La inteligencia consiste no sólo en el conocimiento, sino también en la destreza de aplicar los conocimientos en la práctica – Aristóteles

Práctica n° 6

Nota: es conveniente que haga (también) los ejercicios en papel, usando la notación dada en teoría para poder realizar el parcial escrito con la notación.

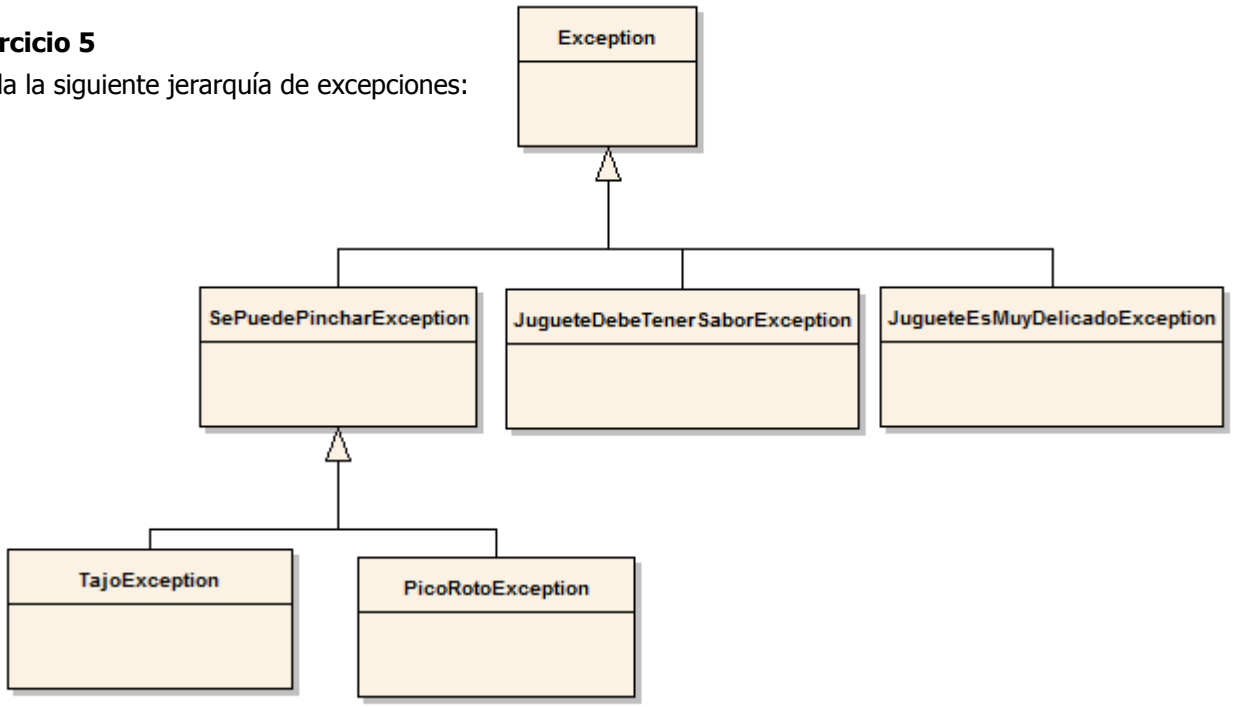
Ejercicio 1 (Opcional)
Cree un archivo de texto que tenga una secuencia de caracteres. Lea el archivo desde Smalltalk para generar luego otro archivo donde se almacenarán los caracteres que no son dígitos, ordenados alfabéticamente. No olvide liberar los recursos que tome.

Ejercicio 2
Dado un String cualquiera, escriba código Smalltalk que verifique que el String es palíndrome (capicúa) utilizando streams.

Ejercicio 3
Describa qué ventajas le encuentra al uso de *doble encapsulamiento*. Describa también las ventajas del *double dispatching*. ¿Existe alguna relación entre el double dispatching y el doble encapsulamiento?

Ejercicio 4
Agregue el mensaje `peek` a las clases `Pila` y `Cola` (ver ejercicio 12 de la práctica 5), que retorna el objeto próximo a ser removido de la pila o la cola cuando se envía el mensaje `pop`, pero sin removerlo. Cuando la pila o la cola está vacía no hay elemento que se pueda *ver* o *remover*, por lo que se pide además modelar estas situaciones creando y levantando una excepción nueva.

