Orientación a Objetos 2 – Práctica 1

Ejercicio 1: Evaluación del protocolo de una clase

Sea la clase `Rectangulo` con 4 variables de instancia (esquinaSuperiorIzquierda, esquinaSuperiorDerecha, esquinaInferiorIzquierda y esquinaInferiorDerecha). Elija uno de los protocolos presentados a continuación y justifique su elección.

Opción 1)

```
Class Rectangulo>>new
Rectangulo>>esquinaSuperiorIzquierda: unPunto
Rectangulo>>esquinaSuperiorDerecha: unPunto
Rectangulo>>esquinaInferiorIzquierda: unPunto
Rectangulo>>esquinaInferiorDerecha: unPunto
```

Opción 2)

```
Class Rectangulo>>newEnOrigenEsquinaSuperiorIzquierda: unPunto
      alto: unNumero ancho:unNumero
Rectangulo>>reubicarEsquinaSuperiorIzquierdaEn: unPunto
Rectangulo>>ancho: unNumero
Rectangulo>>alto: unNumero
```

Ejercicio 2: Delegación

En una oficina hay un jefe que tiene una secretaria quien debe administrar un fichero. Dadas las siguientes implementaciones seleccione la mejor y justifique.

Opción 1)

```
Jefe>>algunMetodo: unaFicha
      self secretaria fichero buscar: unaFicha
```

Opción 2)

```
Jefe>>algunMetodo: unaFicha
      self secretaria buscarEnFichero: unaFicha

Secretaria>>buscarEnFichero: unaFicha
      ^self fichero buscar: unaFicha
```

Ejercicio 3: Polimorfismo

Sean las cuentas bancarias vistas en Objetos 1: `CuentaBancaria` con las sublcases: `CajaDeAhorro` y `CuentaCorriente`. Indique los defectos de cada una de las opciones.

Opción 1)

```
CuentaBancaria>>extraer:unMonto
	|rojo|
	self class = CuentaCorriente
		ifTrue:[rojo:= self rojoPermitido.]
		ifFalse:[rojo := 0.]
	(self saldo + rojo >= unMonto)
		ifTrue: [self saldo: self saldo - unMonto].
```

Opción 2)

```
CuentaBancaria>>extraer:unMonto
	|rojo|
	self tipo = ’cuentacorriente’
		ifTrue:[rojo:= self rojoPermitido.]
		ifFalse:[rojo := 0.]
	(self saldo + rojo >= unMonto)
		ifTrue: [self saldo: self saldo - unMonto].
```

Opción 3)

```
CuentaBancaria>>extraer:unMonto
	^self subclassResponsibility

CajaDeAhorro>>extraer:unMonto
	(self saldo >= unMonto)
		ifTrue: [self saldo: self saldo - unMonto].

CuentaCorriente>>extraer:unMonto
	(self saldo + self rojoPermitido >= unMonto )
		ifTrue: [self saldo: self saldo - unMonto].
```

Opcion 4)

```
CuentaBancaria>>extraer:unMonto
	(self chequearSaldoParaExtraccion:unMonto)
		ifTrue: [self saldo: self saldo - unMonto].
CajaDeAhorro>>chequearSaldoParaExtraccion:unMonto
	^self saldo >= unMonto
CuentaCorreinte>>chequearSaldoParaExtraccion:unMonto
	^self saldo + self rojoPermitido >= unMonto
```

Ejercicio 4: Manejo de colecciones

Se desea encontrar el menor elemento de una colección de números. Indique para cada implentación todos los defectos encontrados, si los hubiera.

Opción 1)

```
UnaClase>>buscarMenorEn: unaColeccion
	|minimo|
	minimo:= 9999.
	unaColeccion do: [:i | (minimo>i) ifTrue: [minimo:= i.]].
	^minimo.
```

Opción 2)

```
UnaClase>>buscarMenorEn: unaColeccion
	|minimo|
	minimo:= 9999.
	unaColeccion do: [:i | minimo:= (minimo min: i).].
	^minimo.
```

Opción 3)

```
UnaClase>>buscarMenorEn: unaColeccion
	|minimo |
	minimo:= 9999.
	actual:= 1.
	[unaColeccion size >= actual]
		whileTrue:[minimo:= minimo min: (unaColeccion at: actual).
			actual := actual + 1. ].
	^minimo
```

Opción 4)

```
UnaClase>>buscarMenorEn: unaColeccion
	|minimo |
	minimo:= 9999.
	1 to: unaColeccion size do: [ :actual |
		minimo:= minimo min: (unaColeccion at: actual). ].
	^minimo
```

Implemente una solución que no presente ninguno de los problemas encontrados.

Ejercicio 5: Double dispatch

Repase la técnica de Double dispatch del libro Smalltalk Best Practices Patterns
http://stephane.ducasse.free.fr/FreeBooks/BestSmalltalkPractices/Draft-Smalltalk%20Best%20Practice%20Patterns%20Kent%20Beck.pdf

Tareas:

1. ¿En qué casos es provechoso utilizar esta técnica?
2. ¿Cómo se aplica la técnica?

