Taller #6
Excepciones y pruebas unitarias

Objetivo general del taller

El objetivo general de este taller es practicar el diseño e implementación de excepciones, utilizándolas para complementar un programa existente.

Objetivos específicos del taller

Durante el desarrollo de este taller se buscará el desarrollo de las siguientes habilidades:

1. Diseñar jerarquías de excepciones a partir de la información sobre un dominio y los posibles problemas que pueden presentarse ahí.
2. Diseñar el contenido de las excepciones (payload) para que sea útil para el diagnóstico o corrección de los problemas.
3. Implementar las excepciones y los mecanismos para su manejo.
4. Diseñar e implementar un conjunto de pruebas unitarias y de integración a partir de su diseño, utilizando un framework de pruebas especializado.

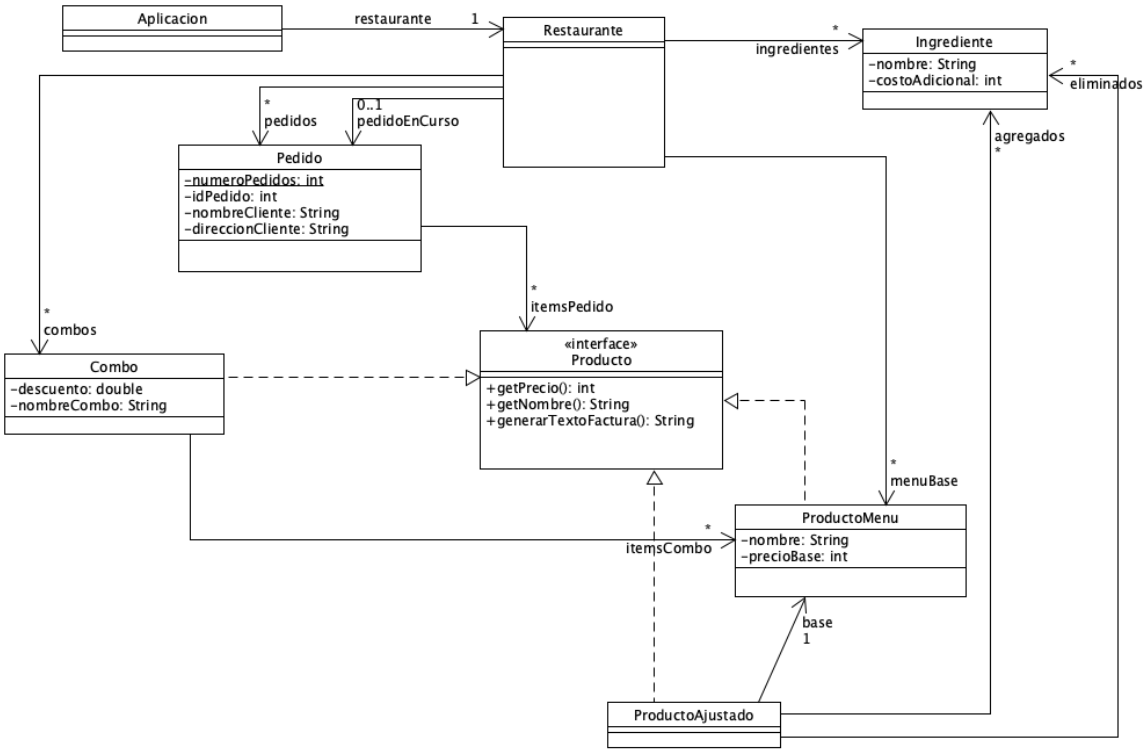
Instrucciones generales

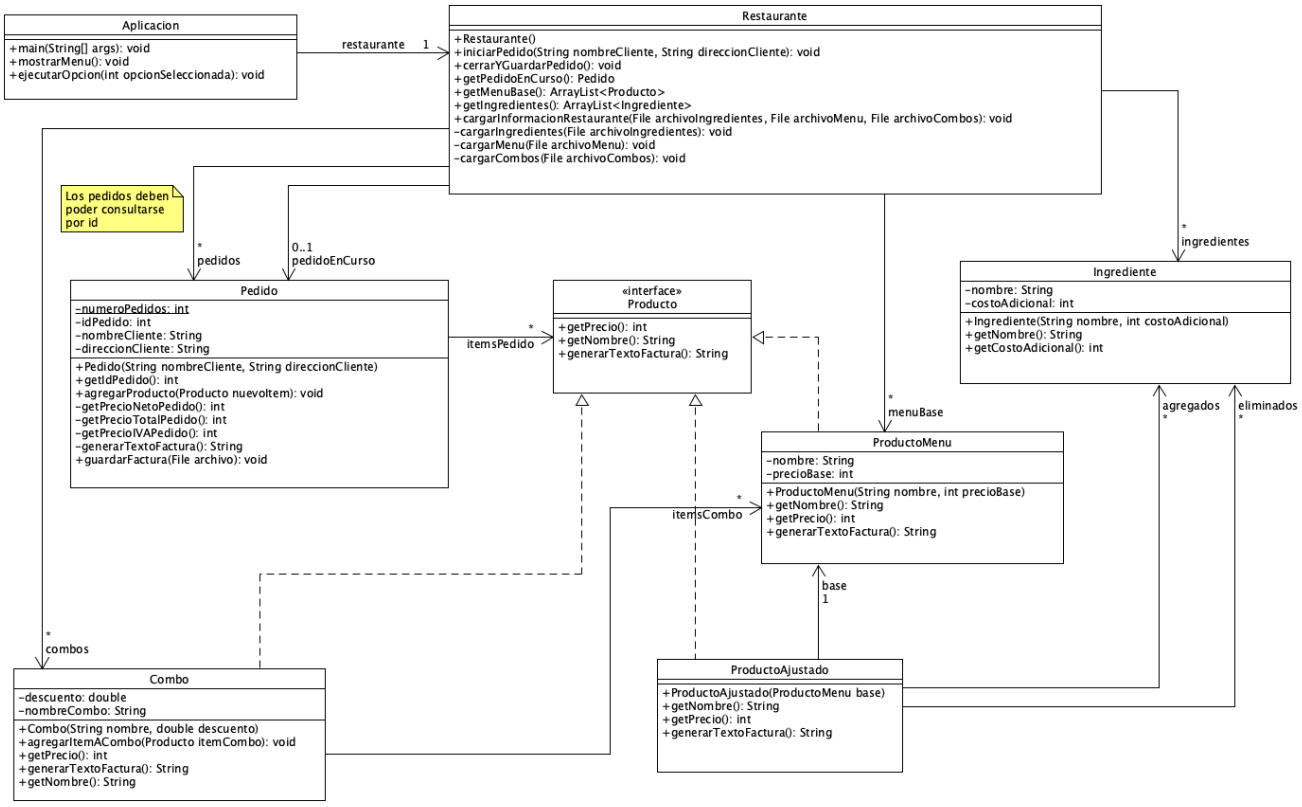
1. Descargue el proyecto que entregó para el Taller 2, renómbrelo como Taller 6 e impórtelo en Eclipse.
2. Realice las modificaciones necesarias al proyecto de acuerdo a los requerimientos que se muestran más adelante. Para manejar los errores relevantes, use el mecanismo de Excepciones de Java siguiendo las restricciones establecidas en el enunciado.

El taller debe desarrollarse de forma individual.

Descripción del caso: Hamburguesas

El caso de estudio para el taller es el mismo del taller 2 que usted ya conoce. El siguiente es un diagrama de clases de alto nivel, seguido de un diagrama de clases detallado.


Aplicacion
restaurante
1
Restaurante
*
ingredientes
Ingrediente
-nombre: String
-costoAdicional: int
*
eliminados
*
pedidos
0..1
pedidoEnCurso
agregados
*
Pedido
-numeroPedidos: int
-idPedido: int
-nombreCliente: String
-direccionCliente: String
*
itemsPedido
*
combos
«interface»
Producto
+getPrecio(): int
+getNombre(): String
+generarTextoFactura(): String
Combo
-descuento: double
-nombreCombo: String
*
menuBase
ProductoMenu
-nombre: String
-precioBase: int
*
itemsCombo
base
1
ProductoAjustado