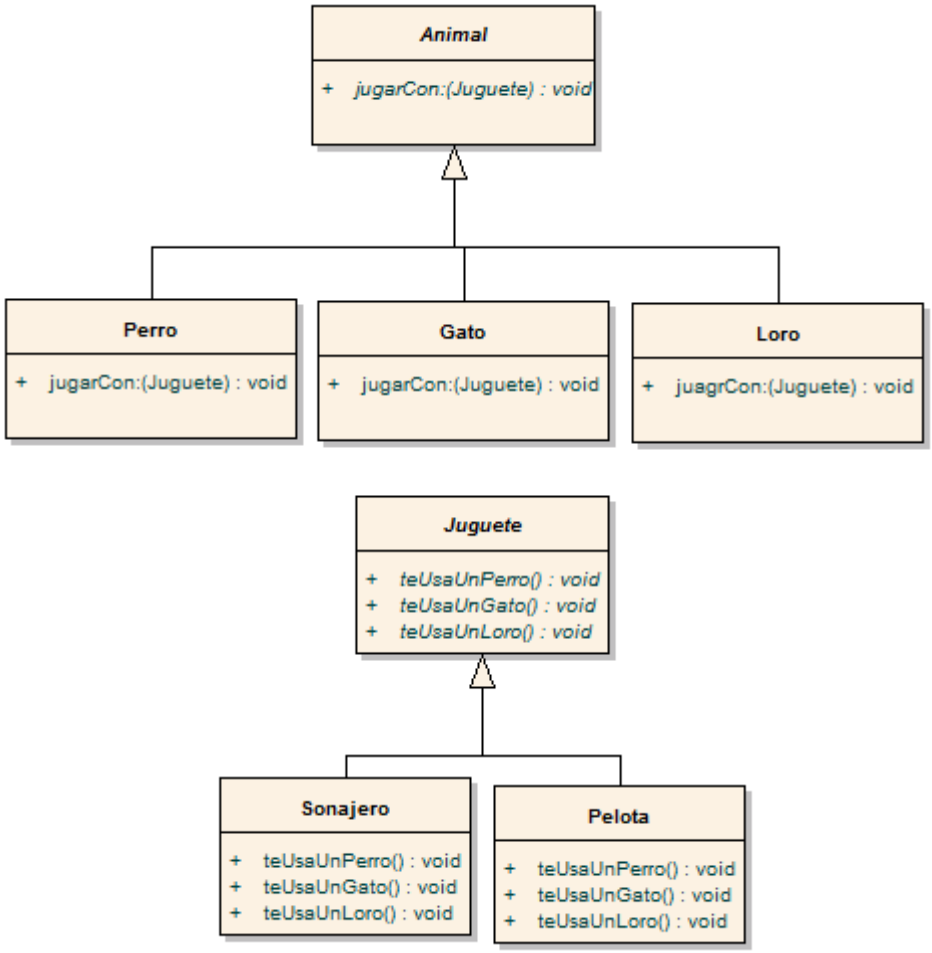
Ejercicio 5
Dada la siguiente jerarquía de excepciones:



Y los siguientes métodos de instancia:

```
Animal>> jugarCon: unJuguete
        self subclassResponsibility

Perro>> jugarCon: unJuguete
        unJuguete teUsaUnPerro

Gato >> jugarCon: unJuguete
        unJuguete teUsaUnGato

Loro >> jugarCon: unJuguete
        unJuguete teUsaUnLoro

Juguete >> teUsaUnPerro
        self subclassResponsibility

Juguete >> teUsaUnGato
        self subclassResponsibility

Juguete >> teUsaUnLoro
        self subclassResponsibility

Sonajero >> teUsaUnPerro
        JugueteEsMuyDelicadoException raiseSignal

Sonajero >> teUsaUnGato
        JugueteDebeTenerSaborException raiseSignal

Sonajero >> teUsaUnLoro
         self error: 'cualquiera...'

Pelota >> teUsaUnPerro
        SePuedePincharException raiseSignal

Pelota >> teUsaUnGato
        TajoException raiseSignal

Pelota >> teUsaUnLoro
        PicoRotoException raiseSignal
```

Se sabe que las clases están modeladas en las siguientes jerarquías:



a) Dados los siguientes códigos:

1) Perro new decidir: Sonajero new
2) Gato new decidir: Pelota new
3) Loro new decidir: Sonajero new
4) Perro new decidir: Pelota new
5) Gato new decidir: Sonajero new
6) Loro new decidir: Pelota new

Agregue manejadores para los llamados detallados, teniendo en cuenta que:

- para las JugueteEsMuyDelicadoException se debe escribir en el transcript el mensaje "Se levantó una excepción porque el juguete no es asecuado para el animal " y con el animal donde ocurrió la excepción.
- para las JugueteDebeTenerSaborException se debe levantar un error con el mensaje "Esto parece simple".
- para las SePuedePincharException no se debe hacer nada
- para las TajoException se debe abrir una ventana de inspect el originador de la misma
- para las PicoRotoException se debe relanzar la excepción
- para el resto, se debe enviar un mensaje al transcript con todo el detalle de la excepción que usted considere necesario.

b - ¿Qué debería cambiar en los casos del inciso anterior, si quisiera que no sucedieran UnhandledException's?

c – ¿Podría definirse un manejador de excepción genérico (es decir, que capture cualquier Exception) y obtener un resultado similar al del ejercicio? Describa la solución si es que existe y compárelas.

Ejercicio 6

Modele con objetos un tren, con sus vagones y furgones. Considere que hay dos tipos de trenes: de pasajeros y de carga. El tren de pasajeros sólo permite agregar vagones al final de todos los vagones, mientras que el de carga sólo permite agregar furgones.

Cada vagón o forgón tiene un volumen, y además conoce explícitamente a su anterior y posterior, por lo que se pide modelar e implementar a un tren *sin usar colecciónes*. Considere que los furgones llevan carga mientras que los vagones no.

a) Para cada uno modele operaciones que permitan saber:

- número de vagones/forgones
- capacidad máxima de carga (medido en toneladas)
- velocidad máxima posible (depende de la velocidad de la locomotora que lo impulse, pero además de la resistencia que agrega cada vagón o furgón tirado). Es decir que la velocidad máxima es la velocidad máxima de la locomotora menos la resistencia de cada uno de sus vagones y furgones.
- volumen total

b) cree un tren de pasajeros con 5 vagones y un furgón. Pregunte la capacidad máxima de carga y velocidad máxima.

c) cree un tren de carga, con 15 forgones. Pregunte su capacidad máxima y su volumen total.

Ejercicio 7

Un edificio contiene cocheras y pisos. De las cocheras sólo se sabe quién es el dueño y su tamaño. De los pisos se sabe que están compuestos de departamentos (la cantidad de departamentos es variable según el piso). Los departamentos poseen dueño y tienen varios ambientes. De cada ambiente se sabe consumo de energía, agua y gas, y la superficie ocupada.

Se pide modelar clases en Smalltalk de forma de:

a) poder crear edificios con cocheras y departamentos.

b) poder calcular el consumo de energia, agua y luz del edificio.

c) conocer la unidad más grande.

d) unidades que pertenecen a un determinado dueño.

3) conjunto de dueños de al menos una unidad, sin repetidos.

Ejercicio 8

En una aplicación se necesita un sistema de lógica trinaria. En este sistema existen tres valores de verdad: si, no y no-se. Estos objetos responden, entre otros, a los mensajes:

```
>> siSi: a Block
>> siNo: aBlock siNoSabe: aBlock
```

También existen estos operadores:

Y	si	no	no-se
si	si	no	no-se
no	no	no	no

no-se	no-se	no	no-se

O	si	no	no-se
si	si	no-se	si
no	no-se	no	no-se
no-se	no	no-se	no-se

Implementar las clases y métodos necesarios para poder manejar esta lógica y para responder a los mensajes detallados.

Ejercicio 9

La oficina de posgrados de la Universidad realiza diversos seminarios de actualización para graduados, masters y doctores en tres áreas temáticas: humanidades, ciencias exactas y ciencias biológicas. Los interesados deben inscribirse a dichos seminarios y pagar un arancel. Los aranceles dependen del título que tenga el asistente, y del área que corresponda el curso. La tabla de precios se describe a continuación:

Titulo/area	Humanidades	Ciencias exactas	Ciencias biológicas
Graduado	100	90	80
Master	85	110	50
Doctor	50	150	85

Se pide implementar el mensaje `OficinaDePosgrados>> inscribir: unAsistente a: unSeminario`, el cual retorna el monto que `unAsistente` debe pagar para inscribirse a `unSeminario`.

Ejercicio 10 (Opcional)

Un supermercado de la zona exhibe sus productos a la venta en varias góndolas. Los productos exhibidos son de diferentes tipos (alimentos, librería, limpieza, etc) y de diferentes marcas. El supermercado decide poner ofertas sobre determinados productos de acuerdo a su tipo y su marca, con lo que el precio final de los productos pueden sufrir una modificación. Además el supermercado pone una vez por semana un descuento especial por clientes que pagan con tarjeta de crédito o débito.
Se pide implementar en Smalltalk el supermercado con sus góndolas de productos, cajas y ofertas, y las compras que los clientes pueden hacer en el supermercado de forma que se pueda preguntar el precio total de una compra que realiza un cliente.

Ejercicio 11

La ciudad de "Small Land" posee un sistema de transporte basado en la venta de tickets que se expenden cada dos cuadras usando máquinas. La persona que desea viajar, debe comprar un ticket y con el ticket se sube al transporte que eligió y que la llevará al destino. Para comprar un ticket, la persona debe indicar el destino (el origen se asume que es un lugar no más lejano que cuatro cuadras a la redonda del lugar donde está la máquina), marcándolo en un mapa. A partir del destino elegido, el sistema le muestra los distintos

tipos de transportes que lo pueden llevar al mismo. Los transportes que existen en la ciudad son: colectivo, taxi, subterráneo y tren. El costo final del ticket depende de la distancia que hay del origen al destino, del proporcional por el tipo de transporte, y del proporcional por el tipo de zona donde queda el destino. Los tipos de zonas son: residencial, industrial, comercial, portuaria, costera, y sub-urbana.
Implemente las clases necesarias para que una máquina expendedora pueda generar un ticket o boleto a partir de la especificación del origen, destino y tipo de transporte, considerando que el valor del ticket se calcula de la siguiente manera:
valor = (distancia en km * proporcional de tipo de transporte) * proporcional de tipo de zona
El proporcional para cada tipo de transporte es el siguiente:

Tipo de transporte	Proporcial (%)
Colectivo	80
Taxi	140
Subterráneo	60
Tren	50

Tipo de zona	Proporcial (%)
Residencial	100
Industrial	80
Comercial	150
Portuaria	80
Costera	110
Suburbana	70

Ejercicio 12
Una batalla de pokemones es un enfrentamiento que se da entre dos entrenadores pokemón. Un entrenador puede llevar de 1 a 6 pokemones consigo y podrá usar hasta 3 pokemones por batalla.
Un pokemón tiene un tipo particular (Agua, Fuego, Eléctrico, etc.) que puede causar mayor o menor daño a otro pokemón, dependiendo también del tipo del otro pokemon. Además, tiene una vida inicial de 10 que irá disminuyendo con los ataques del adversario.
La batalla se desarrolla de la siguiente forma:
- Cada entrenador elige entre 1 y 3 pokemones para la batalla.
- Los pokemones atacan por turno, inflingiendo al enemigo un daño que va de 0 al máximo daño que le puede causar al adversario según sus tipos (ver cuadro a continuación). Este valor es aleatorio pero nunca podrá superar el valor máximo de daño posible definido.

Tipo/Ataca a	Planta	Agua	Tierra	Eléctrico	Fuego
Planta	2	3	3	1	1
Agua	1	2	3	1	3
Tierra	1	1	2	3	3
Eléctrico	3	3	1	2	1
Fuego	3	1	1	3	2

- Cuando un pokemón es vencido será reemplazado por otro y continuará la batalla. Un pokemon vencido no podrá atacar en el turno en que haya sido vencido, pero si lo podrá hacer el pokemon que lo reemplace.
Resultará vencedor aquel entrenador que hubiera vencido a todos los pokemones competidores del entrenador adversario.

Se pide modelar las clases necesarias para implementar la batalla de pokemones y mostrar en el transcript el resultado de cada ronda de la siguiente forma:
Ronda: N
- Entrenador 1:
- Pokemon actual: TipoDelPokemonActual
- Nivel de vida: NivelDeVidaDelPokemon
- Potencia del ataque: ValorDeAtaque

- Entrenador 2:
- Pokemon actual: TipoDelPokemonActual
- Nivel de vida: NivelDeVidaDelPokemon
- Potencia del ataque: ValorDeAtaque

Además se deberá informar, también en el transcript, cuándo un pokemón sea vencido y reemplazado por otro y qué entrenador resultó ganador de la batalla.