Ejercicio 6: 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`

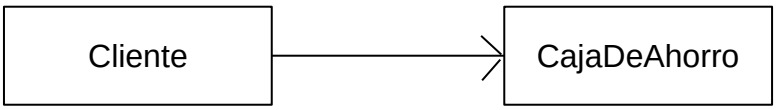
Tarea

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 7: UML 1

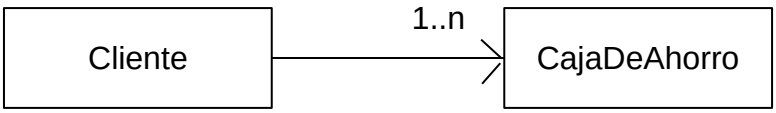
Sea una aplicación bancaria en la cual existen `Clientes` y `CajasDeAhorro`. Para cada uno de los diagramas que se muestran a continuación indique la interpretación correcta del mismo, y justifique indicando en qué elemento del diagrama se basa para su elección.

1. Caso 1:



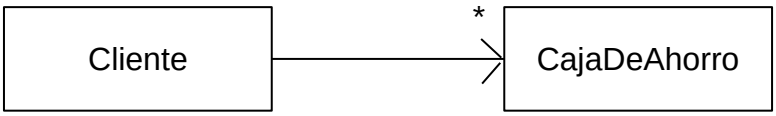
- Un `Cliente` debe estar asociado a una `CajaDeAhorro`.
- Un `Cliente` debe estar asociado a ninguna, una o más de una `CajaDeAhorro`.

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:

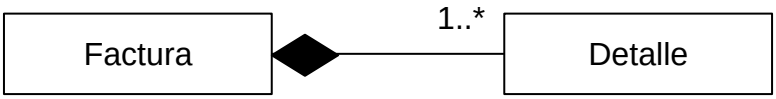


- Un `Cliente` debe estar asociado a una `CajaDeAhorro`.
- Un `Cliente` debe estar asociado a ninguna, una o más de una `CajaDeAhorro`.

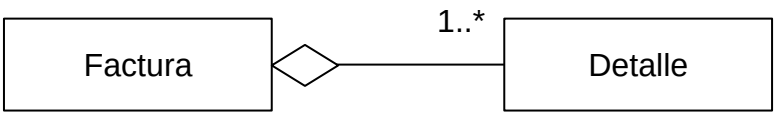
Ejercicio 8: UML 2

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 cuál de los siguientes diagramas es el correcto.

1. Diseño 1:



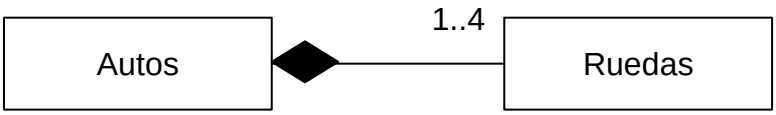
2. Diseño 2:



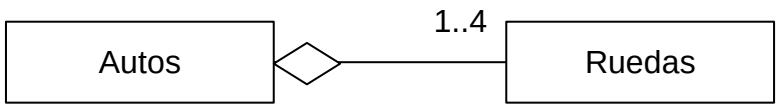
Ejercicio 9: UML 3

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

1. Diseño 1:

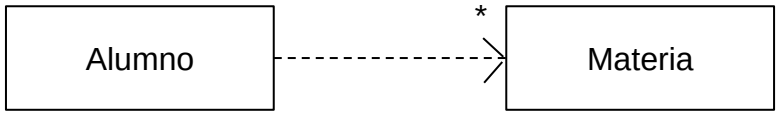


2. Diseño 2:

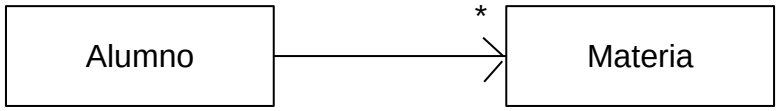


Sea una aplicación que modela `Alumnos` y sus `Materias`. Indique cual es el trazo correcto para la asociación

1. Diseño 1:



2. Diseño 2:



Ejercicio 10: 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.

Tarea

1. Realice un diagrama de clases UML para satisfacer la especificación anterior.
2. Lea la sección “Class Package Diagrams” del siguiente link http://www.agilemodeling.com/artifacts/packageDiagram.htm
3. Analice y discuta con el ayudante la posibilidad de incorporar Packages en el diagrama previo.

Ejercicio 11: 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 y 140 caracteres como máximo.

Tareas

1. Diseñe y discuta con un ayudante los test cases que considere relevantes.
2. Implemente los test cases.
3. Implemente las clases `Twitter`, `User` y `Tweet` junto con todo lo necesario para satisfacer los requerimientos anteriores.