



Trabajo Práctico VI: Excepciones

Excepciones.

Tipos de excepciones.

Creando tipos de excepción.

Qué información debe almacenar la excepción.

Manejo de excepciones: throw, throws, try, catch y finally.

Cuándo y cómo usar excepciones.

Clonación superficial y profunda.

Clonación condicional.

Objetivos de la práctica

Comprender la utilidad de utilizar excepciones.

Crear y lanzar excepciones de usuario.

Utilizar correctamente el orden de las cláusulas catch para tratar excepciones de diferente tipo.

Tratar o delegar excepciones.

Utilizar clonación superficial de objetos.

Sobreescribir el método clone() para realizar una clonación profunda de objetos.

Utilizar excepciones para lograr una clonación condicional de objetos.

Ejercicio:1

Para un sistema de registro de usuarios se tiene la clase Usuario que tiene los atributos nombre y contraseña, ambos de tipo String.

Para que el nombre sea válido deberá ser distinto de null y no ser un String vacío. La contraseña deberá ser diferente null, tener más de seis caracteres, y el primer carácter debe ser una letra.

Escribe la clase Usuario con sus getters y setters correspondientes, teniendo en cuenta que los setters, comprobarán que las cadenas recibidas sean válidas. En caso de que el nombre no sea válido se lanzara una excepción de tipo NombreInvalidoException y se pasara como parámetro al constructor de la excepción el motivo del error. De forma analoga se lanzara la excepción de tipo ContraseñaInvalidaException.

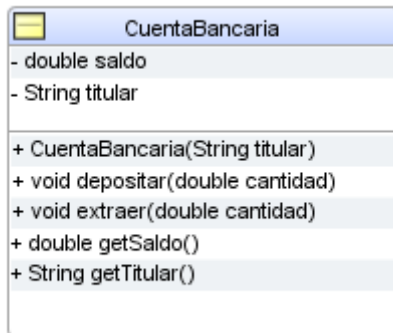
La clase Usuario deberá tener un constructor con dos parámetros de tipo String correspondientes al nombre y contraseña, dicho constructor propagará las excepciones.

Escribe una clase Main para comprobar las funcionalidades de la clase usuario. En dicho método main se tratarán las excepciones correspondientes mostrando los mensajes de error por pantalla.



Ejercicio:2

Escriba la clase CuentaBancaria:

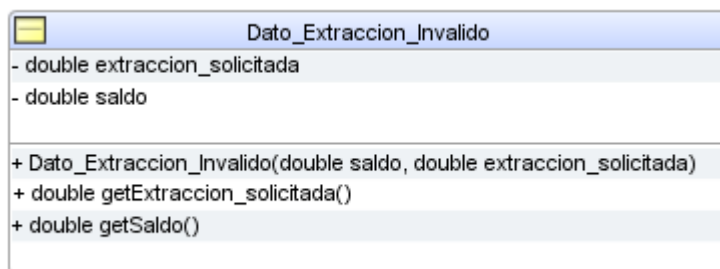


El Método Depositar(double cantidad) ,incrementa el saldo, la cantidad pasada como parámetro, pero lanza una excepción de tipo DepositoInvalidoException, si la cantidad es menor o igual a cero.

La excepción de tipo DepositoInvalidoException, tiene un atributo privado de tipo double llamado cantidadInvalida que se pasa como parámetro en el constructor, y tiene su correspondiente getter. Al momento de lanzar la excepción, se pasa la cantidad que se intenta depositar.

El método Extraer(double cantidad) resta del saldo la cantidad pasada como parámetro si la cantidad es menor o igual a cero. Si la cantidad es mayor que el saldo, el método lanzará una excepción de tipo ExtraccionInvalidaException.

La excepción de tipo ExtraccionInvalidaException, tiene un atributo privado de tipo DatoInvalido (ver ilustracion siguiente) que se pasa como parámetro en el constructor de la excepción, y tiene su correspondiente getter. Al momento de lanzar la excepción, se pasan los datos del depósito inválido.



Inciso a)

Implementa en Java las clase descripta.

Inciso b)

Escribe un programa visual con un JFrame sencillo, donde se muestren los datos de una cuenta bancaria y se nos permita realizar extracciones y depósitos (tanto válidos como inválidos).

