Java Intermedio 2019 - Módulo II

November 27, 2019

1 Java Intermedio - Módulo II

1.0.1 Programa

Enums. Manejo de errores. Introducción al uso de excepciones. Bloques try, catch y finally. La clase Exception. Tipos de excepciones: checked vs. unchecked. Excepciones construidas por el usuario.

1.1 Enums

Un enumerado (o Enum) es una clase "especial" (tanto en Java como en otros lenguajes) que limitan la creación de objetos a los especificados explícitamente en la implementación de la clase. La única limitación que tienen los enumerados respecto a una clase normal es que si tiene constructor, este debe de ser privado para que no se puedan crear nuevos objetos.

Un tipo enumerado restringe los posibles valores que puede tomar una variable.

Dentro de las llaves se declaran las variables (objetos) de que consta el tipo enumerado. Por convención, sus nombres se escriben en letras mayúsculas para recordarnos que son valores fijos (que en cierto modo podemos ver como constantes). Una vez declarado el tipo enumerado, todavía no existen variables hasta que no las creemos explícitamente, de la misma manera que ocurre con cualquier tipo Java. Para crear una variable de tipo enumerado lo haremos con una declaración simple que recuerda a la creación de una variable tipo primitivo: nombreTipoEnumerado miNombreElegido;.

Esta forma de creación de variables de tipo enumerado se justifica porque los tipos enumerados en principio no tienen constructores (más adelante veremos que sí los pueden tener). Los valores de un tipo enumerado son objetos propiamente dichos, ¿de qué tipo? Del tipo enumerado cuyo nombre se haya indicado. No se pueden crear más objetos variantes del tipo enumerado que los especificados en su declaración. Tener en cuenta, porque es una confusión habitual, que los tipos enumerados no son enteros, ni cadenas (aunque a veces podamos hacer que se comporten de forma similar a como lo haría un entero o una cadena). Cada elemento de un enumerado es un objeto único disponible para su uso.

Ejemplo

1.2 Excepciones

En Java existen dos tipos de excepciones: - las de tiempo de compilación (se deben controlar antes de ejecutar el programa) y - las de tiempo de ejecución (ocurren una vez que la aplicación se está ejecutando y no es obligatorio controlarlas).

Las excepciones en tiempo de ejecución pueden ocurrir cuando se produce un error en alguna de las instrucciones de nuestro programa, como por ejemplo cuando se hace una división entre cero, cuando un objeto es 'null' y no puede serlo, cuando no se abre correctamente un fichero, etc. Cuando se produce una excepción se muestra en la pantalla un mensaje de error y finaliza la ejecución del programa.

Cuando en Java se produce una excepción se crea un objeto de una determina clase (dependiendo del tipo de error que se haya producido) que mantendrá la información sobre el error producido y nos proporcionará los métodos necesarios para obtener dicha información. Estas clases tienen como clase padre a la clase Throwable, por lo tanto se mantiene una jerarquía en las excepciones.

Ejemplo

1.3 Bloque Try - Catch - Finally

Java nos permite manejar las excepciones para que nuestro programa continue su ejecución y no se detenga cuando surja alguna. Para ello tenemos la estructura "try – catch – finally". Podemos obviar el finally o el catch pero nunca ambos. En el catch podemos agregar un bloque para cada posible excepción esperada y asi ejecutar un algoritmo diferente en cada caso. Veamos un ejemplo:

Ejemplo

1.4 Excepciones checked vs unchecked

Todas las excepciones en Java heredan de Throwable subdividiéndose en Error y Exception, las primeras son condiciones de error del sistema y las segundas condiciones de error del programa. A su vez las Exception pueden ser checked si heredan de esta y son aquellas que el compilador fuerza a que sean capturadas no pudiendo ignorarse, han de capturarse en una construcción try catch o declarar que el método puede lanzar la excepción no capturada. Las excepciones uncheked heredan de RuntimeException que heredan a su vez de Exception pero tienen la particularidad de que no es necesario capturarlas ni declararlas como que se pueden lanzar debido a que se consideran condiciones de error en la programación como un acceso a un array fuera de rango que produce un ArrayIndexOutOfBounds, el conocido NullPointerException cuando se utiliza una referencia nula, otro es ArithmeticException que se produce al dividir por 0.

Ejemplo

1.4.1 Ejercicio

Crear un paquete para contener una clase que modele un objeto de la realidad (un avion por ejemplo) y sea subclase de una clase padre abstracta (Vehiculo) también presente en el mismo paquete. Debe heredar algunos de sus métodos.

Crear una clase en otro paquete con el nombre Controlador[ObjetoModelado] (ControladorAvion por ejemplo) que contenga al método main (public static void main(Strings[] args)…).

Crear un paquete llamado custom.excepciones dentro del cual crearemos 3 tipos de excepciones customizadas que hereden de algunas de las clases Exceptions contenidas en java.lang.exceptions (https://docs.oracle.com/javase/10/docs/api/java/lang/Exception.html). Dentro del método main vamos a instanciar un objeto de la clase que modelamos y realizaremos una comprobación al momento de pasar los parámetros al constructor con el objetivo de que si alguno de los datos no es correcto nos lanze la excepción customizada.

[ ]: