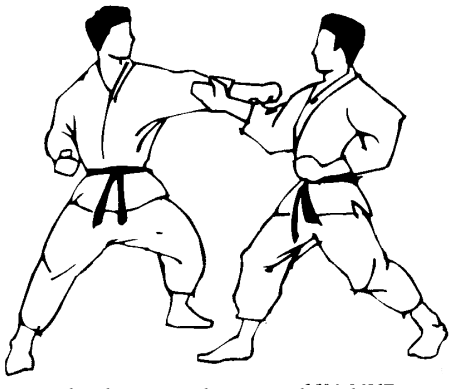
Orientación a Objetos 2 – Práctica 2

Ejercicio 1: Juego de Combate

En un juego de combate marcial participan 2 jugadores. Cada jugador posee una energía vital, la cual varía durante el combate. El combate se organiza en una serie de turnos, en cada turno el jugador decide que movimiento realizar (golpear o bloquear). El combate dura como máximo 10 turnos, y cada jugador comienza el mismo con 100 puntos de energía. Al final del combate puede suceder que:

- Un jugador se quede sin energía, por lo tanto pierde (y gana su oponente).
- Ambos jugadores se quedan sin energía en el mismo turno, en este caso empatan.
- Ambos jugadores terminan el combate con energía mayor que 0. En este caso gana aquel con más energía.



En cada turno los jugadores pueden bloquear o golpear. Por lo tanto se pueden dar 3 casos:

- El jugador A golpea y el jugador B bloquea: en este caso el que golpea (A) pierde 10 puntos de energía.
- Ambos jugadores bloquean: en este caso la energía de ambos se mantiene igual.
- Ambos golpean: el jugador A resta de su energia el 20 % de la energía que tiene B, y el jugador B resta el 20 % de la energía que tiene A.

A modo de ejemplo considere la siguiente secuencia de turnos:

Turno	Mov Jug A	Mov Jug B	Energía A	Energía B
0	-	-	100	100
1	Bloqueo	Bloqueo	100	100
2	Bloqueo	Golpe	100	90
3	Golpe	Golpe	82	70

Usted debe implementar un sistema (clase Juego) donde, dados 2 jugadores que pueden responder con un movimiento para cada turno, determine quién gana o si hubo empate. El resultado del combate se debe informar mediante el `Transcript` con el mensaje `#show:` que toma un string como parámetro. Su implementación debe respetar las siguiente indicaciones:

- La clase `Jugador` debe entender el mensaje `#jugadaParaTurno: unTurno` (donde unTurno es un entero entre 1 y 10) que retorna la jugada para ese turno.
- La clase `Jugador` debe entender el mensaje `#nombre` que retorna el nombre del mismo.
- La clase `Juego` debe implementar el mensaje: `#determinarGanadorEntreJugador: unJugador y: otroJugador`

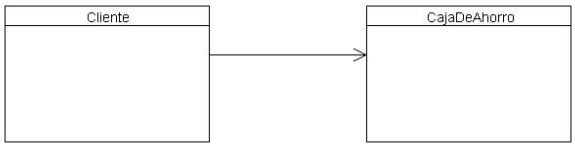
Tarea

1. Realice el diagrama de clases UML.
2. Realice un diagrama de secuencia UML donde se muestre el procesamiento de un golpe contra un bloqueo.
3. Implemente completamente en Smalltalk utilizando test cases. Recomendamos desarrollar primero los test cases y luego el modelo.

Ejercicio 2: UML 1

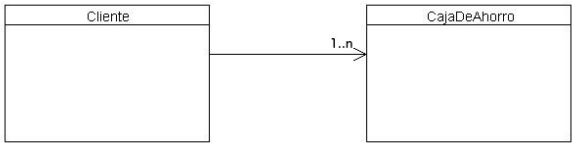
Sea una aplicación bancaria en la cual existen `Clientes` y `CajasDeAhorro`. Para cada uno de los diagramas que se muestran a continuación indique la interpretación correcta del mismo, y justifique indicando en que elemento del diagrama se basa para su elección.

1. Caso 1:



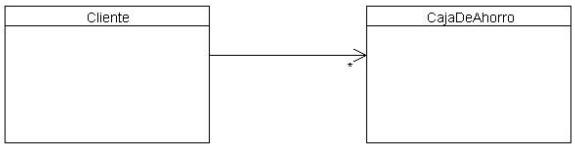
   - Un `Cliente` debe estar asociado a una `CajaDeAhorro`.
   - Un `Cliente` debe estar asociado a ninguna, una o más de una `CajaDeAhorro`.

2. Caso 2:



   - Un `Cliente` debe estar asociado a una `CajaDeAhorro`.
   - Un `Cliente` debe estar asociado a una o más de una `CajaDeAhorro`.