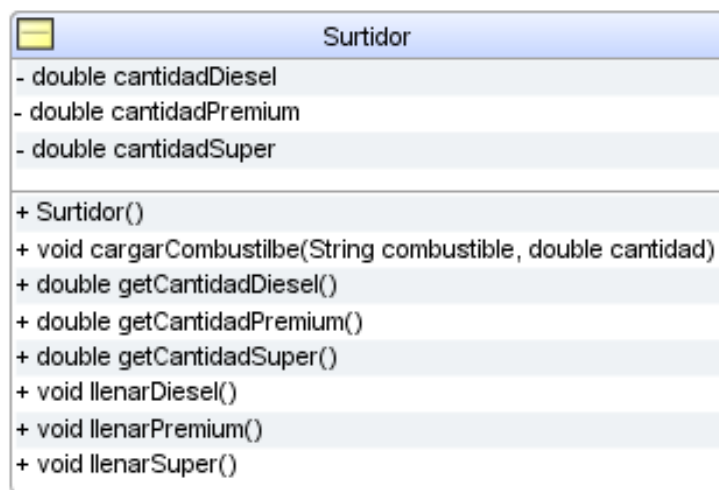
En los casos de extracciones y depósitos inválidos, recupera los datos de la excepción para informar el error en un cuadro de diálogo.



Ejercicio:3

Una estación de servicio cuenta con surtidores de combustible capaces de proveer Gasoil, Nafta Super y Nafta Premium 2000. Todos los surtidores tienen capacidad para almacenar un máximo de 20000 litros de cada combustible. En cada surtidor se mantiene registro de la cantidad de litros disponibles en depósito de cada tipo de combustible, esta cantidad se inicializa en el momento de crearse un surtidor con la cantidad máxima. En cada surtidor es posible cargar o reponer combustible.

Cuando se repone un combustible en el surtidor, se llena el depósito completo de ese combustible. Cada surtidor puede modelarse con el siguiente diagrama:

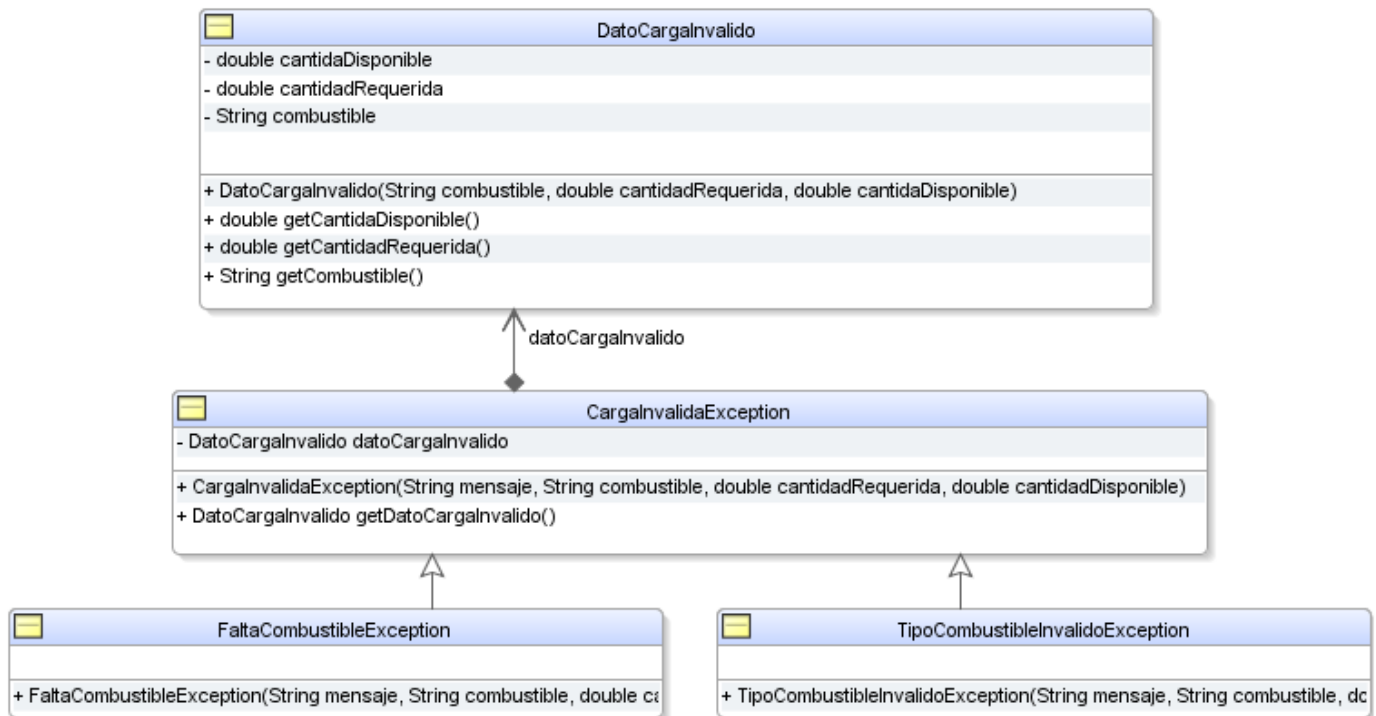


El método `cargarCombustible`, pasa como parámetro el tipo de combustible requerido y la cantidad deseada. Este método puede lanzar una excepción en caso de que suceda alguna de las siguientes situaciones:

- `CargaInvalidaException` en caso de que se solicite una cantidad negativa
- `TipoCombustibleInvalidoException` en caso de que se solicite un combustible desconocido
- `FaltaCombustibleException` en caso de que se solicite una cantidad mayor a la disponible en el surtidor.

En caso de que suceda la última situación, el sistema soluciona el inconveniente cargando lo que se puede. En todos los casos se informará por pantalla la situación ocurrida.

La excepción de tipo `CargaInvalidaException` tiene un atributo privado de tipo `DatoCargaInvalida`, y tendrá su correspondiente getter. Al momento de lanzar la excepción, se pasarán los datos de la carga inválida.

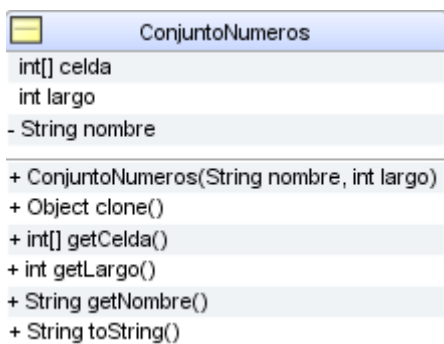


(NOTA: Prestar atención a la línea de herencia entre las diferentes Excepciones)

Escribe un programa visual con un JFrame sencillo que permita verificar los servicios provistos por la clase Surtidor, informando en un área de texto los errores que surjan de cargas inválidas (presta especial atención al orden en que se capturan las excepciones).

Ejercicio:4

Inciso a)

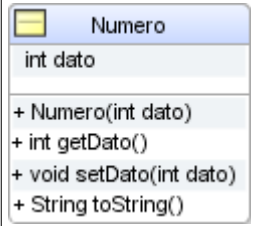


Escribe la clase **ConjuntoNumeros**, que implementa la interfaz **Cloneable**. Considera que deberá hacer una clonación profunda de sus atributos.

Escribe también una clase **Prueba**, que en su método **main**, clonará un **ConjuntoNumeros**, y cambiará un valor de las celdas. Muestra ambos conjuntos por pantalla para comprobar el funcionamiento del método **clone**.



Inciso b)

 <pre>classDiagram class Numero { +int dato +Numero(int dato) +int getDato() +void setDato(int dato) +String toString() }</pre>	<p>Modifica la clase ConjuntoNumeros, de forma que su atributo celda, en lugar de ser un array de int, será un array de objetos de tipo Numero. Analiza las consecuencias que tendrá esto en el método clone.</p>
--	---

