Excepciones

Se pide resolver cada uno de los ejercicios utilizando diagramas de clases UML y el lenguaje de programación Java.

[K.01] - Interface con excepciones.

Crear un proyecto de biblioteca de clases con una interface llamada Divisible, la cuál contendrá un método llamado dividir(double a, double b) que retorna un double.

Se pide implementar la interface en una clase llamada Calculadora, y maneja la excepción de ArithmeticException en el método dividir.

Una vez diseñado el diagrama de clases UML, crear, implementar y probar las entidades en un proyecto de consola.

[K.02] - Burbujeo de excepciones

Crear una aplicación de consola y agregar el código necesario para que se puedan ir almacenando los distintos mensajes de las excepciones generadas. Para ello:

1. Crear una clase que derive de Exception (con el nombre MiExcepcion) que concatene el mensaje que tiene como parámetro de entrada en el constructor con el ya existente.
2. En el método main:
    a. Realizar un burbujeo partiendo de un método, capturando y re lanzando esta misma excepción como mínimo 3 (tres) veces, creando los métodos que sean necesarios y colocando un nuevo mensaje por cada nuevo **throw** que se realice.
    b. Realizar el burbujeo de la excepción MiExcepcion, comenzando en un método de instancia, pasando por un método de estático y capturando por última vez en el “main”.

En ambos casos, mostrar el mensaje obtenido por la última excepción.

[K.03] - Contabilidad

Crear un proyecto de biblioteca de clases la clase Producto que represente un producto en un inventario. Implementa excepciones personalizadas para manejar situaciones anómalas relacionadas con lacantidad y el precio del producto.
Se pide:

1. Crear dos excepciones personalizadas:
- ❖ CantidadInvalidaException: Se lanzará si se intenta establecer una cantidad negativa.
- ❖ PrecioInvalidoException: Se lanzará si se intenta establecer un precio negativo.

2. Crear la clase Producto con las siguientes propiedades:
   - nombre (String)
   - cantidad (int)
   - precio (double)

Debe incluir métodos para establecer y obtener las propiedades, lanzando las excepciones correspondientes. Implementar el método llamado **reabastecer(int)** que aumente la cantidad de un producto, lanzando CantidadInvalidaException si la cantidad a añadir es negativa.

Una vez diseñado el diagrama de clases UML, crear, implementar y probar las entidades en un proyecto de consola.

[K.04] - Los Tesla, un poroto

Crear un proyecto de biblioteca de clases con una interface llamada VehiculoElectrico y varias clases que implementen esta interface y valida las entradas de los métodos utilizando la excepción IllegalArgumentException.

La interface VehiculoElectrico tendrá los siguientes métodos:
- cargarBateria(int cantidad): para cargar la batería del vehículo.
- getNivelBateria( ): para obtener el nivel actual de la batería.

La clase abstracta llamada Vehiculo poseerá los siguientes miembros:
Atributos:
- marca (String)
- modelo(String)

Métodos:
- Un método para **acelerar()**: que reciba la velocidad que debe aumentar en km/h.
- Un método **frenar()**: Sobrecargar este método para que se pueda frenar del todo el vehículo o se pueda indicar la velocidad de frenado.
- Un método **mostrarInformacion()**: que retorne un String con la información del vehículo.

La clase concreta llamada Tesla que extienda Vehiculo e implemente VehiculoElectrico, debe incluir:
- Un campo nivelBateria:int para representar el nivel de carga de la batería en porcentaje (%).
- Implementaciones de los métodos de la interface VehiculoElectrico.
- Un método adicional **autopilotar()**: que retorne un String que indique que el vehículo está en modo de conducción autónoma.

La otra clase concreta llamada ScooterElectrico que extienda Vehiculo e implemente VehiculoElectrico. Esta clase debe tener un método adicional **pasear()**: que retorne un String indicando que el scooter está en movimiento.

Una vez diseñado el diagrama de clases UML, crear, implementar y probar las entidades en un proyecto de consola.

1. Crear una Instancia de Tesla:
   - Crear un objeto de tipo Tesla utilizando un modelo "Model S".
   - Llamar al método mostrarInformacion() para verificar que la información se muestre correctamente, incluyendo la marca, modelo y nivel de batería.

2. Cargar Batería de Tesla:
   - Invocar al método cargarBateria() con el valor 20.
   - Luego, invocar a getNivelBateria() para verificar que el nivel de batería se ha incrementado correctamente.
   - Pruebar cargar la batería con un valor negativo (por ejemplo, -10) y verificar que se lanza una excepción o se maneja adecuadamente, dado que no se debe permitir cargar una cantidad negativa.

3. Verificar el método autopilotar():
   - Invocar al método autopilotar() y verificar que se muestre el mensaje correspondiente.

4. Crear una Instancia de ScooterElectrico:
   - Crear un objeto de tipo ScooterElectrico utilizando el modelo "Xiaomi M365".
   - Invocar al método mostrarInformacion() para verificar que la información se muestre correctamente.

5. Cargar Batería de ScooterElectrico:
   - Invocar al método cargarBateria() con el valor 15.
   - Verificar el nuevo nivel de batería usando getNivelBateria().
   - Intentar cargar la batería con un valor negativo (por ejemplo, -5) y verificar que se lanza una excepción.

[K.05] - Hostel manager

Crear un sistema que gestione reservas en un hotel. La estructura debe incluir una interface llamada Reservable, que declare métodos para realizar y cancelar reservas. Luego, crear una clase llamada Hotel que implemente la interface. Además, se debe incluir la clase Habitacion, que representará una habitación en el hotel. El hotel mantendrá una lista de reservas y gestionará la lógica relacionada.

La interface Reservable, declara los siguientes métodos:
- boolean realizarReserva(String nombreCliente, int numeroDeNoches): para realizar una reserva.
- boolean cancelarReserva(String nombreCliente): para cancelar una reserva.

La clase Hotel:

- Implementa la interfaz Reservable.
- Mantiene una lista de reservas utilizando un HashMap<String, Integer> (donde la clave es el nombre del cliente y el valor es el número de noches reservadas).
- Contiene una lista de habitaciones (puede ser un ArrayList<Habitacion>).
- Lanza una excepción personalizada llamada ReservaInvalidaException si se intenta realizar una reserva con un número de noches menor o igual a cero.
- Lanza otra excepción personalizada llamada ReservaNoExistenteException si se intenta cancelar una reserva que no existe.

Implementa métodos de sobrecarga para realizar reservas:

- boolean realizarReserva(String nombreCliente, int numeroDeNoches, String tipoHabitacion): para realizar una reserva con un tipo de habitación.
- boolean realizarReserva(String nombreCliente, int numeroDeNoches, String tipoHabitacion, boolean desayunoIncluido): para realizar una reserva con un tipo de habitación y opción de desayuno.

La clase Habitacion posee:
Atributos:

- tipoHabitacion (String)
- desayunoIncluido (boolean)

Métodos:

- getDetalles(): devuelve, en formato de cadena de caracteres, todos los detalles de la habitación (tipo y si incluye desayuno o no).

Una vez diseñado el diagrama de clases UML, crear, implementar y probar las entidades en un proyecto de consola.