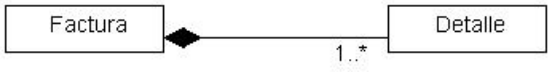
3. Caso 3



   - Un `Cliente` debe estar asociado a una `CajaDeAhorro`.
   - Un `Cliente` debe estar asociado a niguna, una o más de una `CajaDeAhorro`.

Ejercicio 3: UML 2

Sea una aplicación de facturación que modela las `Facturas` y el `Detalle` (renglones) de la misma, en donde el `Detalle` no tienen sentido si no existe la `Factura` a la cual pertenece. Indique cual de los siguientes diagramas es el correcto.



(a) Diseño 1

(b) Diseño 2

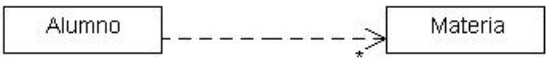
Ejercicio 4: UML

1. Sea una aplicación que modela un `Auto` y sus `Ruedas`. En este caso, las `Ruedas` pueden existir sin el `Auto`. Indique cual de los siguientes diagramas es el correcto.

(a) Diseño 1

(b) Diseño 2

2. Sea una aplicación que modela `Alumnos` y sus `Materias`. Indique cual es el trazo correcto para la asociación.

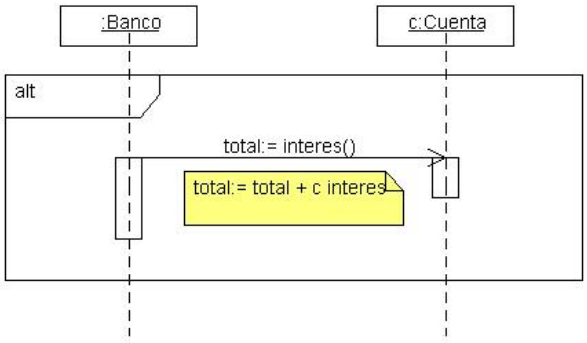


(a) Diseño 1

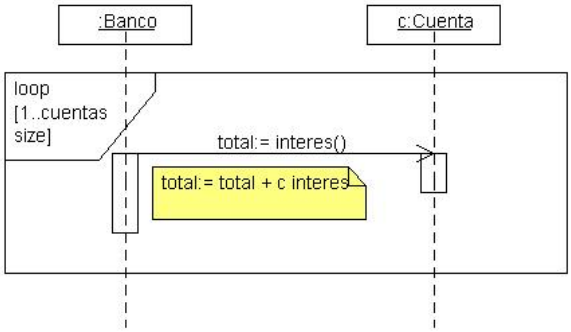
(b) Diseño 2

Ejercicio 5: UML 5

Sea un diagrama de secuencia que refleja el hecho de que el `Banco` calcula el total de intereses en base al interés de cada una de sus `CajasDeAhorro`. Indique cual es el diagrama correcto.



(a) Diseño 1



(b) Diseño 2

Ejercicio 6: Modele usando UML

En Twitter los usuarios registrados pueden postear y leer mensajes de 140 caracteres. Ud. debe modelar parte del sistema donde nos interesa que quede claro lo siguiente:

- Cada usuario conoce todo los Tweets que hizo.
- Un tweet puede ser re-tweet de otros, y este tweet debe conocer a su tweet de origen.
- Twitter debe conocer a todos los usuarios del sistema.
- Los tweets de un usuario se deben eliminar cuando el usuario es eliminado. No existen tweets no referenciados por un usuario.

Realice un diagrama UML que cumpla con la especificación anterior.

Ejercicio 7: Implemente en Smalltalk

A partir del enunciado anterior considere los siguientes requerimientos adicionales:

- Los usuarios se identifican por su *screenName*.
- No se pueden agregar dos usuarios con el mismo screenName.
- Los tweets deben tener un texto de 1 caracter como mínimo 140 caracteres como máximo.

Tareas:

1. Diseñe y discuta con un ayudante los test cases que considere relevantes respecto de los requerimientos del ej. 7 y 8.
2. Implemente los test cases.
3. Implemente las claseses Twitter, User y Tweet junto con todo lo necesario para satisfacer los requerimientos anteriores.

Ejercicio 8: Patrón Adapter

Lea el capítulo 4 del libro Design Patterns de Gamma et al. y responda a las siguientes preguntas:

1. ¿Qué es un patrón estructural?
2. El capítulo menciona dos formas de implementar el patrón Adapter: como patrón estructural de clase y como patrón estructural de objeto. ¿Cuáles son esas dos formas? ¿Cuál de ellas no es implementable en Smalltalk? ¿Porqué?