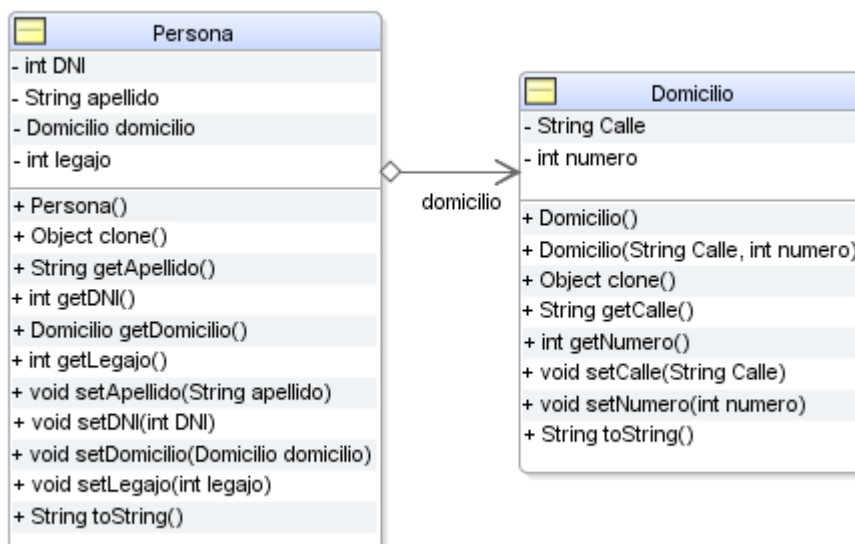
Ejercicio:5

Inciso a)

Escribe las clases Persona y Domicilio.

Persona implementara las interfaces Cloneable y Comparable. El criterio de orden de una persona sera por su apellido, si el apellido es el mismo, entonces se decidirá el orden por DNI.

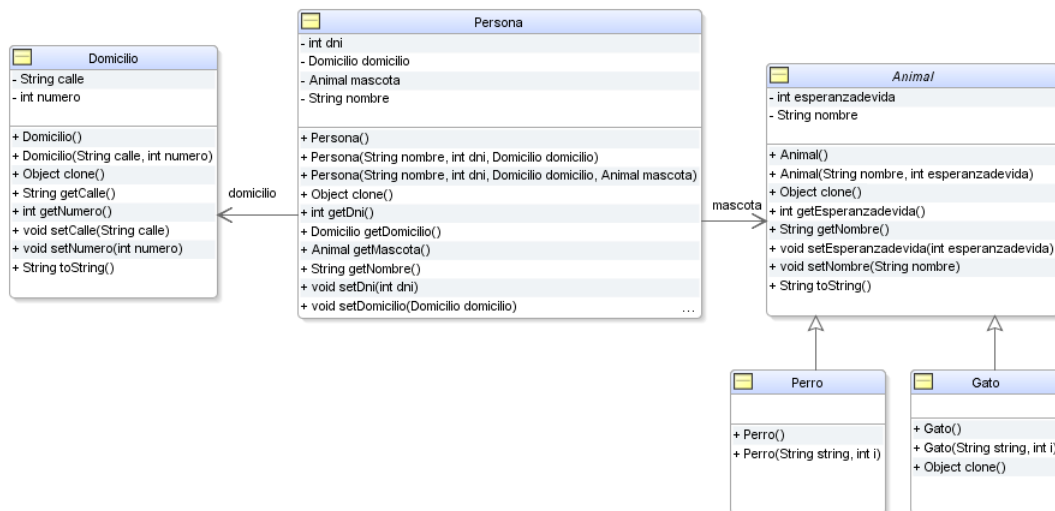
La clase Persona debe permitir una clonación profunda.





Inciso b)

Analiza el siguiente diagrama.



La clase Perro y la clase Gato tienen un atributo nombre de tipo String y un atributo edad de tipo int, pero tienen la diferencia de que los perros son cloneables y los gatos no.