Aplicacion
+main(String[] args): void
+mostrarMenu(): void
+ejecutarOpcion(int opcionSeleccionada): void
restaurante
1
Restaurante
+Restaurante()
+iniciarPedido(String nombreCliente, String direccionCliente): void
+cerrarYGuardarPedido(): void
+getPedidoEnCurso(): Pedido
+getMenuBase(): ArrayList<Producto>
+getIngredientes(): ArrayList<Ingrediente>
+cargarInformacionRestaurante(File archivoIngredientes, File archivoMenu, File archivoCombos): void
-cargarIngredientes(File archivoIngredientes): void
-cargarMenu(File archivoMenu): void
-cargarCombos(File archivoCombos): void
Los pedidos deben poder consultarse por id
*
pedidos
0..1
pedidoEnCurso
*
ingredientes
Pedido
-numeroPedidos: int
-idPedido: int
-nombreCliente: String
-direccionCliente: String
+Pedido(String nombreCliente, String direccionCliente)
+getIdPedido(): int
+agregarProducto(Producto nuevoItem): void
-getPrecioNetoPedido(): int
-getPrecioTotalPedido(): int
-getPrecioIVAPedido(): int
-generarTextoFactura(): String
+guardarFactura(File archivo): void
*
itemsPedido
«interface»
Producto
+getPrecio(): int
+getNombre(): String
+generarTextoFactura(): String
Ingrediente
-nombre: String
-costoAdicional: int
+Ingrediente(String nombre, int costoAdicional)
+getNombre(): String
+getCostoAdicional(): int
*
menuBase
agregados
*
eliminados
*
ProductoMenu
-nombre: String
-precioBase: int
+ProductoMenu(String nombre, int precioBase)
+getNombre(): String
+getPrecio(): int
+generarTextoFactura(): String
*
itemsCombo
base
1
*
combos
Combo
-descuento: double
-nombreCombo: String
+Combo(String nombre, double descuento)
+agregarItemACombo(Producto itemCombo): void
+getPrecio(): int
+generarTextoFactura(): String
+getNombre(): String
ProductoAjustado
+ProductoAjustado(ProductoMenu base)
+getNombre(): String
+getPrecio(): int
+generarTextoFactura(): String

Parte 1: Excepciones

Construya las clases `IngredienteRepetidoException` y `ProductoRepetidoException`, las cuales deben extender la nueva clase abstracta `HamburguesaException`. Estas dos clases se usarán para reportar si, cuando se cargue la información del restaurante, hay un ingrediente o un producto repetido. Diseñe estas clases para que tengan información suficiente para que se pueda diagnosticar el problema.

Modifique los métodos `cargarIngredientes` y `cargarMenu` para que verifiquen si hay ingredientes o productos repetidos y lancen las excepciones si es necesario.

Modifique cualquier otro método donde le aparezcan errores de compilación. Como el problema que está anunciando la excepción es un problema de los datos, lo único que se debe hacer para manejar el error es anunciarle al usuario que hay un problema en los datos.

Parte 2: Excepción Pedido

El restaurante agregó una nueva restricción: un pedido no puede superar un valor total de 150.000 pesos. Cree una nueva clase para reportar este problema y asegúrese de que el método `agregarProducto` de la clase `Pedido` lance esta excepción si es necesario.

Modifique todo lo que se sea necesario en el resto de la aplicación para poder manejar este error, informándole al usuario que no se puede agregar el producto deseado al pedido.

Parte 3: Pruebas unitarias y de integración

En esta parte del taller, usted tendrá que diseñar e implementar pruebas unitarias y de integración utilizando el framework JUnit. Idealmente se deberían implementar primero pruebas unitarias y luego las de integración, pero el proyecto tiene comparativamente pocas clases muy acopladas entre ellas, lo cual hace difícil probarlas por separado.

Las pruebas que ustedes van a construir deben procurar que todos los métodos interesantes de las clases sean verificados: una parte importante del esfuerzo que tendrán que hacer corresponde al diseño de los escenarios de prueba.

Actividades

Para cada una de las siguientes actividades, diseñe los escenarios de prueba, las acciones y la forma de verificar el resultado, ANTES de empezar la implementación. Es usual que haya varios escenarios para probar una misma clase.

1. Construya una clase llamada `ProductoMenuTest` donde implemente pruebas para la clase `ProductoMenú`. Asegúrese de que sus pruebas cubran el 100% de la clase, sin contar los métodos `getNombre` y `getPrecio`.
2. Construya una clase llamada `ProductoAjustadoTest` donde implemente pruebas para la clase `ProductoAjustado`. Asegúrese de que sus pruebas cubran el 100% de la clase, sin contar el método `getNombre`. Fíjese que en este caso es muy importante probar el método `getPrecio`.
3. Construya una clase llamada `ComboTest` donde implemente pruebas para la clase Combo. Asegúrese de que sus pruebas cubran el 100% de la clase, sin contar los métodos `getNombre` y `getPrecio`.
4. Construya una clase llamada `PedidoTest` donde implemente pruebas para la clase `Pedido`. Asegúrese de que sus pruebas cubran el 100% de la clase.

En la clase `PedidoTest` es particularmente importante que usted verifique los métodos `generarTextoFactura` y `guardarFactura`. Tenga en cuenta esto dentro del diseño de sus escenarios de prueba.

Tenga también en cuenta que debe verificar que se lance correctamente la excepción cuando el valor total de un pedido supere los 150.000 pesos.

Entrega

1. Cree un repositorio privado para la organización donde quedará el taller 6 y dejen ahí todos los archivos relacionados con su entrega.

2. Entregue a través de Bloque Neón el URL para el repositorio, en la actividad designada como "**Taller 6**". Sólo se debe hacer una entrega por grupo.