POOB
Escuela Colombiana de Ingeniería Julio Garavito

Ingeniería de Sistemas

Programación Orientada a Objetos
Excepciones

LABORATORIO 04
27/10/2023

Joan Steven Acevedo Aguilar
joan.acevedo@mail.escuelaing.edu.co

Santiago Sanchez Monroy
santiago.sanchez-m@mail.escuelaing.edu.co

Profesor a cargo:

Maria Irma Diaz Rozo
maria.diaz@mail.escuelaing.edu.co

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.

Disfraces

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

   No funcionaba porque no existía la clase de excepciones y las clases estaban utilizando esta clase. Ya se creó con los mensajes correspondientes de cada excepción hasta el momento.

```
public class CostumeShopException extends Exception
{
    public static final String PRICE_EMPTY = "PRICE_EMPTY";
    public static final String PRICE_ERROR = "PRICE_ERROR";
    public static final String COMPLETE_EMPTY = "COMPLETE_EMPTY";
    public static final String IMPOSSIBLE = "IMPOSSIBLE";

    public CostumeShopException(String message){
        super(message);
    }
}
```

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

```java
/**
 * Nos da el precio del disfraz completo
 *
 * @return int con el precio del disfraz completo
 * @throws PRICE_EMPTY si el precio es vacio
 * @throws PRICE_ERROR si el precio es incorrecto
 */
@Override
public int price() throws CostumeShopException{
    int price = 0;
    if(pieces.size() == 0){
        throw new CostumeShopException(CostumeShopException.COMPLETE_EMPTY);
    }else{
       for(Basic c: pieces){
        price += c.price();
        }
    }
    price = price+this.makeUp - (((price+this.makeUp)*this.discount)/100);
    return price;
}
```

4. Dada la documentación y el diseño, codifiquen y prueben el método price(type).
5. 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?
   En este zip se tiene la carpeta costumeShop-2023-2 que contiene 3 archivos que son de tipo .java y se llaman:
   - CostumeShop
   - CostumeShopGUI
   - Log

   ¿Cómo están organizados?
   Estan todos en una sola carpeta como se mencionó anteriormente
   ¿Cómo deberían estar organizados?
   Pero deberian estar organizados en los correspondientes paquetes que son presentacion y dominio de esta forma
   - Presentacion:
     1. CostumeShopGUI
   - Dominio:
     1. CostumeShop
     2. Log

2. Estudien el diseño del programa: diagramas de paquetes y de clases.
   ¿cuántos paquetes tenemos?
   Por el momento solo tenemos un paquete que es el de dominio
   ¿Cuántas clases tiene el sistema?
   Actualmente tenemos 5 clases en el proyecto y son las siguientes:
   - Costume: Clase abstracta
   - Basic: Clase que hereda de Costume
   - Complete: Clase que hereda de Costume
   - CostumeShopException: Clase de excepciones del proyecto
   - Complete Test: Clase de pruebas

   ¿cómo están organizadas?
   ¿cuál es la clase ejecutiva?
   En este caso la clase ejecutiva sería la clase abstracta Costume
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, 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?
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 un disfraz. Funcionalidad robusto [En lab04.doc, CostumeShop.astay *.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 del disfraz ya existe?
b. ¿Y si en precio no da un número? ¿o no da un número negativo?
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 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?
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 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?
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?