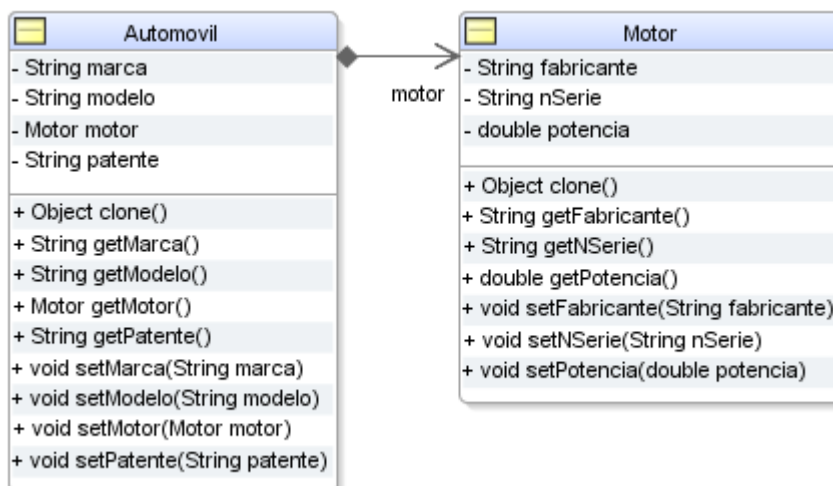
Se desea generalizar y crear una clase Animal de la cual se extienden Perro y Gato. Analiza como solucionar la clonación.

Una Persona tiene una mascota que es un Animal y un domicilio. La clase Persona implementa la interfaz Cloneable, y realiza una clonación profunda.

Escribe una clase Prueba con un método main y comprueba el comportamiento al intentar clonar una persona que tenga un perro y otra persona que tenga un gato.

Ejercicio:6

Se tienen las clases **Automovil** y **Motor**, ambas cloneables.





¿Qué problemas podría ocasionar la siguiente implementación del método clone en la clase **Automovil**?

@Override

```
public Object clone() throws CloneNotSupportedException
{
    Automovil aux = (Automovil) super.clone();
    Motor motorClonado=new Motor();
    motorClonado.setFabricante(this.motor.getFabricante());
    motorClonado.setNSerie(this.motor.getNSerie());
    motorClonado.setPotencia(this.motor.getPotencia());
    aux.setMotor(motorClonado);
    return aux;
}
```

Ejercicio:7 referentes al juego de estrategia de las guías anteriores

Modifica el ejercicio del juego de estrategia de la guía anterior agregando los siguientes cambios:

Inciso a)

En la interfaz Movable, elimina el método setXY. Los personajes, solo podrán cambiar de posición a través del método **void incrementaPos(double valorX, double valorY)**.

La clase Personaje agrega el atributo privado **double** distanciaMaximaDeDeplazamiento que deberá tomar los siguientes valores iniciales:

Guerrero = 5; Arquero = 8; Caballero = 10;

Si un Personaje pretende incrementar su posición de forma que este desplazamiento supere la distanciaMaximaDeDeplazamiento se deberá lanzar una excepción de tipo IncrementoImposibleException

La clase IncrementoImposibleException debe tener los atributos privados

double maxDistanciaSoportada

doube distanciaPretendida

Ambos atributos se pasan como parámetro al constructor y deben tener sus correspondientes setters.

El método **ataca(Personaje)** de la clase Personaje ahora debe ser un método de tipo void y en caso de no poder realizar el ataque porque el adversario esta fuera de alcance, deberá lanzar la excepción AtaqueImposibleException que tendrá los atributos atacante y atacado (ámbos de tipo Personaje) que se pasan por parámetro en el constructor de la excepción y que cuentan con sus correspondientes getters.

Inciso b)

La clase Mapa, que contiene una colección de Personajes, y con la cual interactúa la ventana, tendrá los métodos:

void mueve (Personaje, double valorX, double valorY)

void ataca(Personaje atacante, Personaje atacado)



El método ataca deberá propagar la excepción que se lance. En caso de recibir la excepción, la ventana mostrara un cuadro de dialogo emergente con los datos de dicha excepción.

El método mueve, no propaga la excepción, en caso de no poder mover al personaje de acuerdo a los valores requeridos, se deberá mover el personaje en la dirección requerida la distancia máxima posible.

Inciso c)

Diseña la clase Mago, que tendrá comportamiento similar al Caballero, con la diferencia de que el método **incrementaPos(double valorX, double valorY)** nunca lanza excepciones, ya que el mago puede teletransportarse a cualquier posición.

Inciso d)

Realiza los cambios necesarios para que la clase Personaje sea Comparable. El criterio de comparación es alfabéticamente a través de su nombre, si dos personajes tienen el mismo nombre se decidirá el orden a través de su vitalidad.

Inciso e)

Realiza los cambios necesarios para que las clases Mapa y Personajes sean siempre Cloneables (clonación profunda).