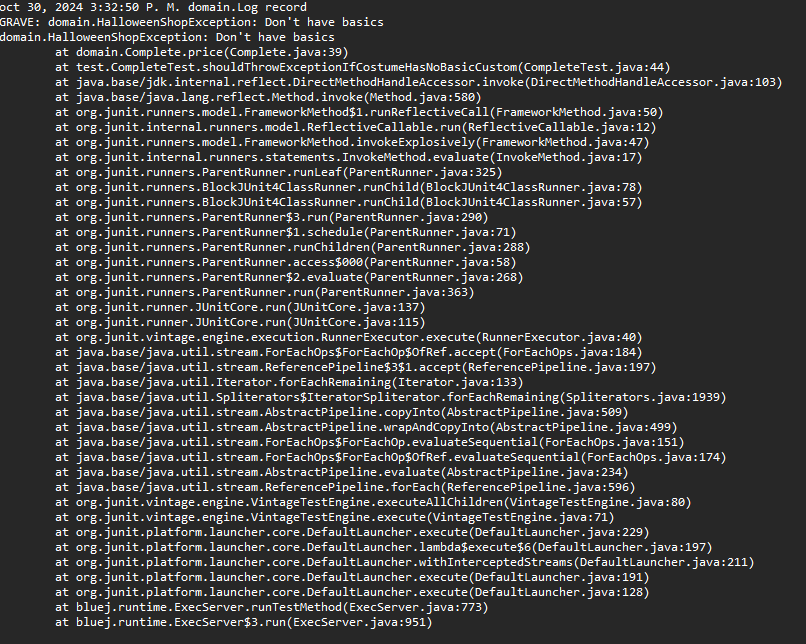
Hay dos fallos:



1. ¿Qué información tiene el archivo de errores?







Viendo el archivo log que se generó al ejecutar las pruebas unitarias CompleteTest en tiempo real nos damos cuenta que:

El archivo log detectó los errores que ocurrieron cuando se ejecutaron las pruebas y estas son debido a la clase de HalloweenShopException especificando en cada sub-test que clase de excepción ocurrió con JUnit y BlueJ.

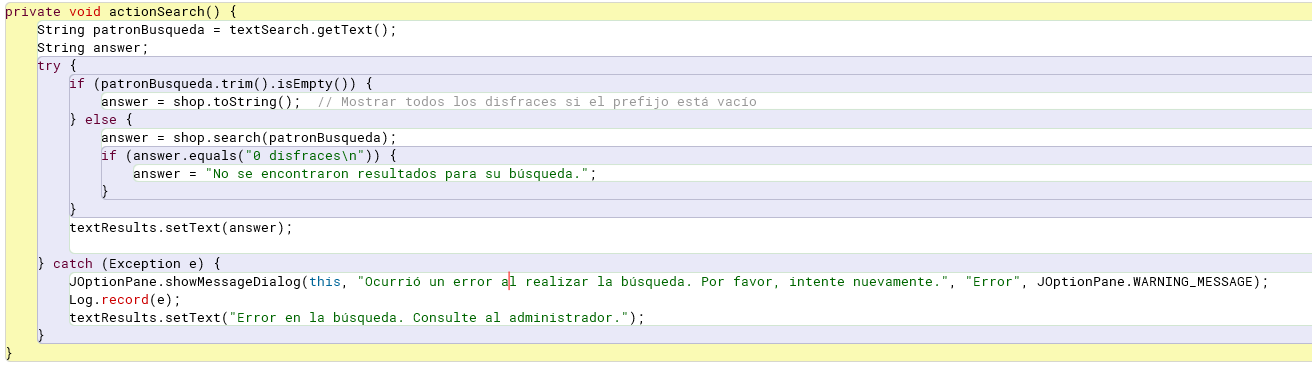
1. Refactorice la funcionalidad para que sea más amable con el usuario.
2. ¿Cuál es la propuesta?

Nuestra propuesta es poner más comentarios al usuario al momento de que este escriba algo incorrecto, detecte el error, lo mande al archivo log, pero pues esto no lo va a saber el usuario. A el usuario solo se le vuelve a preguntar, y se le dice porque está mal lo que le escribió, siempre dando un trato muy adecuado.

Se enriquece el usuario porque:

* Se muestran todos los disfraces cuando el prefijo de búsqueda está vacío.
* Se informa con claridad cuando no se encuentran resultados en lugar de simplemente mostrar "0 disfraces".
* Se capturan y manejan excepciones, asegurando que la aplicación registre los errores y continúe su ejecución.

1. ¿Cómo la implementa?



**RETROSPECTIVA**

1. **¿Cuál fue el tiempo total invertido en el laboratorio por cada uno de ustedes? (Horas/Hombre)**

El tiempo total invertido fue de 20 horas en trabajo en equipo.

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

El laboratorio fue terminado porque el trabajo en equipo fue vital para cumplir este objetivo.

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

La práctica XP más útil fue **Simplicity** porquesiempre tratamos de que el código fue lo más legible y sencillo posible, pero desde que cumpliera con los requerimientos, así aprovechamos mejor el tiempo.

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

El mayor logro es terminar el laboratorio porque se cumplieron con todos los requerimientos, haciendo de este un buen laboratorio.

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

El mayor problema fue entender bien como funcionaban los logs como archivo. Con investigación se logró evidenciar lo útil que pueden llegar a ser en un proyecto gigante, donde se administran ahí todo tipo de excepciones y manejo de errores.

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

Hicimos bien todas las partes de este laboratorio, desde BlueJ hasta terminal porque nos complementábamos en que uno si no podía solucionar algo, él otro ayudaba, y así intercalado, así **Pair Programming.** Nos comprometemos a aplicar esta práctica XP **Simplicity** en los demás laboratorios porque a veces nos vamos por lo difícil y la solución es más fácil de lo que parece.